



Algorithmen und Datenstrukturen

Tutorium V

Michael R. Jung

18. - 23. 05. 2016



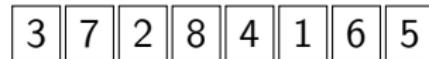


1 Sortieralgorithmen

- SelectionSort
- InsertionSort
- BubbleSort
- MergeSort
- QuickSort



An folgender Beispielinstanz werden einige Sortierverfahren veranschaulicht:

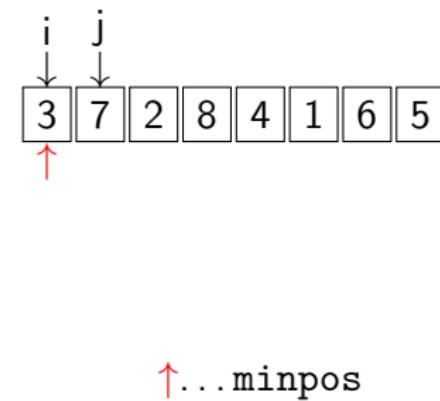


Sortieralgorithmen



SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```



Sortieralgorithmen

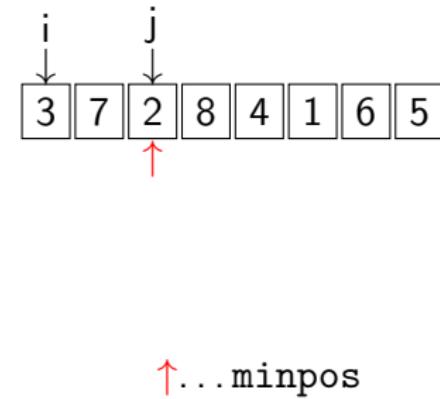
A horizontal sequence of 100 circles. The first circle is solid black, and the remaining 99 circles are hollow with a thin black outline.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



Sortieralgorithmen

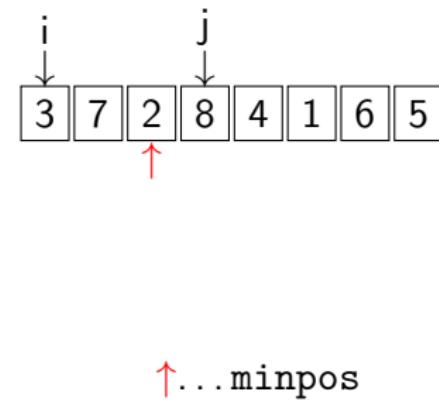
A horizontal sequence of 100 circles. The first circle is filled black, while all others are white with black outlines.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



Sortieralgorithmen

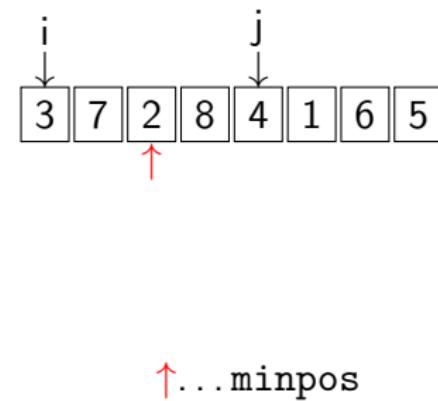
A horizontal sequence of 100 circles, with the 50th circle being black and all others white.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



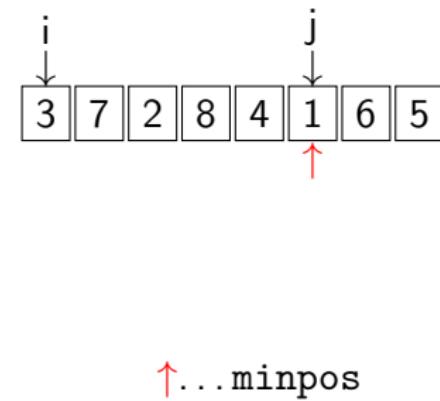
Sortieralgorithmen

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```

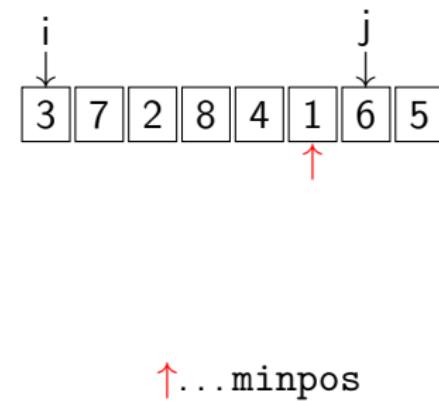


Sortieralgorithmen

ooooooooooooo●ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

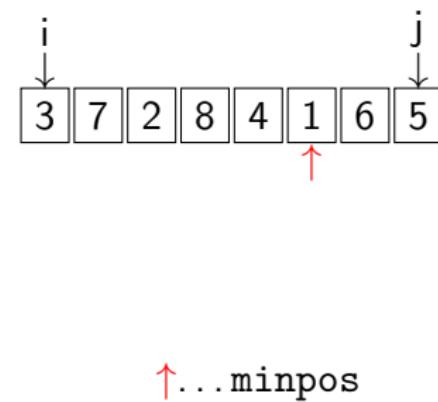


Sortieralgorithmen

oooooooo●ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

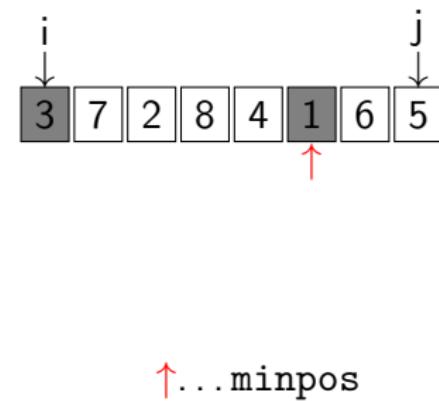


Sortieralgorithmen

oooooooo●ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo
ooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

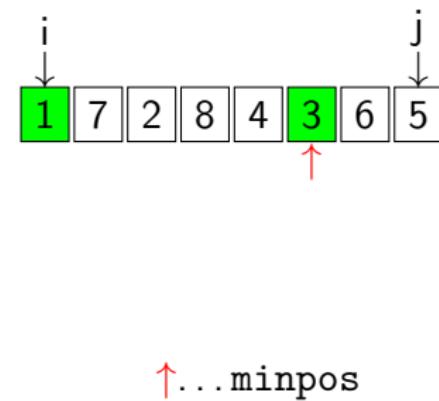


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooooooooooo
oooooooooooooooooooooooooooo
oooooooooooooooooooooooooooo
oooooooooooooooooooo
oooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

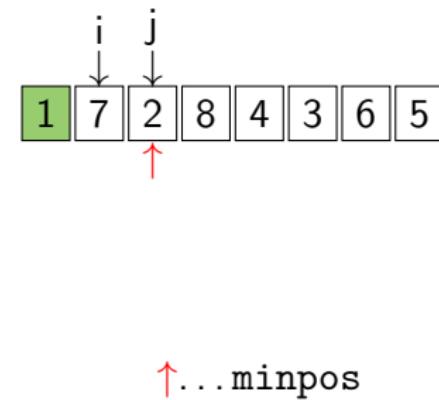


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```



Sortieralgorithmen

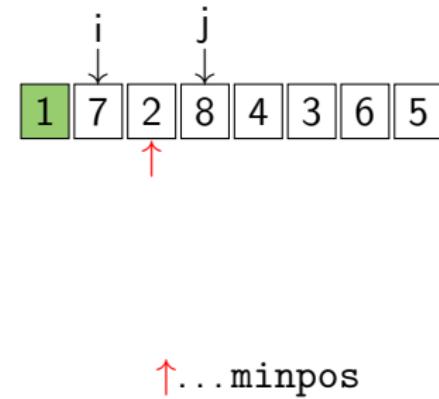
A horizontal sequence of 100 circles arranged in a single row. The 50th circle from the left is filled black, while all other circles are white with black outlines.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



Sortieralgorithmen

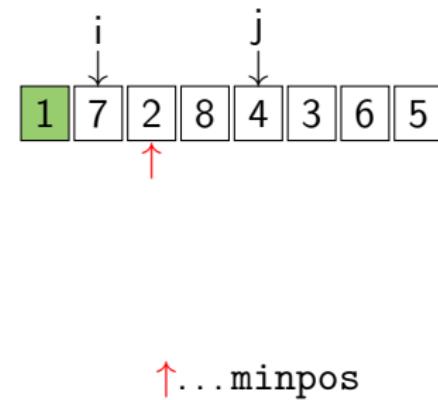
A horizontal sequence of 100 circles arranged in a single row. The circles are white with black outlines, except for the 50th circle which is solid black. This visual representation likely corresponds to the 'black' character in the sequence of 100 elements.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```

