Algorithmische Mathematik

Wintersemester 2013
Prof. Dr. Marc Alexander Schweitzer und
Dr. Einar Smith
Patrick Diehl und Daniel Wissel

universitätbonn

Übungsblatt 2.

Abgabe am **04.11.2013**.

Aufgabe 1. (Datenstruktur: Binärer Heap)

Eine wichtige Datenstruktur für Algorithmen ist der Heap (Halde). In dieser ist die Bestimmung des Minimums oder Maximums von n Schlüsseln in $\mathcal{O}(1)$ möglich. In unserem Fall verwenden wir als Datenstruktur für den binären Heap ein Array $A = \{a_0, a_1, \dots a_{n-1}\}$ der Länge n. Für ein Array der Länge $n = 7$ sieht die graphische Repräsentation des Heaps wie folgt aus:

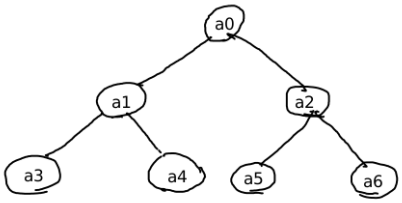


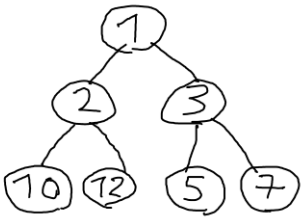
Abbildung 1: Graphische Darstellung eines Heaps

Wir betrachten folgende Heapvarianten:

- Maxheap:
  Ein Heap heißt MaxHeap, wenn folgende Ordnung $a_i \geq a_{2i+1}$ und $a_i \geq a_{2i+2}$ im Array A gilt.
- MinHeap:
  Ein Heap heißt MinHeap, wenn folgende Ordnung $a_i \leq a_{2i+1}$ und $a_i \leq a_{2i+2}$ im Array A gilt.

a. Geben Sie die graphische Darstellung des Heaps zum Array $A = \{7, 5, 6, 3, 1, 2, 4\}$ an und entscheiden Sie, ob es sich um einen Max- oder MinHeap handelt.

b. Transferieren Sie folgenden Heap in die Arraydarstellung und entscheiden Sie, ob es sich um einen Max- oder MinHeap handelt.



(2 + 2 = 4 Punkte)

Aufgabe 2. (Heapgenerieung)

Für einen effizienten Heapaufbau wird die Funktion *void heapify(int a[], int n, int i)* benötigt. Ein mögliches Szenario für diese Methode wäre der Fall, dass im Array das erste Element ausgetauscht wird. In diesem Fall kann es sein, dass im Array die MinHeap- oder MaxHeap-Ordnung nicht mehr gilt. Dann muss das ersetzte Element in den linken oder rechten Teilheap eingefügt werden. Das Element muss so weit in den Teilheap versickern, bis die Heapordnung wieder erfüllt ist.

```
1  void heapify(int a[], int n, int i){
     int j, h=a[i];
3    if(i<0) return;
     while(i<n/2){
5      j=i+i+1;
       if(j+1<n && a[j]<a[j+1]) j++;
7      if(h>=a[j]) break;
       a[i]=a[j]; i=j;
9    }
     a[i]=h;
11 }
```

Ein kompletter Heap kann nun wie folgt generiert werden

```
1  void construct(int a[], int n){
     int i;
3    for(i=n/2-1; i>=0; i--)
       heapify(a,n,i);
5  }
```

a. Zeigen Sie, dass die Methode *heapify* in der Komplexitätsklasse $\mathcal{O}(\log(n))$ liegt.

b. Zeigen Sie, dass die Methode *construct* in der Komplexitätsklasse $\mathcal{O}(n)$ liegt.

c. Führen Sie die Generierung eines MaxHeaps auf dem Array $A = \{3, 5, 1, 8, 2, 4, 7, 6\}$ durch. Verwenden Sie die graphische Darstellung des Heaps.

$(2 + 4 + 4 = 10$ Punkte)

Aufgabe 3. (Heapsort)

Beim Heapsort-Algorithmus wird aus dem Eingabearray ein MaxHeap generiert. Auf dem generiertem MaxHeap wird dann wieder jeweils die Funktion *heapify* aufgerufen.

```
1  void heapsort(int a[], int n){
     int i, tmp;
3    construct(a,n);
     for(i=n-1; i>=0; i--){
5      tmp=a[0]; a[0]=a[i]; a[i]=tmp;
       heapify(a,i,0);
7    }
   }
```

a. Zeigen Sie, dass Heapsort in der Komplexitätsklasse $\mathcal{O}(n \log(n))$ liegt.

b. Führen Sie Heapsort auf dem in Aufgabe 2 generierten MaxHeap durch.

$(2 + 4 = 6$ Punkte)

Programmieraufgabe 1. (Bubblesort, Quicksort und Heapsort)

a. Implementieren Sie Bubblesort (1.4) als Funktion *bubblesort(double* array, int length)*

b. Implementieren Sie Quicksort (1.20) als Funktion *quicksortOwn(double* array, int length)*

c. Implementieren Sie Heapsort als Funktion *heapsort(double* array, int length)*

d. Messen Sie die Laufzeit für die drei Sortieralgorithmen für gleichverteilte Zufallszahlen und erstellen Sie ein Diagramm. Drucken Sie das Diagramm aus und geben Sie es mit den Theorieaufgaben zusammen ab.

Hinweise:

- Die Zahlen in Klammern beziehen sich auf die Kapitel des Tafelanschriebs.
- Auf der Homepage[1] finden Sie die Datei *util.h*. Diese können Sie in ihr Programm mittels *#include "util.h"* in ihr Programm einbinden. Dann können Sie folgende Funktionen nutzen:
  - *void randomArray(double* array, int length)* Befüllt ein Array mit gleichverteiten Zufallszahlen.
  - *void beginTime()* Startet die Zeitmessung.
  - *float endTime()* Liefert die Zeit zwischen *beginTime()* und *endTime()*

(20 Punkte)

Abgabe am 11.11.2013 nach der Vorlesung A oder Vorlesung B

[1] `http://www.ins.uni-bonn.de/teaching/vorlesungen/AlMaWS13/`