

Algorithmen & Datenstrukturen

Übungsstunde 5

Miniquiz 5

- Passwort: **sorting4**
- 13 Fragen
- 15 Minuten
- 4 Punkte insgesamt

Programm

- Miniquiz
- Rückmeldung Serie 3, PG2, PG3
- Besprechung Aufgaben
- Theory Recap
- Prüfungsaufgabe

Besprechung Miniquiz

Rückmeldung Serie 3

- Verwendung von Hint/Theorem welches gegeben wird soll gekennzeichnet werden
- Irgendwo muss stehen was wir beweisen
- Pseudocode Fehler: Zuweisung vs. Gleichheitstest, Wiederholende Statements, Variablen vertauschen, Variabel Initialisierung

Theory Recap

Inhalt

- Sortieralgorithmen Teil 2
- Abstrakte Datentypen
- Datenstrukturen

Quicksort

- Pivotelement p wählen, korrekte Position von p finden, durch einordnen (größer?, kleiner?) von restlichen Elementen
- in-place
- $O(n \cdot \log(n))$
- rekursiv implementiert

```
int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1);
    for (int j=low; j<high; j++){
        if (arr[j] < pivot){
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;

    return i+1;
}

void sort(int arr[], int low, int high){
    if (low < high){
        int pi = partition(arr, low, high);
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```

Heap-Sort

- Siehe Video: <https://youtu.be/Q1yi1eaqN7I?si=390cnFE6ILfS8Qc6>
- in-place
- $O(n \cdot \log(n))$
- rekursiv implementiert

```
public static int[] heapSort(int[] arr) {
    for(int i = arr.length/2; i >= 0; i--) {
        restoreHeapCondition(arr, i, arr.length);
    }
    for(int i = arr.length-1; i >= 1; i--) {
        int tmp = arr[0];
        arr[0] = arr[i];
        arr[i] = tmp;
        restoreHeapCondition(arr, 0, i);
    }