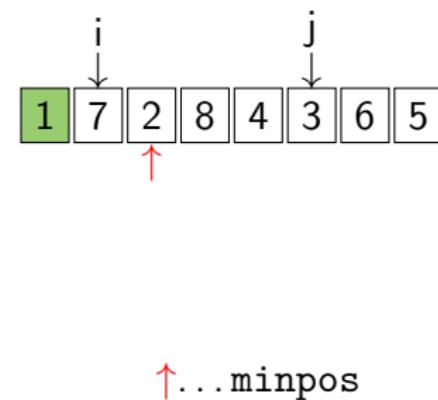


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

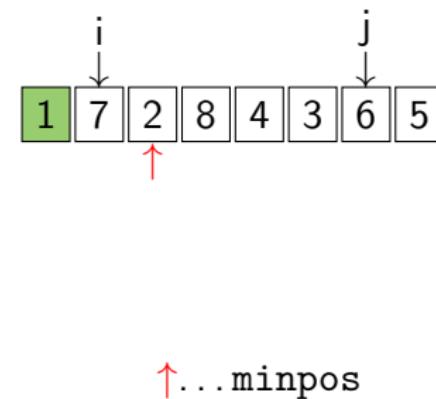


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

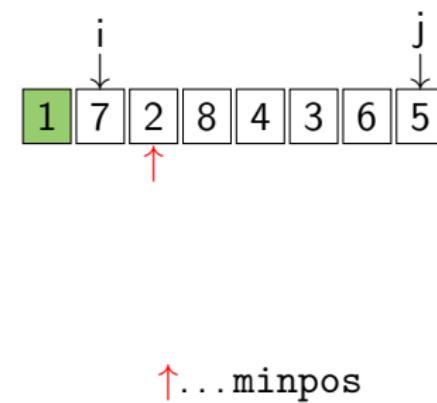


Sorteralgorithmen

oooooooooooo●oooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

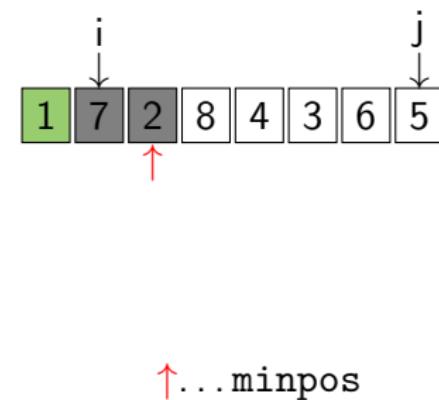


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

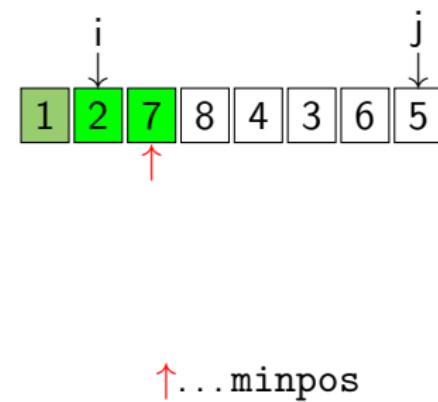


Sortieralgorithmen

oooooooooooooo●oooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

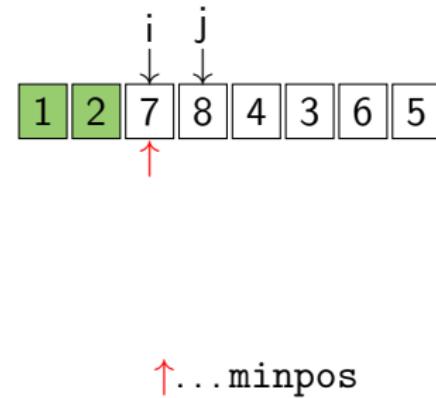


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

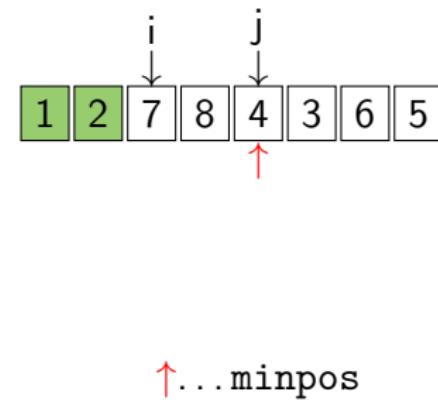


Sorteralgorithmen

oooooooooooo●oooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

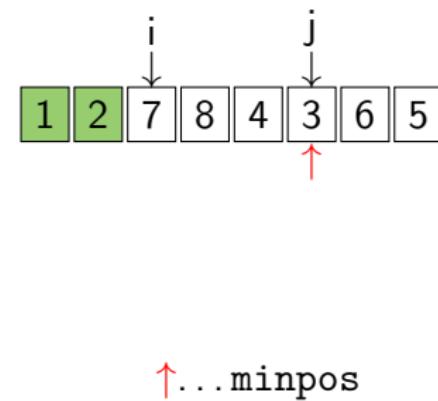


Sortieralgorithmen

oooooooooooo●oooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

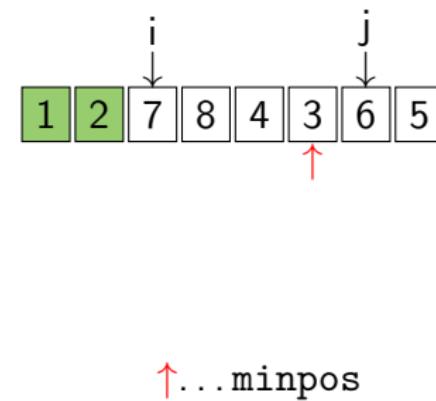


Sortieralgorithmen

oooooooooooooooo●oooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

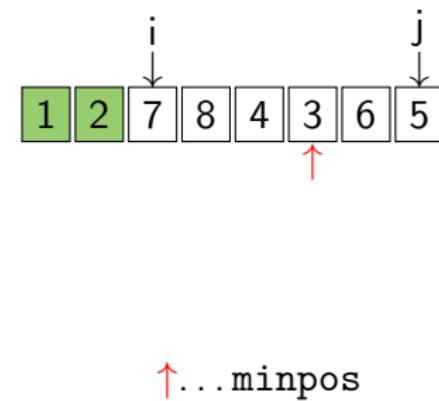


Sortieralgorithmen

oooooooooooooooo●oooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

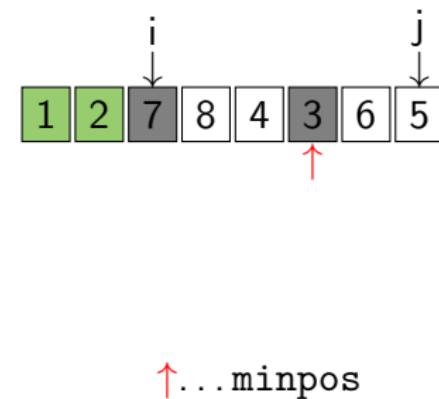


Sortieralgorithmen

oooooooooooooooo●oooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```



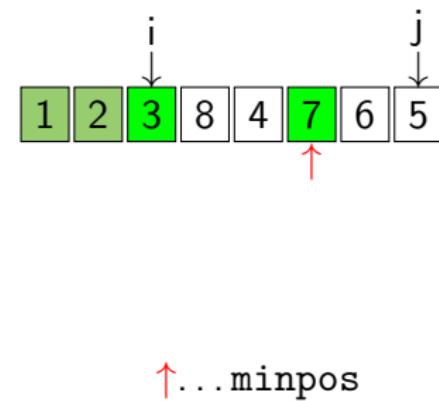
Sortieralgorithmen

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```

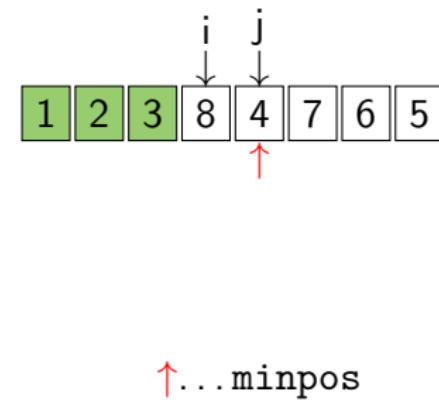


Sortieralgorithmen

oooooooooooooooooooo●oooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

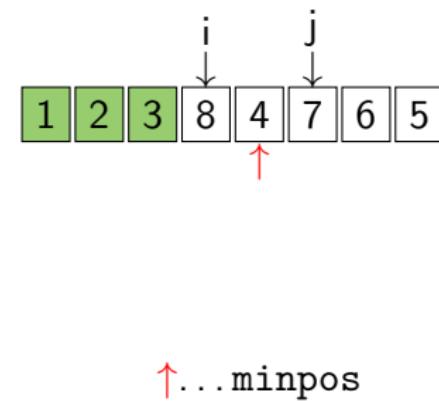


Sortieralgorithmen

oooooooooooooooooooo●oooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

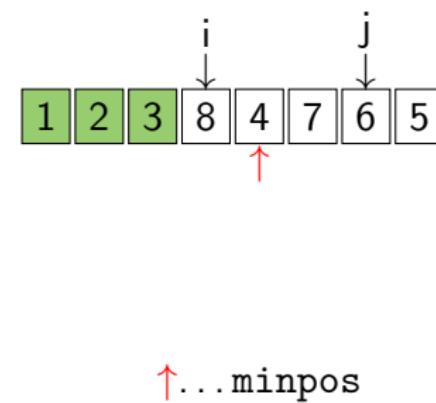


Sortieralgorithmen

oooooooooooooooooooo●ooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

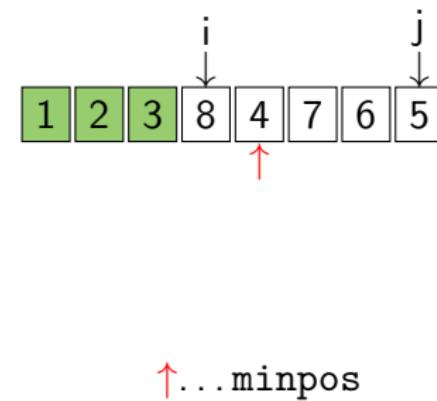


Sorteralgorithmen

oooooooooooooooooooooooo●oooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

