Übungsblatt 6

Abgabe via Moodle.
Deadline Fr. 16ter Juni

Aufgabe 1 (*Heaps*, $1+1$ *Punkte*)

1. Erfüllt ein aufsteigend sortierter Array als Heap die Heapeigenschaft? (Begründen Sie!)
2. Begründen Sie warum das Parent-Element des Elements mit Index j in einem als Array gespeicherten Heap den Index $\lfloor \frac{j}{2} \rfloor$ hat.

Lösung:

1. In einem als Array gespeicherten Heap werden die Schichten nacheinander gespeichert, da der Array sortiert ist, ist jedes Element einer Schicht kleiner als alle Elemente der darunterliegenden Schicht. Damit ist die Heapeigenschaft überall erfüllt.
2. Die Kindelemente des Elements mit Index i haben die Indizes $2i$ und $2i+1$ im Array, und es gilt $\lfloor \frac{2i}{2} \rfloor = \lfloor \frac{2i+1}{2} \rfloor = i$.

Aufgabe 2 (*Binäre Heaps*, 6 *Punkte*)

Nehmen Sie die letzten 4 Ziffern Ihrer Matrikelnummer und bilden Sie daraus alle möglichen binären Heaps. Stellen Sie dabei die Heaps als implizites Feld dar.

Hinweis: Die Wurzel des Heaps ist minimal.

Lösung:
Beispiel: Matrikelnummer 1579534

Alle möglichen impliziten Heaps sind $\langle 3,4,5,9 \rangle$, $\langle 3,5,4,9 \rangle$ und $\langle 3,4,9,5 \rangle$.

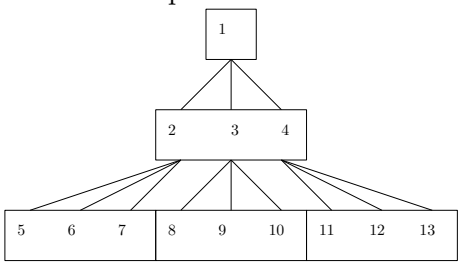
Aufgabe 3 (*d-näre Heaps*, $5+3$ *Punkte*)

Bis jetzt kennen Sie binäre Heaps und deren implizite Repräsentation in einem Array. Entwickeln Sie nun einen d-nären Heap:

- Statt maximal 2 Kindern hat jedes Element nun maximal d Kinder.
- Der Heap sei in einem Array gespeichert (ohne Lücken).

1. Geben Sie für das Element mit Arrayindex j die Formeln zum Berechnen des Elterelements und der d Kindelemente an (vom 1. Kind bis zum d. Kind). Begründen Sie ihre Formeln. Diese Formeln sind nicht so einfach wie die für einen binären Heap.
2. Geben Sie zudem eine Pseudocode-Implementierung von *deleteMin* und *siftDown* an.

Lösung:
Hier ein Beispiel für $d = 3$ und 3 Ebenen. (die Nummern sind die Indizes im Array)



1. Ein d-närer Heap enthält 1 Element in der ersten Ebene, d in der zweiten und allgemein d^{k-1} Elemente in der k-ten Ebene. Die ersten k Ebenen enthalten damit (falls alle Ebenen voll sind) $\sum_{i=0}^{k-1} d^i = \frac{d^k-1}{d-1}$ Elemente. Damit das Element mit Arrayindex j in Ebene $k+1$ liegt ($k \geq 1$) muss daher gelten: $\frac{d^k-1}{d-1} < j \leq \frac{d^{k+1}-1}{d-1}$ oder $\lceil \log_d(j(d-1)+1) \rceil = k+1$. Das Element mit dem kleinsten Index in Ebene $k+1$ ist $\frac{d^k-1}{d-1} + 1$. Das i-te Element in Ebene k hat die Elemente $\frac{d^k-1}{d-1} + d(i-1) + 1, \frac{d^k-1}{d-1} + d(i-1) + 2, \ldots, \frac{d^k-1}{d-1} + d(i-1) + d$ als Kinder. Definiere nun $e(j) := \lceil \log_d(j(d-1)+1) \rceil$ als Ebenenfunktion für den Index j. Dann gilt für die Indizies der Kinder:

$$\begin{aligned}
kind_1(j) &= d(j - (\tfrac{d^{e(j)-1}-1}{d-1} + 1)) + \tfrac{d^{e(j)}-1}{d-1} + 1 = & dj - d + 1 + 1 \\
kind_2(j) &= d(j - (\tfrac{d^{e(j)-1}-1}{d-1} + 1)) + \tfrac{d^{e(j)}-1}{d-1} + 2 = & dj - d + 1 + 2 \\
&\vdots & \vdots & \vdots \\
kind_d(j) &= d(j - (\tfrac{d^{e(j)-1}-1}{d-1} + 1)) + \tfrac{d^{e(j)}-1}{d-1} + d = & dj - d + 1 + d
\end{aligned}$$