    return arr;
}

public static int[] restoreHeapCondition(int[] arr, int i, int n) {
    int j;
    while(2*i < n) {
        j = 2*i;
        if(j+1 < n) {
            if(arr[j] < arr[j+1]) {
                j++;
            }
        }
        if(arr[i] >= arr[j]) {
            break;
        }
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
        i = j;
    }
    return arr;
}
```

Abstrakte Datentypen

- **Heap:** Schneller Zugriff auf Maximum/Minimum
- **Liste:** feste Reihenfolge der Objekte
 - **Array:** wird benutzt, wenn Anzahl Elemente bekannt, viele nützliche Funktionen in Java
 - **(doubly) - Linked List:** Elemente bestehen aus Schlüssel und Pointer, der auf das nächste bzw. nächste und vorherige Element zeigt -> Spezielle Art von Liste
 - `.insert(x)`, `.poll()`, `.get(i)`

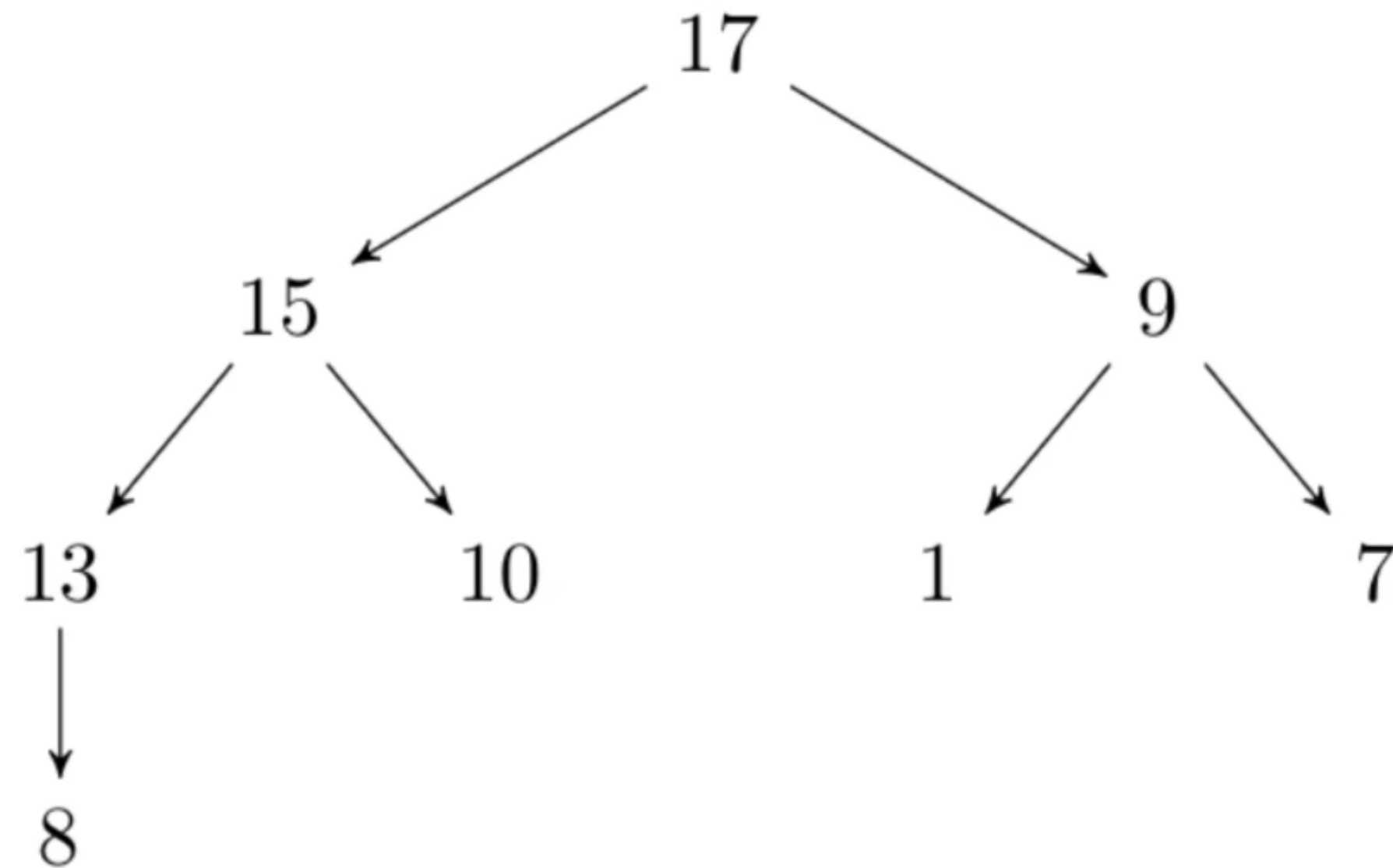
Abstrakte Datentypen

- **Stack:** LIFO mit Operationen push(x), pop(), top() - mit linked-list implementiert
- **Queue:** FIFO mit enqueue(x), dequeue() - mit doubly-linked-list implementiert
- **Priority Queue:** wird mit Heap implementiert

Prüfungsaufgaben

i) Draw a Max-Heap that contains the keys 3, 6, 1, 5, 7, 2.

ii) Draw the Max-Heap obtained from the following Max-Heap by performing the operation DELETE-MAX once.



Consider the following recursive function that takes as an input a natural number m that is a power of two (that is, $m = 2^k$ for some nonnegative integer k).

Algorithm 3 $g(m)$

```
if  $m > 1$  then
     $g(m/2)$ 
     $g(m/2)$ 
     $g(m/2)$ 
     $g(m/2)$ 
    for  $i = 1, \dots, m^2$  do
         $f()$ 
else
     $f()$ 
```

Let $T(m)$ be the number of calls of the function f in $g(m)$.

i) Give a recursive formula for $T(m)$.

ii) Write $T(m)$ in $\mathcal{O}$ -notation in terms of m (as tight and simplified as possible).

Exercise Recommendations

Aufgabenblatt 5

- 5.1 Dauert nicht lang und ist sehr nützlich
- 5.2 skippen wenn sonst schon viel zu tun
- 5.3 Bonus
- 5.4 Bonus
- 5.5 Kann man sich mal durchdenken und allenfalls implementieren

Leetcode of the Week

- 143. Reorder List
- <https://leetcode.com/problems/reorder-list/>

Peergrading

- Aufgabe 4.4
- Grading Scheme in Lösungen beachten