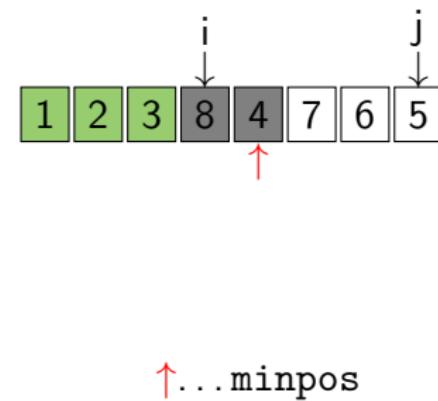


Sortieralgorithmen

oooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

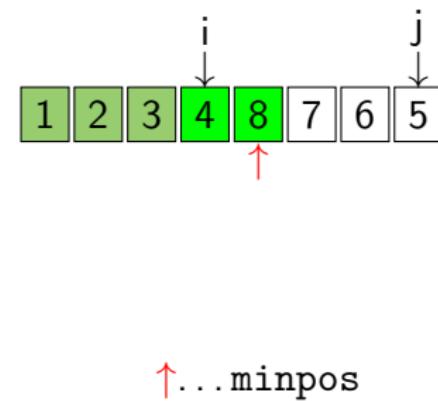


Sorteralgorithmen

oooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

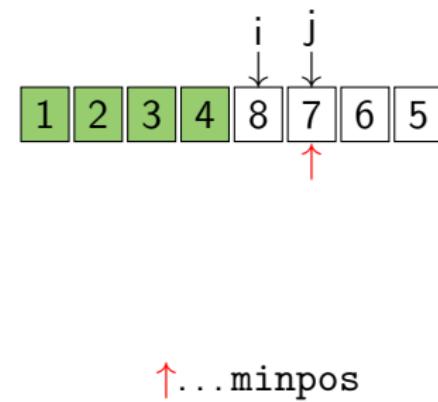


Sortieralgorithmen

oooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos]>S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

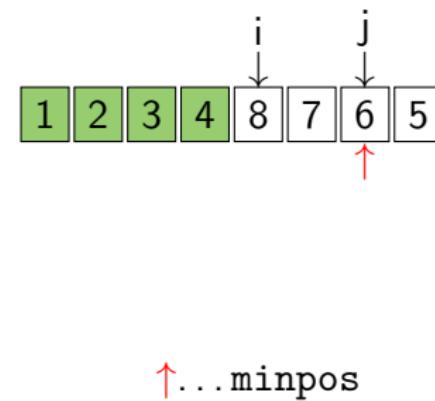


Sortieralgorithmen

oooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

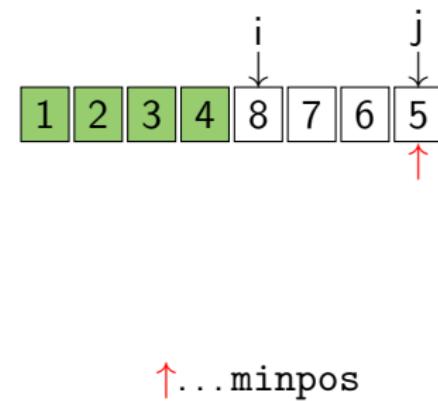


Sortieralgorithmen

oooooooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos]>S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

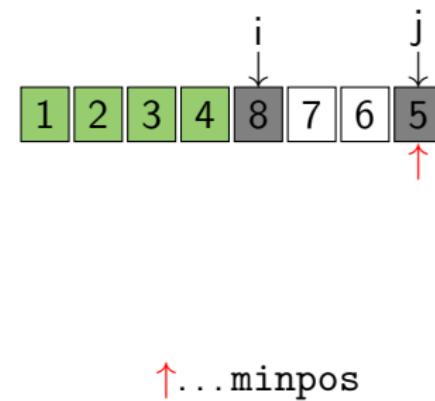


Sortieralgorithmen

oooooooooooooooooooooooooooo●oooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos]>S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

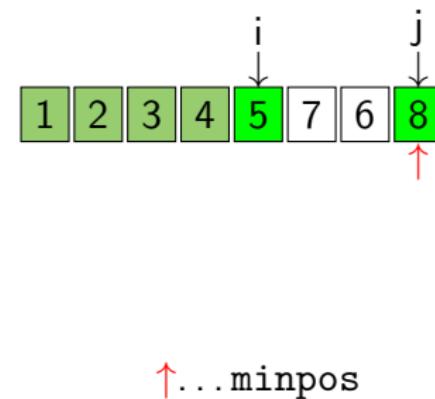


Sorteralgorithmen

oooooooooooooooooooooooooooo●ooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos]>S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

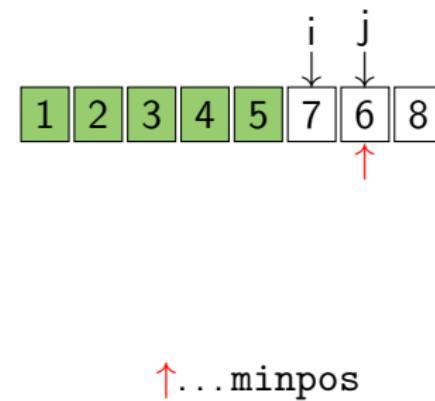


Sorteralgorithmen

oooooooooooooooooooooooooooo●oooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

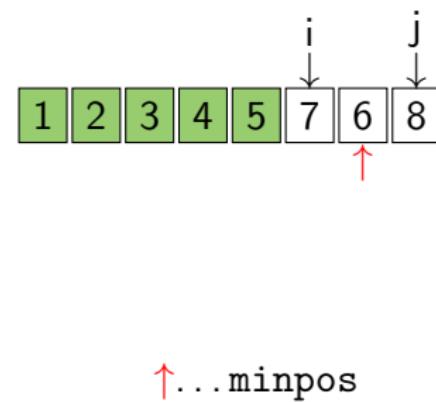


Sorteralgorithmen

oooooooooooooooooooooooooooooooo●oooo
oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```



Sortieralgorithmen

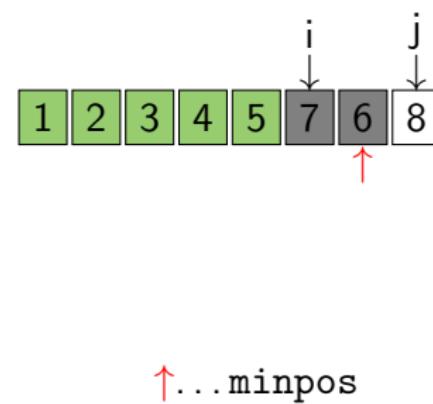
A rectangular grid of small, light blue circles arranged in horizontal rows. The grid has approximately 10 columns and 10 rows. A single dark blue circle is located at the top-right corner of the grid.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



Sortieralgorithmen

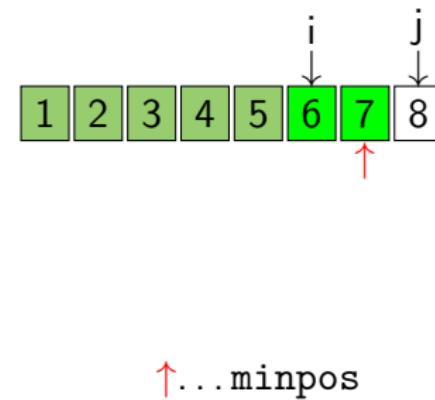
A horizontal sequence of 100 small circles arranged in a grid pattern. The circles are white with black outlines, set against a dark blue background. They are organized into approximately 10 rows and 10 columns, creating a visual representation of a large dataset or a specific algorithmic state.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```



Sortieralgorithmen

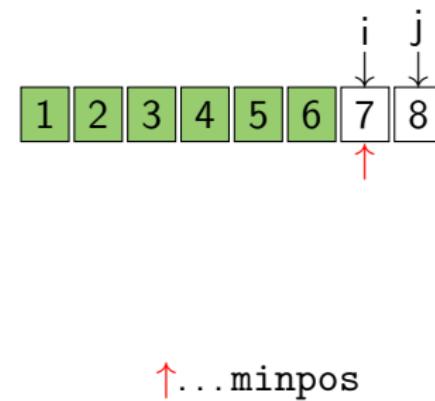
A horizontal sequence of 100 small circles, with the last circle being solid black.

