Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Stefan Kratsch
Dr. K. Ahrens, F. Hegerfeld, F. Nelles, Dr. P. Schäfer

Berlin, den 28.05.2018

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 4

Abgabe: Montag den 11.06.2018 bis 11:10 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten (RUD 25, Raum 3.321). Die Übungsblätter sind in Gruppen von 2 Personen zu bearbeiten. Die Lösungen sind auf nach Aufgaben getrennten Blättern abzugeben. Heften Sie bitte die zu einer Aufgabe gehörenden Blätter vor der Abgabe zusammen. Vermerken Sie auf allen Abgaben Ihre Namen, Ihre **CMS-Benutzernamen**, Ihre Abgabegruppe (z.B. AG123) aus Moodle, und Ihren Übungstermin (z.B. Do 13 Uhr bei Florian Nelles), zu dem Sie Ihre korrigierten Blätter zurückerhalten werden.
Beachten Sie die Informationen auf der Übungswebseite (`https://hu.berlin/algodat18`).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays auf diesem Blatt beginnt bei 1 (und endet also bei $|A|$).
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist gemeint, dass Sie eine möglichst gute obere Schranke der Zeitkomplexität angeben und diese begründen sollen.
- Die durch gaußsche Klammern $\lfloor\ \rfloor$ gekennzeichnete Abrundungsfunktion weist einer reellen Zahl die nächstliegende nicht größere ganze Zahl zu.

Aufgabe 1 (Parkplatzsuche) — 6 + 4 = 10 Punkte

Stellen Sie sich folgendes Szenario vor: Sie wohnen an einer unendlich langen Straße und kommen nach einem anstrengenden Tag mit dem Auto zu Hause an. Leider ist weit und breit kein Parkplatz in Sicht. Aus Erfahrung wissen Sie jedoch, dass es irgendwo auf dieser Straße einen Parkplatz geben muss. Da Sie müde und zugleich umweltbewusst sind, wollen Sie diesen Parkplatz finden und dabei möglichst wenig Fahrtweg zurücklegen.

Wenn Sie wüssten, in welcher Richtung sich dieser Parkplatz befindet, würden Sie direkt in diese Richtung fahren und dabei an L belegten Parkplätzen vorbeifahren (vgl. Abbildung 1).

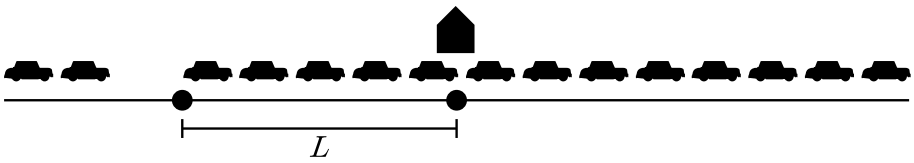

Abbildung 1: Darstellung der Ausgangslage bei der Parkplatzssuche.

1. Entwerfen Sie einen Algorithmus (hier genügt eine informelle Beschreibung), mit dem Sie höchstens einen Weg von $\mathcal{O}(L)$ zurücklegen müssen, um den Parkplatz zu finden.
2. Analysieren Sie die Laufzeit ihres Algorithmus in Abhängigkeit von L.

Hinweis: Sie starten direkt vor Ihrem Haus. Die Straße hat keine Abzweigung und Sie können nur vor- oder zurückfahren. Beachten Sie außerdem, dass Sie L nicht kennen.

Aufgabe 2 (Suchverfahren) **(5 + 3 + 5) + 5 = 18 Punkte**

1. Die Interpolationssuche eignet sich am besten dafür, in einem sortierten Array mit annähernder Gleichverteilung der enthaltenen Werte zu suchen.

   a) Führen Sie einen Schreibtischtest für die Interpolationssuche durch, bei dem das folgende Array A nach dem Wert $c = 34$ durchsucht wird. Geben Sie dazu an, mit welchen Werten die Variablen l, r, diff, range und rank nach jedem Aufruf von Zeile 9 belegt sind.

   $A =$

5	6	9	10	12	13	34	39	43	52	63	76

   b) Geben Sie eine fünfelementige Worst-case-Instanz für die Interpolationssuche an, wobei das Element, nach dem gesucht wird, im Array enthalten sein soll. Geben Sie dazu das zu durchsuchende, sortierte Array mit fünf *verschiedenen* Elementen sowie das Element, nach dem gesucht werden soll, an.

   c) Erläutern Sie, wie sich eine Worst-Case-Instanz für beliebig große Arrays (ohne Duplikate) erstellen lässt, und begründen Sie, warum es sich um Worst-Case-Instanzen handelt. Auch hier soll das gesuchte Element stets im Array enthalten sein.

