Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Friedrich Gretz, Benjamin Kaminski, Thomas Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2-3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Mittwoch, den 20.05.2015 um 12:15 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- In Aufgaben, bei denen Sie Algorithmen implementieren sollen, dürfen Sie Ihre Lösung als Pseudo-Code abgeben. Abgaben in verbreiteten imperativen Sprachen wie Java oder C++ sind ebenfalls erlaubt.

Tutoraufgabe 1 (Sortieralgorithmus):

Gegeben sei der folgende Sortieralgorithmus:

```
void sort(int E[]) {
  int i,j,m;
  for (i = 0; i < E.length; i++) {
    m = i;
    for (j = i + 1; j < E.length; j++) {
      if (E[j] <= E[m]) {
          m = j;
      }
    }
    int v = E[i];
    E[i] = E[m];
    E[m] = v;
  }
}
```

a) Nutzen Sie den gegebenen Algorithmus, um das folgende Array zu sortieren. Geben Sie den Zustand des Arrays nach jedem Durchlauf der äußeren Schleife an.

1	3	2	7	0	4	8	5	7	6

b) Geben Sie in wenigen Worten wieder, wie der gegebene Algorithmus funktioniert. In der Vorlesung wurden mehrere Sortierverfahren genannt (siehe Folien). Welcher Name passt zu diesem Algorithmus?

c) Ist der Sortieralgorithmus stabil? Falls dies nicht der Fall ist, geben Sie an, wie er angepasst werden muss, damit er stabil wird!

d) Welche Average-Case Laufzeit besitzt der gegebene Sortieralgorithmus für eine Eingabe der Länge n? Geben Sie die Komplexitätsklasse $\Theta(T_{sort}(n))$ für ein Array E mit Länge `n = E.length` an und begründen Sie Ihre Antwort.

Tutoraufgabe 2 (Mergesort):

Sortieren Sie das folgende Array mithilfe von Mergesort aus der Vorlesung. Geben Sie dazu das Array nach jeder Merge-Operation an.

3	2	8	4	1	5	6	7	4

Tutoraufgabe 3 (Max- und Min-Heaps):

a) Bestimmen Sie, ob folgende Arrays Heaps (Max-Heaps) sind. Falls nicht, geben Sie an wo die Heapeigenschaft verletzt ist.

1.)

56	**47**	**56**	**10**	**20**	**50**	**51**	**1**	**2**	**3**	**18**

2.)

56	**56**	**47**	**10**	**20**	**50**	**51**	**1**	**2**	**3**	**18**

3.)

56	**47**	**56**	**10**	**51**	**20**	**50**	**1**	**2**	**3**	**18**

4.)

56	**47**	**56**	**10**	**5**	**20**	**50**	**1**	**2**	**3**	**18**

b) In der Vorlesung wurden so genannte Max-Heaps vorgestellt. D.h jedes Element ist größer/gleich seiner Kinder. Ein *Min-Heap* ist ein Heap bei dem jedes Element kleiner/gleich seiner Kinderelemente ist.
Bestimmen sie, ob die folgenden Arrays Min-Heaps sind, und falls nicht, geben sie an, wo die Heapeigenschaft verletzt ist

1.)

1	**5**	**1**	**8**	**10**	**4**	**15**	**11**	**12**	**14**	**11**

2.)

0	**1**	**5**	**8**	**10**	**4**	**15**	**11**	**12**	**14**	**11**

3.)

1	**5**	**1**	**11**	**8**	**4**	**15**	**11**	**12**	**14**	**10**

4.)

1	**5**	**1**	**11**	**15**	**4**	**8**	**11**	**12**	**14**	**10**

Tutoraufgabe 4 (Heapsort):

Sortieren Sie das folgende Array mithilfe von Heapsort aus der Vorlesung. Geben Sie dazu das Array nach jeder Swap-Operation an und geben Sie zum jeweils noch unsortierten Arraybereich zusätzlich die grafische Darstellung als Heap an.

5	3	7	0	3	6	1	8

Tutoraufgabe 5 (Quicksort):

Sortieren Sie das folgende Array mithilfe von Quicksort aus der Vorlesung. Geben Sie dazu das Array nach jeder Partition-Operation an.

8	2	4	7	5	6	1	3

Aufgabe 6 (Rekursiver Sortieralgorithmus): (2 + 3 = 5 Punkte)

```
sort(int [ ] A, int l, int r) {
  if (A[l] > A[r]){
    exchange(A[l], A[r]);
  }
  if (l < r-1){
    k:= (r-l+1) div 3;
    sort(A, l, r-k);
    sort(A, l+k, r);
    sort(A, l, r-k);
  }
}
```

a) Bestimmen Sie für den gegebenen Sortieralgorithmus die Komplexitätsklasse Θ im Best-, Worst- und Average-Case für den Aufruf `sort(A,0,n-1)` in Abhängigkeit von `n`, der Anzahl der zu sortierenden Elemente aus dem Array `A`. Dabei verursacht ein Vergleich zwischen Arrayelementen eine Kosteneinheit, alle anderen Operationen sind "kostenlos".

b) Der vorgestellte Algorithmus ist nicht stabil. Ändern Sie den Algorithmus so ab, dass er stabil wird.

Aufgabe 7 (Mergesort): (3 Punkte)

Sortieren Sie das folgende Array mithilfe von Mergesort aus der Vorlesung. Geben Sie dazu das Array nach jeder Merge-Operation an.

4	1	2	6	8	3	7

Aufgabe 8 (Quicksort): (3 Punkte)

Sortieren Sie das folgende Array mithilfe von Quicksort aus der Vorlesung. Geben Sie dazu das Array nach jeder Partition-Operation an.

7	0	4	9	8	6	5	3	1	2

Aufgabe 9 (Heapsort): (5 Punkte)

Sortieren Sie das folgende Array mithilfe von Heapsort aus der Vorlesung. Geben Sie dazu das Array nach jeder Swap-Operation an und geben Sie zum jeweils noch unsortierten Arraybereich zusätzlich die grafische Darstellung als Heap an.

4	1	2	5	7	3	1	6

Aufgabe 10 (Beweis der Lemmata aus der Vorlesung): (2 + 4 + 3 = 9 Punkte)

a) Beweisen Sie, dass ein Heap mit n Elementen die Höhe $\lfloor \log_2 n \rfloor$ hat.
Hinweis: Sie dürfen als bekannt voraussetzen, dass die minimale Höhe eines **Binärbaumes** *mit n Elementen* $h = \lceil \log_2(n+1) \rceil - 1$ *ist.*

b) Beweisen Sie, dass ein Heap mit n Elementen $\lceil \frac{n}{2} \rceil$ Blätter besitzt.
Was sagt das über die Anzahl der inneren Knoten des Heaps?

c) Beweisen Sie, dass ein Heap maximal $\lceil n/2^{h+1} \rceil$ Knoten mit der Höhe h hat.
Hinweis: Die Höhe h des Knotens entspricht der Länge des längsten Pfades zu einem Blatt des Heaps.