SelectionSort

```

S: array_of_names;
n := |S|
for i = 1..n-1 do
    min_pos := i;
    for j = i+1..n do
        if S[min_pos]>S[j] then
            min_pos := j;
        end if;
    end for;
    tmp := S[i];
    S[i] := S[min_pos];
    S[min_pos] := tmp;
end for;

```

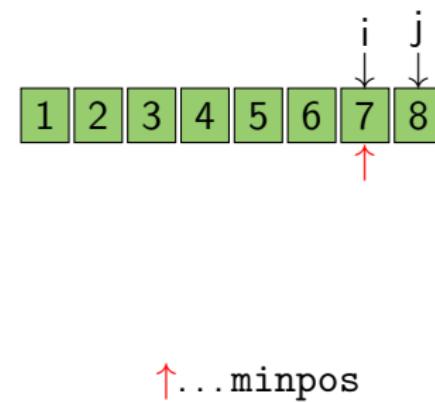


Sorteralgorithmen

oooooooooooooooooooooooooooooooooooo●
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooo

SelectionSort

```
S: array_of_names;  
n := |S|  
for i = 1..n-1 do  
    min_pos := i;  
    for j = i+1..n do  
        if S[min_pos] > S[j] then  
            min_pos := j;  
        end if;  
    end for;  
    tmp := S[i];  
    S[i] := S[min_pos];  
    S[min_pos] := tmp;  
end for;
```

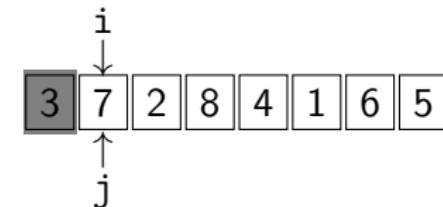


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

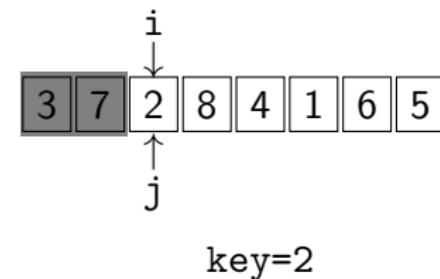


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

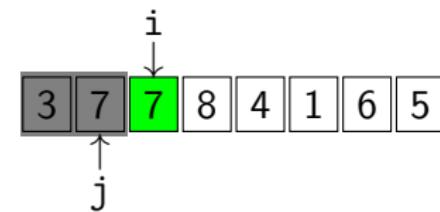


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



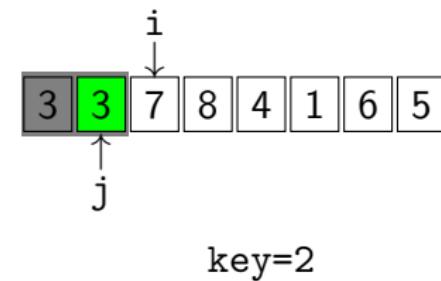
Sortieralgorithmen

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
    j := i;
    key := S[j];
    while (S[j-1]>key) and (j>1) do
        S[j] := S[j-1];
        j := j-1;
    end while;
    S[j] := key;
end for;

```

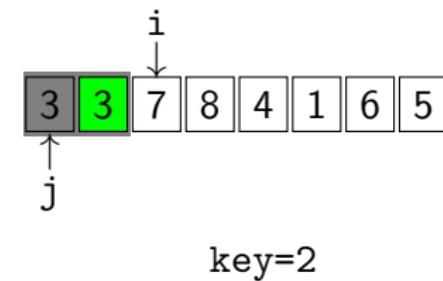


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen

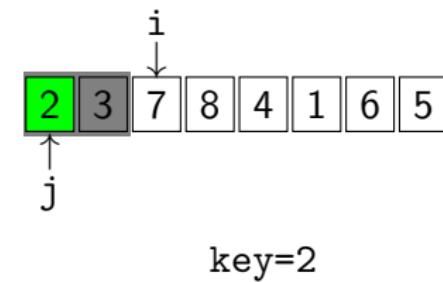
A horizontal sequence of small circles representing data points. The first circle is black, while all others are white.

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
    j := i;
    key := S[j];
    while (S[j-1]>key) and (j>1) do
        S[j] := S[j-1];
        j := j-1;
    end while;
    S[j] := key;
end for;

```

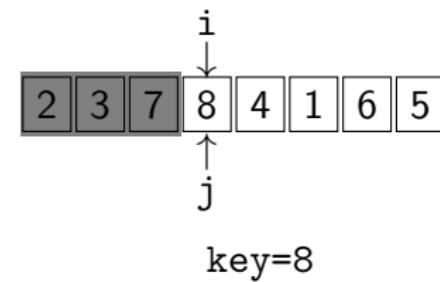


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen

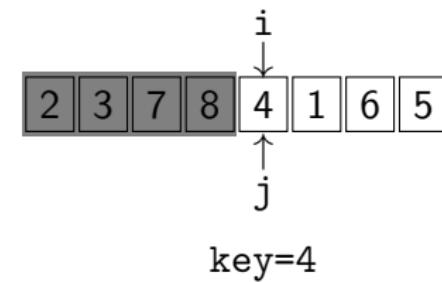
A horizontal sequence of 100 circles arranged in a single row. The circles are white with black outlines, except for the 10th circle from the left which is filled black.

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
    j := i;
    key := S[j];
    while (S[j-1]>key) and (j>1) do
        S[j] := S[j-1];
        j := j-1;
    end while;
    S[j] := key;
end for;

```

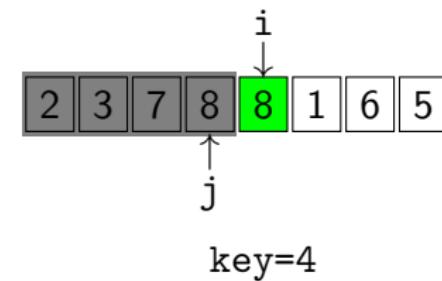


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

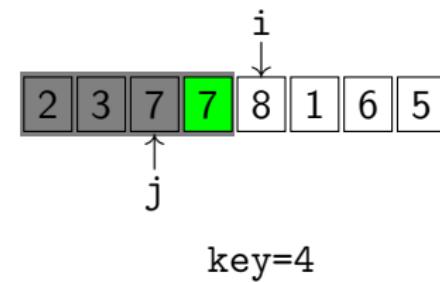


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

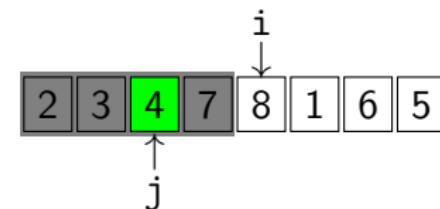


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



key=4

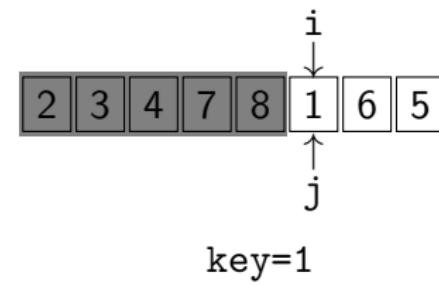


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

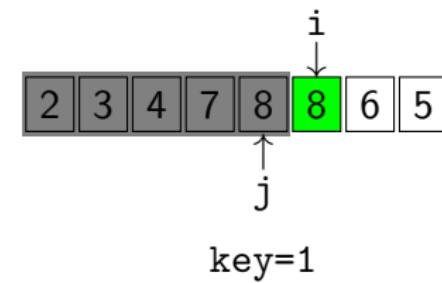


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen

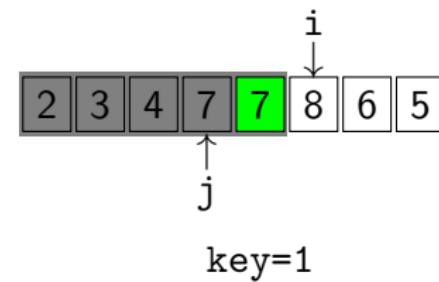
A horizontal sequence of 100 circles, with the 50th circle being solid black and all others being hollow white circles with black outlines.

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;

```



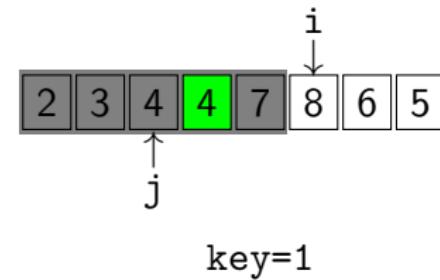
Sortieralgorithmen

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
  j := i;
  key := S[j];
  while (S[j-1]>key) and (j>1) do
    S[j] := S[j-1];
    j := j-1;
  end while;
  S[j] := key;
end for;

```

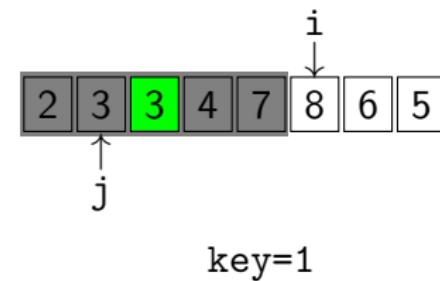


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen

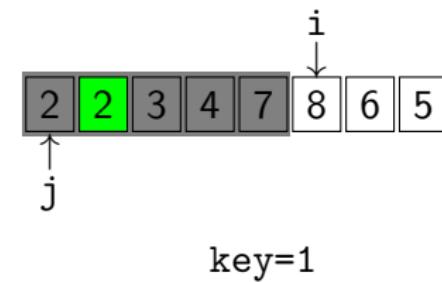
A horizontal sequence of small circles, some filled black and some white, representing a binary vector. The pattern starts with a black circle, followed by several white circles, then a black circle, and so on, creating a repeating sequence.

InsertionSort