2. Die Fibonacci-Zahlen sind wie folgt definiert: $F_1 = 1$, $F_2 = 1$ und $F_k = F_{k-2} + F_{k-1}$ für alle $k > 2$.

   Führen Sie einen Schreibtischtest für die Fibonacci-Suche durch, bei dem das folgende Array B nach dem Wert $c = 17$ durchsucht wird. Geben Sie dazu an, mit welchen Werten die Variablen fib, fib1, fib2, offset und m nach jedem Aufruf von Zeile 7 belegt sind.

$B =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Hinweis: Verwenden Sie die Pseudocodes auf der folgenden Seite.

Interpolationssuche

Input: sortiertes Array A der Länge n, zu suchende Zahl c
Output: **true** falls c in A, sonst **false**

```
l := 1; r := n;
while c ≥ A[l] and c ≤ A[r] do
  diff := c − A[l];
  range := A[r] − A[l];
  if range = 0 then
    rank := l;
  else
    rank := l + ⌊(r − l) * diff / range⌋;
  end if
  if c > A[rank] then
    l := rank + 1;
  else
    if c < A[rank] then
      r := rank − 1;
    else
      return true;
    end if
  end if
end while
return false;
```

Fibonacci-Suche

Input: sortiertes Array A der Länge n, zu suchende Zahl c
Output: **true** falls c in A, sonst **false**

```
fib2 := 1; fib1 := 1; fib := 2;
while fib ≤ n do
  fib2 := fib1; fib1 := fib; fib := fib1 + fib2;
end while
offset := 0;
while fib > 1 do
  m := min(offset + fib2, n);
  if c < A[m] then
    fib := fib2; fib1 := fib1 − fib2; fib2:= fib − fib1;
  else
    if c > A[m] then
      fib := fib1; fib1 := fib2; fib2 := fib − fib1; offset := m
    else
      return true;
    end if
  end if
end while
return false;
```

Aufgabe 3 (Suche in sortierten Arrays) **4 + 5 + 7 = 16 Punkte**

Implementieren Sie die nachfolgenden drei Suchverfahren für die Suche eines Werts k in einem aufsteigend sortierten Array A mit Werten vom Typ `Long`:

(a) *binäre Suche*,

(b) *Interpolations-Suche*,

(c) *Fibonacci-Suche*.

Ergänzen Sie den fehlenden Code in der Vorlage `SortedSearch.java`, welche Sie auf der Übungswebseite[1] vorfinden. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen, dürfen jedoch keine außer den von Java bereitgestellten Standard-Bibliotheken verwenden. Verwenden Sie für alle Vergleiche die vorgegebene Klasse `CountingComparator`, welche die Anzahl der Vergleiche zählt. Ein Beispiel für die Implementierung einer Suche in dem vorgegebenen Quellcode-Gerüst finden Sie in der Klasse `SortedSearch.LinearSearch`.
Stellen Sie sicher, dass alle Testfälle in der main-Methode der Datei SortedSearch.java erfolgreich durchlaufen. Achten Sie außerdem auf Randbedingungen und Spezialfälle, die von den Testfällen vielleicht nicht vollständig abgedeckt werden.
Hinweis zur Abgabe: Ihr Java-Programm muss unter *Java 1.8* auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, so dass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Quellcodes `SortedSearch.java` erfolgt über Moodle.

Aufgabe 4 (Untere Schranke für merge) **6 Punkte**

Gegeben seien zwei aufsteigend sortierte Arrays X und Y mit jeweils n in konstanter Zeit vergleichbaren Elementen.
Zeigen Sie: Lässt man als einzige Operation Vergleiche von je zwei Elementen zu, so benötigt jeder deterministische Algorithmus im schlechtesten Fall mindestens $2n - 1$ Vergleiche, um beide Arrays zu einem aufsteigend sortierten Gesamtarray Z der Länge $2n$ zu verschmelzen.

*Hinweis: Überlegen Sie sich zunächst eine Instanz, in der beim Verschmelzen (***merge***) möglichst viele Vergleiche anfallen. Zeigen Sie für diese Instanz dann, dass jeder der Vergleiche nötig ist, da sich der Algorithmus sonst für diese Instanz und eine leicht modifizierte Instanz exakt gleich verhält und der Algorithmus somit nicht korrekt sein kann.*

[1] https://hu.berlin/algodat18