Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2022

H. Pflug, J. Dietel FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Klausurvorbereitung, Blatt 1 20.06.2022

Aufgabe 1

Welche minimale Laufzeitkomplexität haben die folgenden Codefragmente bezüglich der Größe des Werts *n*? Falls sich Worst- und Best-Case unterscheiden, geben Sie beide Werte an.

a)
```
for (int i=1; i<=n; i++) {
  a[i] = 0;
}
```

b)
```
for (int i=1; i<=n; i++) {
 for (int j=1; j<=n; j++) {
   k++;
 }
}
```

c)
```
for (int i=1; i<=n; i++) {
  for (int j=1; j<=i; j++) {
    k++;
  }
}
```

d)
```
for (int i=1; i<=n; i++) {
  a[i]=0;
}
for (int i=1; i<=n; i++) {
  for (int j=1; j<=n; j++) {
    a[i]=a[i]+i+j;
  }
}
```

e)
```
if (x>100) {
  y=x;
} else {
  for (int i=1; i<=n; i++) {
    if (a[i]>y) {
      y=a[i];
    }
  }
}
```

f)
```
for (int i=1; i<=n; i++) {
  if (i>2) {
    return;
  }
}
```

g)
```
int i=n;
while(i>=1) {
  x++;
  i/=2;
}
```

h)
```
int innereSchrittweite=1;
for (int i=0; i<n; i++) {
  for (int k=0; k<n; k+=innereSchrittweite) {
    innereSchrittweite*=2;
  }
}
```

i)
```
public static long fakultaet(int n) {
  if (n>0) {
    return (n*fakultaet(n-1));
  } else {
    return 1;
  }
}
```

j)
```
public static int quersumme(long n) {
  int sum=0;
  while(n>0) {
    sum+=n%10;
    n/=10;
  }
  return sum;
}
```

k)
```
public static boolean hasDoppelte(int[] m) {
   //O in Abhaengigkeit zur Feldlaenge
  Arrays.sort(m);
  for (int i=1; i<m.length; i++) {
    if (m[i-1]==m[i]) {
      return false;
    }
  }
  return true;
}
```

Aufgabe 2

Gegeben seien folgende Klassendefinitionen für eine einfach verkettete Liste:

```
public class Node
{
	public Node next;
	public int value;

	public Node (int wert)
	{
		value = wert;
	}
}

public class List
{
	private Node first;

	public List (Node node)
	{
		first = node;
	}
}
```

Ergänzen Sie die Klasse `List` durch folgende Methoden:

```
public void addFirst(int wert)
```

Fügt `wert` als ersten Knoten in die Liste ein.

```
public int get (int pos)
```

gibt aus der Liste den Wert des Eintrags an der Stelle `pos` zurück.
Falls die Stelle `pos` nicht existiert, wird eine `RuntimeException` geworfen.

Hinweise:

- Das erste Element hat die Position 0.
- Sie dürfen den Klassen weitere Hilfsmethoden oder –attribute hinzufügen. Machen Sie aber deutlich, zu welcher Klasse die Methode bzw. das Attribut gehört.

Aufgabe 3

Implementieren Sie die Funktion

```
public static int[] mische(int[] a, int[] b)
```

die zwei bereits aufsteigend sortierte Integer-Felder übergeben bekommt und diese beiden Felder zu einem einzigen sortierten Feld zusammenfasst und dieses zurückgibt. Gehen Sie dabei wie beim Merge-Sort-Algorithmus vor.

Hinweis: Die beiden übergebenen Felder sind nicht unbedingt gleich groß.

Aufgabe 4

Schreiben Sie eine Methode

```
public double[] sortierteTeilliste(double[] liste, int n).
```

Die Methode sucht im Feld *liste* nach einer aufsteigend sortierten Teilliste von *n* Werten und gibt diese Teilliste als Rückgabewert zurück. Bei mehreren möglichen Teillisten wird die zurückgegeben, die am weitesten vorne im Feld steht. Ist keine Teilliste der erforderlichen Größe im Feld vorhanden, wird `null` zurückgegeben. Beispiel: Das Feld *liste* habe das Aussehen:

```
2, 5, 10, 3, 7, 11, 4, 1, 5, 9, 14, 20, 24, 13, 14
```

Für n=3 ergibt sich als Teilliste (2, 5, 10).

Für n=4 ergibt sich als Teilliste (1, 5, 9, 14). Beachten Sie, dass dies der Anfang der größeren sortierten Teilliste (1, 5, 9, 14, 20, 24) ist. Dies ist erlaubt.

Aufgabe 5

Gegeben sei die folgende Klasse *Node*:

```
public class Node {
   public double data;
   public Node left;
   public Node right
}
```

Schreiben Sie eine Methode

```
public static boolean isHeap(Node root);
```

die *true* zurückgibt, falls der Baum mit der Wurzel *root* ein Heap ist (mit dem größten Element an der Spitze) und *false* sonst.

Aufgabe 6

Zu Beginn des neuen Schuljahrs sollen 50 Schüler möglichst gleichmäßig auf 2 Klassen aufgeteilt werden. Da es in den Klassen wiederholt Ärger gegeben hat, erstellen die Lehrer eine Liste, die Schülerpaare enthält, die auf keinen Fall zusammen in eine Klassen kommen sollen.

a) Wie kann man die Liste von Schülerpaaren in einem Graphen darstellen? Was sind die Kanten, was sind die Knoten?
b) Welche Bedingung muss für diesen Graphen gelten, damit die Schüler in 2 Klassen aufgeteilt werden können?
c) Beschreiben Sie einen Algorithmus, der überprüft, ob die Bedingung aus Teil b) für einen gegebenen Graphen erfüllt ist?

Wie erkennen Sie am Graphen, dass es mehrere Möglichkeiten der Aufteilung gibt?