```

S: array_of_names;
n := |S|
for i = 2..n do
    j := i;
    key := S[j];
    while (S[j-1]>key) and (j>1) do
        S[j] := S[j-1];
        j := j-1;
    end while;
    S[j] := key;
end for;

```

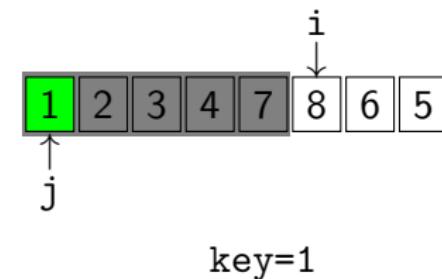


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

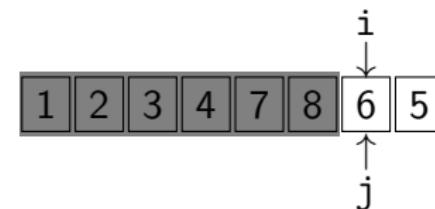


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

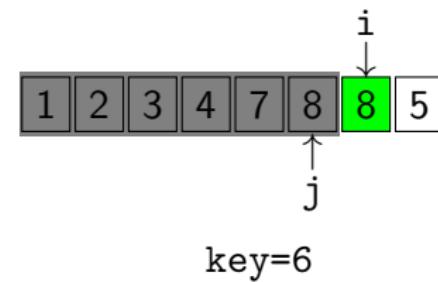


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

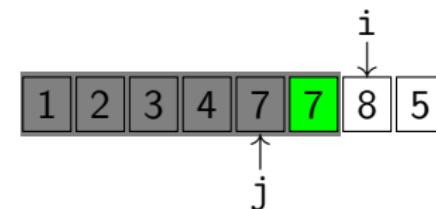


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



key=6

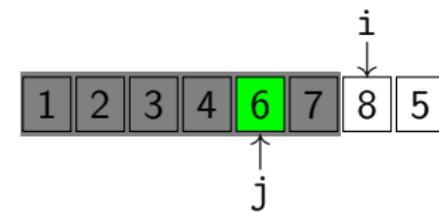


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



key=5



Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

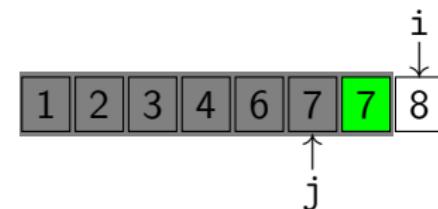


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



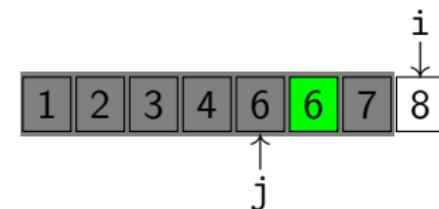
key=5

Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



key=5

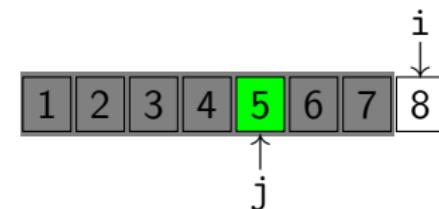


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```

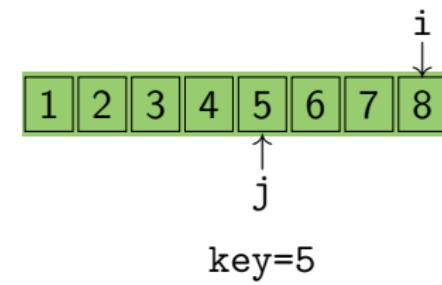


Sortieralgorithmen



InsertionSort

```
s: array_of_names;  
n := |s|  
for i = 2..n do  
    j := i;  
    key := s[j];  
    while (s[j-1]>key) and (j>1) do  
        s[j] := s[j-1];  
        j := j-1;  
    end while;  
    s[j] := key;  
end for;
```



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

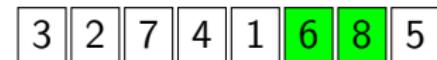


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

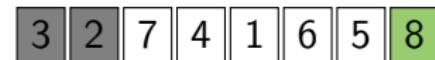


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

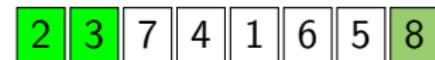


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

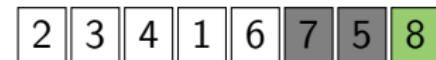


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

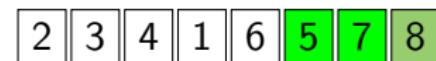


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

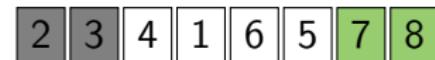


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

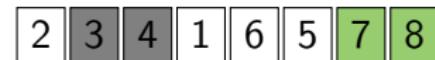


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

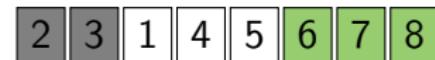


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

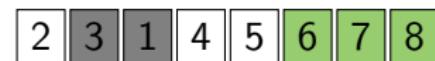


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

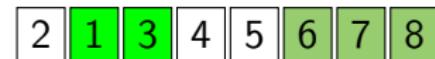


Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps



Sortieralgorithmen



BubbleSort

- Go through array again and again
- Compare all **direct neighbors**
- Swap if in wrong order
- Repeat until a loop finishes without a single swaps

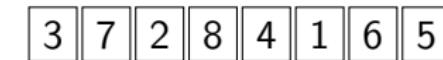


Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
●oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
    end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := 1;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$$l=0$$

$$r=7$$

$$m=3$$



Sortieralgorithmen

MergeSort

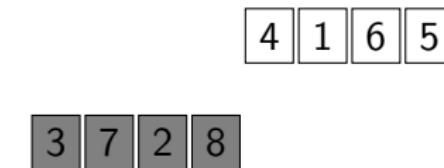
```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-1) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer)
  B: array[l..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := l;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then      # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=0

r=3

m =



Sortieralgorithmen

A horizontal sequence of 100 circles arranged in a single row. The circles are white with black outlines, except for the 10th circle from the left which is solid black.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]≤S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 =

r=:

m=0



Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left filled black.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i] <= S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then                # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 = 0

r=0



Sortieralgorithmen

MergeSort

```

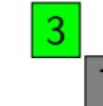
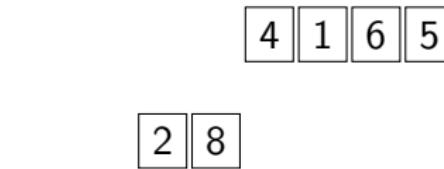
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then                # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



] =

r=



Sorteralgorithmen

```
ooooooooooooooooooooooo  
ooooooooooooooooooooooo  
ooooooooooooooooooooooo  
oooooooo●ooooooooooooooo  
ooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
    end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$l=0$

$r=1$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooo●oooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;      # Start of 1st list  
    j := m+1;    # Start of 2nd list  
    k := 1;      # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;      # Next target  
    end while;  
    if i>m then    # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$l=0$

$r=1$

$m=0$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooo●oooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$l=0$

$r=1$

$m=0$



Sortieralgorithmen

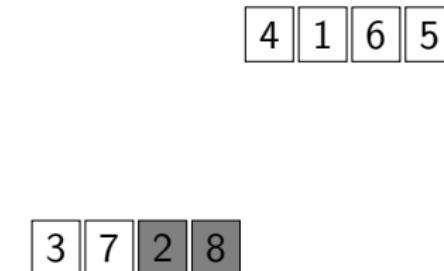
MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-1) div 2;
    mergesort( S, l, 1+m);
    mergesort( S, 1+m+1, r);

    #merges two sorted lists
    merge( S, l, 1+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}
function void merge(S array;
                    l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i=m then        # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=2

r=3

m=0



Sortieralgorithmen

A horizontal sequence of 20 small circles arranged in a single row. The 11th circle from the left is filled black, while all other circles are unfilled.

MergeSort

```

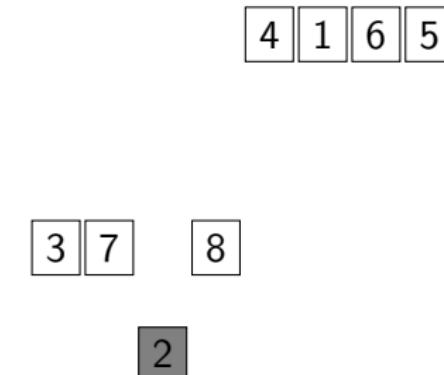
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]≤S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=2

r=2



Sortieralgorithmen

MergeSort

```

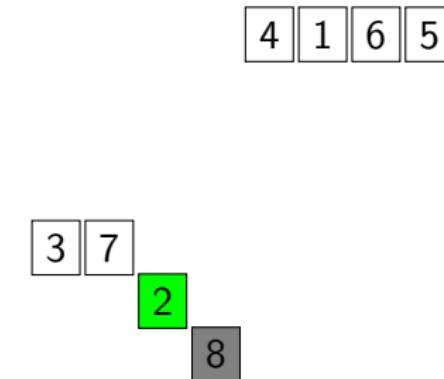
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := 1;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=3

r=3



Sortieralgorithmen

A horizontal sequence of 20 small circles arranged in a single row. The 11th circle from the left is filled black, while all other circles are unfilled.