   Für das Elterelement gilt ($j > 1 + d$):

$$elter(j) = \lfloor \frac{j - (\frac{d^{e(j)-1}-1}{d-1} + 1)}{d} \rfloor + \frac{d^{e(j)-2} - 1}{d-1} + 1 = \lfloor \frac{j + d - 2}{d} \rfloor$$

2. Implementierung von *deleteMin* wie beim binären Heap (siehe Vorlesungsfolien), nur *siftDown* muss geändert werden.

   Implementierung von *siftDown* wobei $h[1..n]$ der Array ist, der den Heap enthält:

```
1: procedure siftDown(i ∈ ℕ)
2:     If kind_1(i) ≤ n Then        //i is not a leaf
3:         m := kind_1(i)
4:         For j := 2 upto d Do
5:             If kind_j(i) ≤ n ∧ h[kind_j(i)] ≤ h[m] Then m:= kind_j(i)
6:         If h[i] > h[m] Then       //heap property violated
7:             swap(h[i], h[m])
8:             siftDown(m)
```

Aufgabe 4 (*Heapsort ausführen,* 8 *Punkte*)

Führen Sie *heapSortDecreasing* auf dem Array | 15 | 7 | 9 | 2 | 12 | 8 | 3 | 10 | 5 | aus. Schreiben Sie das Array nach jeder Vertauschung oder anderem wichtigen Schritt auf. Das Array soll zum Schluss **absteigend** sortiert sein.

Lösung:
Zunächst wird aus dem völlig unsortierten Array ein Heap konstruiert:

Benutze die Funktion *buildHeapBackwards* aus der Vorlesung:

siftDown(4):

15	7	9	2	12	8	3	10	5

siftDown(3):

15	7	9	2	12	8	3	10	5
15	7	3	2	12	8	9	10	5

siftDown(2):

15	7	3	2	12	8	9	10	5
15	2	3	7	12	8	9	10	5
15	2	3	5	12	8	9	10	7

siftDown(1):

15	2	3	5	12	8	9	10	7
2	15	3	5	12	8	9	10	7
2	5	3	15	12	8	9	10	7
2	5	3	7	12	8	9	10	15

Und nun der eigentliche Heap-Sort: Sortierte Elemente werden *kursiv* dargestellt.

2	5	3	7	12	8	9	10	15
15	5	3	7	12	8	9	10	*2*
3	5	15	7	12	8	9	10	*2*
3	5	8	7	12	15	9	10	*2*
10	5	8	7	12	15	9	*3*	*2*
5	10	8	7	12	15	9	*3*	*2*
5	7	8	10	12	15	9	*3*	*2*
9	7	8	10	12	15	*5*	*3*	*2*
7	9	8	10	12	15	*5*	*3*	*2*
15	9	8	10	12	*7*	*5*	*3*	*2*
8	9	15	10	12	*7*	*5*	*3*	*2*
12	9	15	10	*8*	*7*	*5*	*3*	*2*
9	12	15	10	*8*	*7*	*5*	*3*	*2*
9	10	15	12	*8*	*7*	*5*	*3*	*2*
12	10	15	*9*	*8*	*7*	*5*	*3*	*2*
10	12	15	*9*	*8*	*7*	*5*	*3*	*2*
15	12	*10*	*9*	*8*	*7*	*5*	*3*	*2*
12	15	*10*	*9*	*8*	*7*	*5*	*3*	*2*
15	*12*	*10*	*9*	*8*	*7*	*5*	*3*	*2*
15	*12*	*10*	*9*	*8*	*7*	*5*	*3*	*2*

Aufgabe P6 (*Minimum Absolute Difference in an Array, optional*)

Für die praktischen Übungen verwenden wir die Plattform `www.hackerrank.com`. Hier müssen Sie sich registrieren um an den Übungen teilzunehmen. Unter dem Link

`https://www.hackerrank.com/adsi-2023`

finden die praktischen Übungen in der Form eines Programmierwettbewerbs statt.

Die sechste Challenge heißt "Minimum Absolute Difference in an Array". Ihre Aufgabe ist es aus einem gegebenen Array an Integer Werten die beiden Zahlen herauszusuchen, welche am nächsten beieinander liegen. Das heißt bei denen der Betrag der Differenz am kleinsten ist. Zurückzugeben ist die kleinste gefundene Differenz.

Eine genauere Beschreibung, sowie ein Beispiel finden Sie auf HackerRank.