MergeSort

```

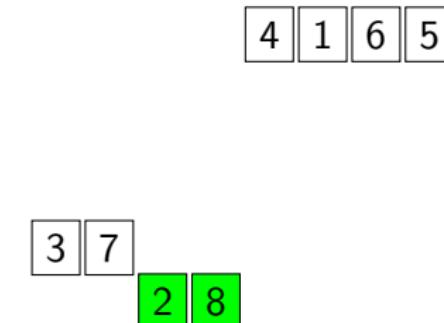
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := 1;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=2

r=3

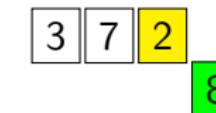


Sortieralgorithmen

oooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo
oooooooooooooooo●oooooooooooooooooooo
oooooooooooooooooooooooooooooooooooo

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
    end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;      # Start of 1st list  
    j := m+1;    # Start of 2nd list  
    k := 1;      # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;      # Next target  
    end while;  
    if i>m then    # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$$l=2$$

$$r=3$$

$$m=0$$



Sortieralgorithmen

MergeSort

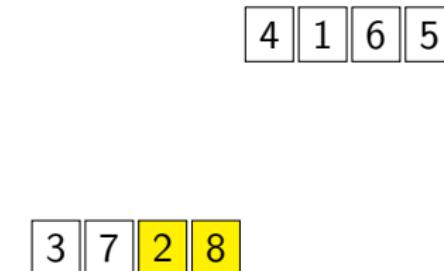
```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-1) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, 1+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[l..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i=m then      # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=

r=3

m =



Sortieralgorithmen

MergeSort

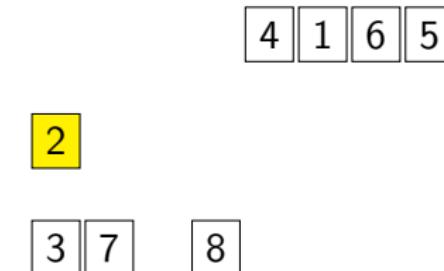
```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]≤S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=

r=3

$$m =$$



Sortieralgorithmen



MergeSort

```
function void mergesort(S array;
                        l,r integer) {
    if (l<r) then
        #Sort each ~50% of array
        m := (r-1) div 2;
        mergesort( S, l, l+m);
        mergesort( S, l+m+1, r);

        #merges two sorted lists
        merge( S, l, l+m ,r);
    else
        # Nothing to do, 1-element list
        end if;
}

function void merge(S array;
                     l,m,r integer) {
    B: array[1..r-l+1];
    i := l;          # Start of 1st list
    j := m+1;        # Start of 2nd list
    k := 1;          # Target list
    while (i<=m) and (j<=r) do
        if S[i]<=S[j] then
            B[k] := S[i]; # From 1st list
            i := i+1;
        else
            B[k] := S[j]; # From 2nd list
            j := j+1;
        end if;
        k := k+1;        # Next target
    end while;
    if i>m then      # What remained?
        copy S[j..r] to B[k..k+r-j];
    else
        copy S[i..m] to B[k..k+m-i];
    end if;
    # Back to original list
    copy B[1..r-l+1] to S[l..r];
}
```



$$l=0$$

$$r=3$$

$$m=1$$



Sortieralgorithmen



MergeSort

```
function void mergesort(S array;
                        l,r integer) {
    if (l<r) then
        #Sort each ~50% of array
        m := (r-1) div 2;
        mergesort( S, l, l+m);
        mergesort( S, l+m+1, r);

        #merges two sorted lists
        merge( S, l, l+m ,r);
    else
        # Nothing to do, 1-element list
        end if;
}

function void merge(S array;
                    l,m,r integer) {
    B: array[1..r-l+1];
    i := l;          # Start of 1st list
    j := m+1;        # Start of 2nd list
    k := 1;          # Target list
    while (i<=m) and (j<=r) do
        if S[i]<=S[j] then
            B[k] := S[i]; # From 1st list
            i := i+1;
        else
            B[k] := S[j]; # From 2nd list
            j := j+1;
        end if;
        k := k+1;        # Next target
    end while;
    if i>m then      # What remained?
        copy S[j..r] to B[k..k+r-j];
    else
        copy S[i..m] to B[k..k+m-i];
    end if;
    # Back to original list
    copy B[1..r-l+1] to S[l..r];
}
```



$$l=0$$

$$r=3$$

$$m=1$$



Sortieralgorithmen

A horizontal sequence of 100 circles, with the 50th circle filled black.

MergeSort

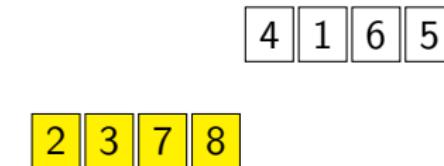
```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-1) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, 1+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then      # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=

r=3

m=3



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooo●oooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$$l=4$$

$$r=7$$

$$m=1$$

Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooo●oooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$l=4$

$r=5$

$m=0$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooo●oooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
    end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := 1;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```

2 3 7 8

6 5

1

4

$l=4$

$r=4$



Sortieralgorithmen

MergeSort

```

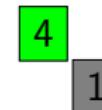
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then                # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=5

r=5



Sortieralgorithmen

A horizontal sequence of 100 circles arranged in a single row. The circles are white with black outlines, except for the 50th circle from the left which is solid black.

MergeSort

```

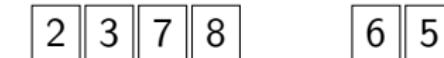
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then                # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=4

r=5

m=0



Sortieralgorithmen



MergeSort

```
function void mergesort(S array;
                        l,r integer) {
    if (l<r) then
        #Sort each ~50% of array
        m := (r-1) div 2;
        mergesort( S, l, l+m);
        mergesort( S, l+m+1, r);

        #merges two sorted lists
        merge( S, l, l+m ,r);
    else
        # Nothing to do, 1-element list
        end if;
}

function void merge(S array;
                     l,m,r integer) {
    B: array[1..r-l+1];
    i := l;          # Start of 1st list
    j := m+1;        # Start of 2nd list
    k := l;          # Target list
    while (i<=m) and (j<=r) do
        if S[i]<=S[j] then
            B[k] := S[i]; # From 1st list
            i := i+1;
        else
            B[k] := S[j]; # From 2nd list
            j := j+1;
        end if;
        k := k+1;        # Next target
    end while;
    if i>m then      # What remained?
        copy S[j..r] to B[k..k+r-j];
    else
        copy S[i..m] to B[k..k+m-i];
    end if;
    # Back to original list
    copy B[1..r-l+1] to S[l..r];
}
```

2 3 7 8

6 5

1
4

$l=4$

$r=5$

$m=0$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```

2	3	7	8
---	---	---	---

6	5
---	---

1	4
---	---

l=4

r=5

m=0



Sortieralgorithmen



MergeSort

```
function void mergesort(S array;
                        l,r integer) {
    if (l<r) then
        #Sort each ~50% of array
        m := (r-1) div 2;
        mergesort( S, l, l+m);
        mergesort( S, l+m+1, r);

        #merges two sorted lists
        merge( S, l, l+m ,r);
    else
        # Nothing to do, 1-element list
        end if;
}

function void merge(S array;
                     l,m,r integer) {
    B: array[1..r-l+1];
    i := l;          # Start of 1st list
    j := m+1;        # Start of 2nd list
    k := 1;          # Target list
    while (i<=m) and (j<=r) do
        if S[i]<=S[j] then
            B[k] := S[i]; # From 1st list
            i := i+1;
        else
            B[k] := S[j]; # From 2nd list
            j := j+1;
        end if;
        k := k+1;        # Next target
    end while;
    if i>m then      # What remained?
        copy S[j..r] to B[k..k+r-j];
    else
        copy S[i..m] to B[k..k+m-i];
    end if;
    # Back to original list
    copy B[1..r-l+1] to S[l..r];
}
```



$$l=6$$

$$r=7$$

$$m=0$$

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left filled black.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then                # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



6

1 =

r=6



Sortieralgorithmen

A horizontal sequence of 100 small circles arranged in a single row. The circles are white with black outlines, except for the 50th circle from the left, which is solid black.

MergeSort

```

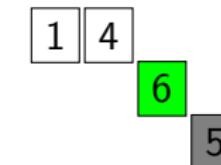
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 =

r=



Sortieralgorithmen

A horizontal sequence of 100 small circles arranged in a single row. The circles are white with black outlines, except for the 50th circle from the left, which is solid black.

MergeSort

```

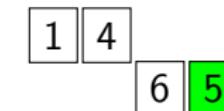
function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]≤S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 =

r=



Sortieralgorithmen

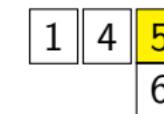


MergeSort

```
function void mergesort(S array;
                        l,r integer) {
    if (l<r) then
        #Sort each ~50% of array
        m := (r-1) div 2;
        mergesort( S, l, l+m);
        mergesort( S, l+m+1, r);

        #merges two sorted lists
        merge( S, l, l+m ,r);
    else
        # Nothing to do, 1-element list
        end if;
}

function void merge(S array;
                     l,m,r integer) {
    B: array[1..r-l+1];
    i := l;          # Start of 1st list
    j := m+1;        # Start of 2nd list
    k := 1;          # Target list
    while (i<=m) and (j<=r) do
        if S[i]<=S[j] then
            B[k] := S[i]; # From 1st list
            i := i+1;
        else
            B[k] := S[j]; # From 2nd list
            j := j+1;
        end if;
        k := k+1;        # Next target
    end while;
    if i>m then      # What remained?
        copy S[j..r] to B[k..k+r-j];
    else
        copy S[i..m] to B[k..k+m-i];
    end if;
    # Back to original list
    copy B[1..r-l+1] to S[l..r];
}
```



$$l=6$$

$$r=7$$

$$m=0$$

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left filled black.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 =

r=

m=0



Sortieralgorithmen

oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$$l=4$$

$$r=7$$

$$m=1$$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooo●oooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$l=4$

$r=7$

$m=1$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooo●oooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```



$$l=4$$

$$r=7$$

$$m=1$$



Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooo●oooooooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```

1

2 3 7 8

4 5 6

$l=0$

$r=7$

$m=3$

Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooo●oooo  
oooooooooooooooooooooooooooooooooooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
        end if;  
    }  
  
function void merge(S array;  
                     l,m,r integer) {  
    B: array[1..r-l+1];  
    i := 1;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```

1 2

3 7 8

4 5 6

$$l=0$$

$$r=7$$

$$m=3$$

Sortieralgorithmen

```
oooooooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooooooooo●oooo
```

MergeSort

```
function void mergesort(S array;  
                        l,r integer) {  
    if (l<r) then  
  
        #Sort each ~50% of array  
        m := (r-1) div 2;  
        mergesort( S, l, l+m);  
        mergesort( S, l+m+1, r);  
  
        #merges two sorted lists  
        merge( S, l, l+m ,r);  
    else  
        # Nothing to do, 1-element list  
    end if;  
}  
  
function void merge(S array;  
                    l,m,r integer) {  
    B: array[1..r-l+1];  
    i := l;          # Start of 1st list  
    j := m+1;        # Start of 2nd list  
    k := 1;          # Target list  
    while (i<=m) and (j<=r) do  
        if S[i]<=S[j] then  
            B[k] := S[i]; # From 1st list  
            i := i+1;  
        else  
            B[k] := S[j]; # From 2nd list  
            j := j+1;  
        end if;  
        k := k+1;        # Next target  
    end while;  
    if i>m then      # What remained?  
        copy S[j..r] to B[k..k+r-j];  
    else  
        copy S[i..m] to B[k..k+m-i];  
    end if;  
    # Back to original list  
    copy B[1..r-l+1] to S[l..r];  
}
```

1 2 3

7 8 4 5 6

$l=0$

$r=7$

$m=3$

Sortieralgorithmen

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1 =

r=

m=3



Sortieralgorithmen

A horizontal sequence of 100 small circles arranged in a single row. The circles are evenly spaced and have a uniform size.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then
    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=

r=

m=3



Sortieralgorithmen

A horizontal row of 100 small circles, with the 99th and 100th circles filled black.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-1];
  i := 1;           # Start of 1st list
  j := m+1;         # Start of 2nd list
  k := 1;           # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;         # Next target
  end while;
  if i>m then       # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-1] to S[l..r];
}

```



1 = 0

r=

m=3



Sortieralgorithmen

A horizontal row of 50 small circles, with the last circle being black.

MergeSort

```

function void mergesort(S array;
                        1,r integer)
if (l < r) then

    #Sort each ~50% of array
    m := (r-1) div 2;
    mergesort( S, 1, 1+m);
    mergesort( S, 1+m+1, r);

    #merges two sorted lists
    merge( S, 1, 1+m ,r);
else
    # Nothing to do, 1-element list
end if;
}

function void merge(S array;
                    1,m,r integer) {
B: array[1..r-1];
i := 1;           # Start of 1st list
j := m+1;         # Start of 2nd list
k := 1;           # Target list
while (i <= m) and (j <= r) do
    if S[i] <= S[j] then
        B[k] := S[i]; # From 1st list
        i := i+1;
    else
        B[k] := S[j]; # From 2nd list
        j := j+1;
    end if;
    k := k+1;         # Next target
end while;
if i > m then      # What remained?
    copy S[j..r] to B[k..k+r-j];
else
    copy S[i..m] to B[k..k+m-i];
end if;
# Back to original list
copy B[1..r-1] to S[1..r];
}

```



1=0

r=

m=3



Sortieralgorithmen

A rectangular grid of 100 small circles, arranged in 10 rows and 10 columns. The circles are evenly spaced and have a thin black outline.

MergeSort

```

function void mergesort(S array;
                        l,r integer)
  if (l<r) then

    #Sort each ~50% of array
    m := (r-l) div 2;
    mergesort( S, l, l+m);
    mergesort( S, l+m+1, r);

    #merges two sorted lists
    merge( S, l, l+m ,r);
  else
    # Nothing to do, 1-element list
  end if;
}

function void merge(S array;
                     l,m,r integer) {
  B: array[1..r-l+1];
  i := l;                      # Start of 1st list
  j := m+1;                     # Start of 2nd list
  k := 1;                       # Target list
  while (i<=m) and (j<=r) do
    if S[i]<=S[j] then
      B[k] := S[i]; # From 1st list
      i := i+1;
    else
      B[k] := S[j]; # From 2nd list
      j := j+1;
    end if;
    k := k+1;                  # Next target
  end while;
  if i>m then      # What remained?
    copy S[j..r] to B[k..k+r-j];
  else
    copy S[i..m] to B[k..k+m-i];
  end if;
  # Back to original list
  copy B[1..r-l+1] to S[l..r];
}

```



1=0

r=

m=3



Sortieralgorithmen

A decorative horizontal border consisting of a repeating pattern of small, light-colored circles arranged in a grid-like fashion.

QuickSort

```

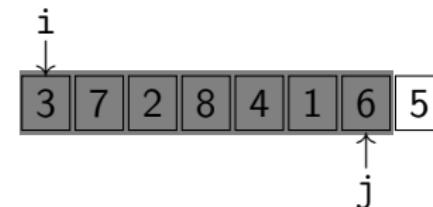
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal sequence of 100 circles, with the first circle being solid black and the rest being hollow white circles with black outlines.

QuickSort

```

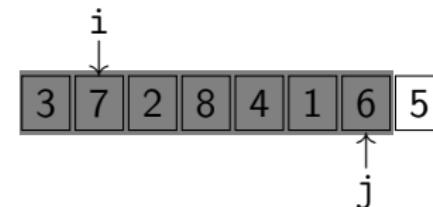
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

QuickSort

```

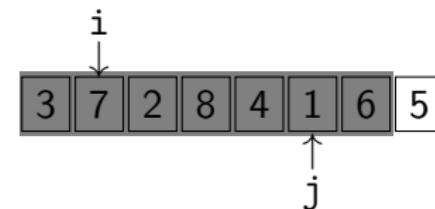
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

QuickSort

```

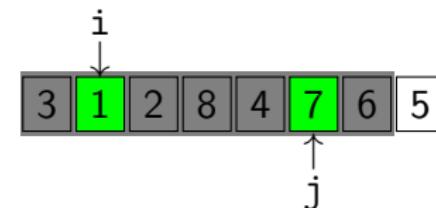
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.  swap( S[i], S[r] );
18.  return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal row of 50 small circles, with the 13th circle from the left filled black.

QuickSort

```

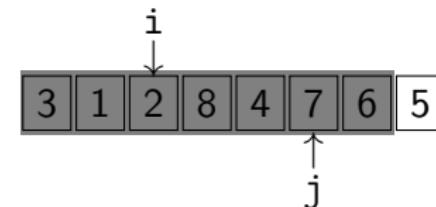
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal row of 50 small circles, with the 50th circle being filled black.

QuickSort

```

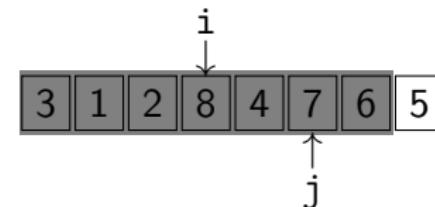
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal sequence of 20 small circles, with the 11th circle from the left filled black.

QuickSort

```

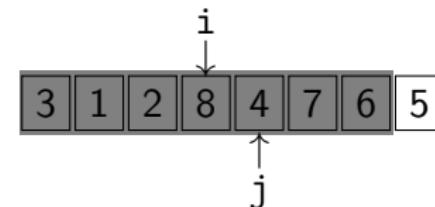
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal sequence of 100 small circles arranged in a single row. The 50th circle from the left is filled black, while all other circles are unfilled.

QuickSort

```

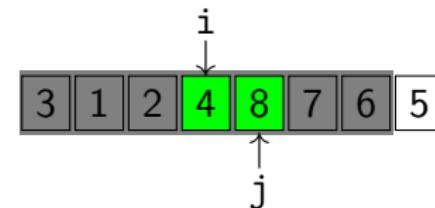
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=

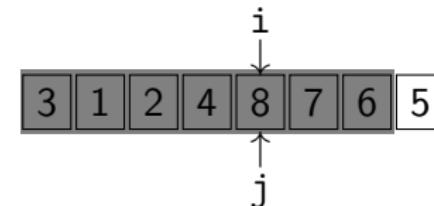
Sortieralgorithmen

oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooo
oooooooooooo●oooooooooooooooooooooooooooo

QuickSort

```
1. func void qsort(S array;  
2.           l,r integer) {  
3.   if r≤l then  
4.       return;  
5.   end if;  
6.   pos := divide( S, l, r);  
7.   qsort( S, l, pos-1);  
8.   qsort( S, pos+1, r);  
9. }
```

```
1. func int divide(S array;  
2.           l,r integer) {  
3.   val := S[r];  
4.   i := l;  
5.   j := r-1;  
6.   repeat  
7.     while (S[i]≤val and i<r)  
8.       i := i+1;  
9.     end while;  
10.    while (S[j]>=val and j>l)  
11.      j := j-1;  
12.    end while;  
13.    if i<j then  
14.      swap( S[i], S[j]);  
15.    end if;  
16.  until i≥j;  
17.  swap( S[i], S[r]);  
18.  return i;  
19. }
```



val=5

pos=



Sortieralgorithmen

QuickSort

```

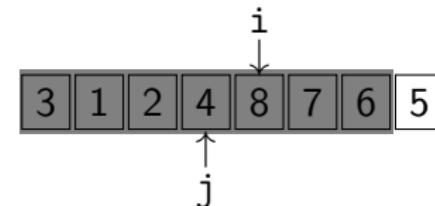
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.  swap( S[i], S[r] );
18.  return i;
19. }

```



val=5

pos=

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left being black.

QuickSort

```

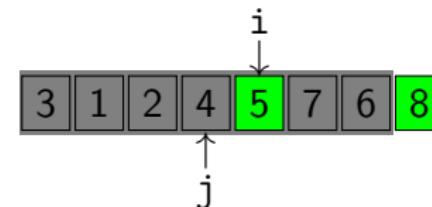
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r <= 1 then
4.     return;
5.   end if;
6.   pos := divide( S, l, r );
7.   qsort( S, l, pos-1 );
8.   qsort( S, pos+1, r );
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=5

pos=5



Sortieralgorithmen

A horizontal row of 50 small circles, with the 25th circle from the left filled black.

QuickSort

```

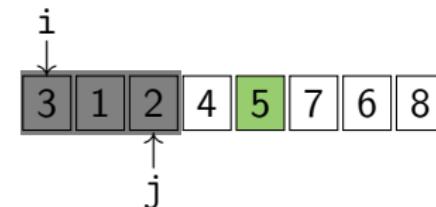
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=4

pos=

Sortieralgorithmen

A horizontal row of 50 small circles, with the 25th circle from the left filled black.

QuickSort

```

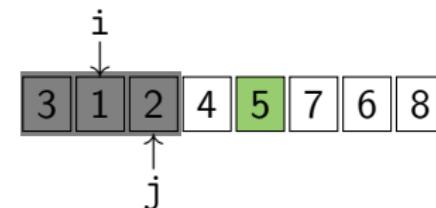
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=4

pos=

Sortieralgorithmen

A horizontal row of 50 small circles, with the 26th circle from the left being filled black.

QuickSort

```

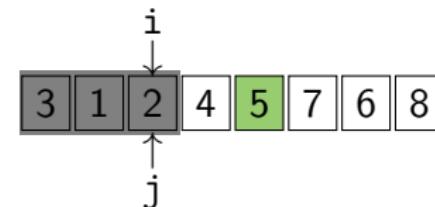
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=4

pos=

Sortieralgorithmen

QuickSort

```

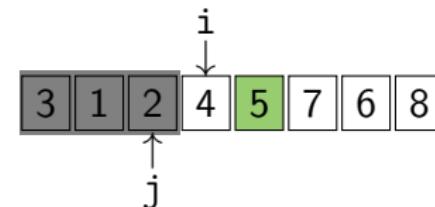
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=4

pos=

Sortieralgorithmen

A horizontal sequence of 20 small circles arranged in a single row. The 10th circle from the left is filled black, while all other circles are hollow.

QuickSort

```

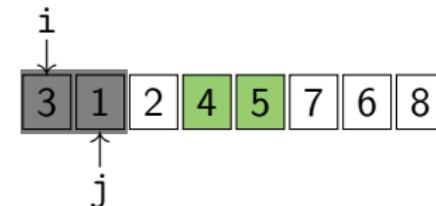
1. func void qsort(S array;
2.                      l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=2

pos=



Sortieralgorithmen

QuickSort

```

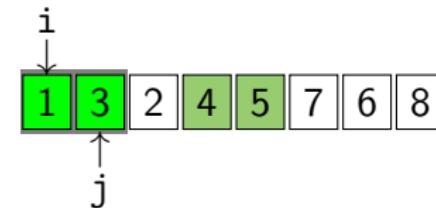
1. func void qsort(S array;
2.                      l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i>j then
14.      swap( S[i], S[j]);
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r]);
18.    return i;
19. }

```



val=2

pos=



Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left highlighted by a black dot.

QuickSort

```

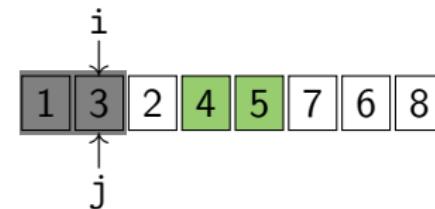
1. func void qsort(S array;
2.                      l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i<j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=2

pos=



Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left highlighted by a black dot.

QuickSort

```

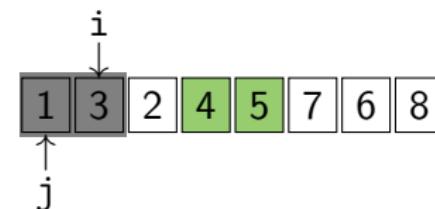
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r );
7.   qsort( S, l, pos-1 );
8.   qsort( S, pos+1, r );
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=2

pos=2



Sortieralgorithmen

QuickSort

```

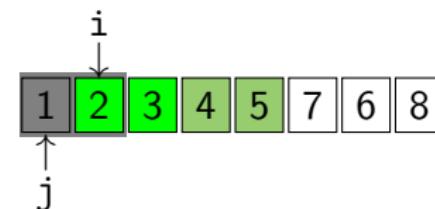
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer)
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=

pos=

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left highlighted by a black dot.

QuickSort

```

1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=

pos=



Sortieralgorithmen



QuickSort

```
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i] <= val and i < r)
8.       i := i+1;
9.     end while;
10.    while (S[j] >= val and j > l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i >= j;
17.  swap( S[i], S[r]);
18.  return i;
19. }
```



val =

pos =



Sortieralgorithmen



QuickSort

```
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }
```

```
1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i] <= val and i < r)
8.       i := i+1;
9.     end while;
10.    while (S[j] >= val and j > l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j]);
15.    end if;
16.  until i >= j;
17.  swap( S[i], S[r]);
18.  return i;
19. }
```



val =

pos =



Sortieralgorithmen

A decorative horizontal border at the bottom of the page, featuring a repeating pattern of small, light blue-outlined circles arranged in a grid-like fashion.

QuickSort

```

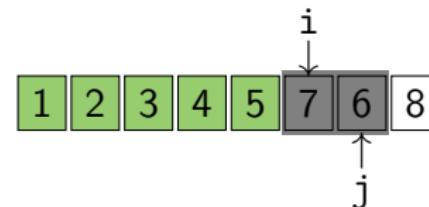
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.  swap( S[i], S[r] );
18.  return i;
19. }

```



val=8

pos=



Sortieralgorithmen

A decorative horizontal border at the bottom of the page, featuring a repeating pattern of small, light blue-outlined circles arranged in a grid-like fashion.

QuickSort

```

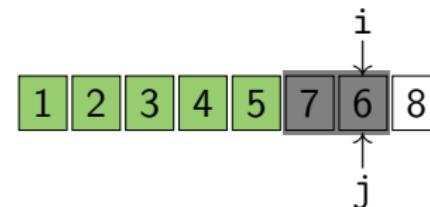
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.  swap( S[i], S[r] );
18.  return i;
19. }

```



val=8

pos=

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left filled black.

QuickSort

```

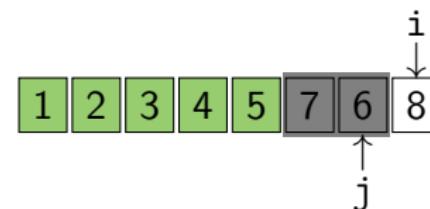
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=8

pos=

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left highlighted by a black dot.

QuickSort

```

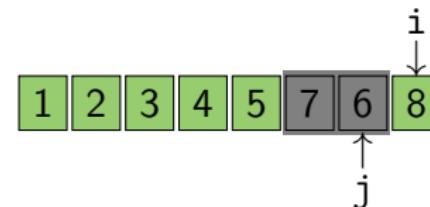
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=8

pos=8

Sortieralgorithmen

A horizontal row of 100 small circles, with the 50th circle from the left highlighted by a black dot.

QuickSort

```

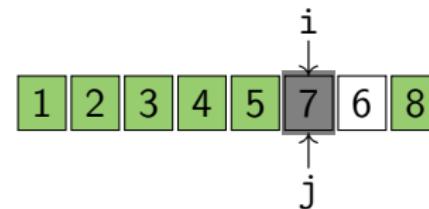
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=6

pos=

Sortieralgorithmen

A decorative horizontal border at the bottom of the page, featuring a repeating pattern of small, light blue-outlined circles arranged in a grid-like fashion.

QuickSort

```

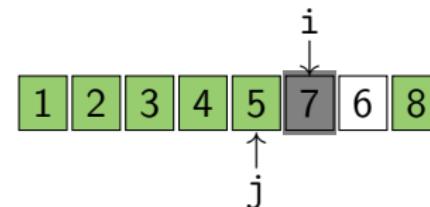
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r );
7.   qsort( S, l, pos-1 );
8.   qsort( S, pos+1, r );
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=6

pos=

Sortieralgorithmen

QuickSort

```

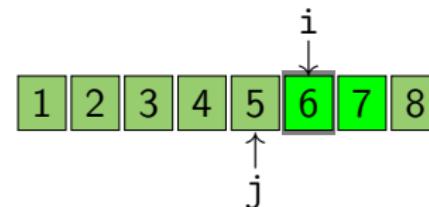
1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r );
7.   qsort( S, l, pos-1 );
8.   qsort( S, pos+1, r );
9. }

```

```

1. func int divide(S array;
2.                   l,r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.  swap( S[i], S[r] );
18.  return i;
19. }

```



val=6

pos=

Sortieralgorithmen

A horizontal row of 100 small circles, with the last circle being black.

QuickSort

```

1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=

pos=



Sortieralgorithmen

QuickSort

```

1. func void qsort(S array;
2.                   l,r integer) {
3.   if r≤l then
4.     return;
5.   end if;
6.   pos := divide( S, l, r);
7.   qsort( S, l, pos-1);
8.   qsort( S, pos+1, r);
9. }

```

```

1. func int divide(S, array;
2.                   l, r integer) {
3.   val := S[r];
4.   i := l;
5.   j := r-1;
6.   repeat
7.     while (S[i]<=val and i<r)
8.       i := i+1;
9.     end while;
10.    while (S[j]>=val and j>l)
11.      j := j-1;
12.    end while;
13.    if i < j then
14.      swap( S[i], S[j] );
15.    end if;
16.    until i=j;
17.    swap( S[i], S[r] );
18.    return i;
19. }

```



val=

pos=

