Algorithmen und Datenstrukturen

Übungsblatt 2

Prof. Dr. U. Schöning
T. Büchler
Institut für Theoretische Informatik

Wintersemester 2020/21
Abgabe bis 20.11. **13:00 Uhr**, Besprechung: 23.-27.11.

Aufgabe 2.1 *4 Punkte.*

a) Wieviele (paarweise Element-) Vergleiche muss ein Sortierverfahren nach der in der Vorlesung bewiesenen Formel mindestens machen, um 64 Elemente zu sortieren?

b) Wieviele Vergleiche benötigt QuickSort im average-case höchstens bei 64 Elementen, nach der in der Vorlesung bewiesenen oberen Abschätzung?

c) Wieviele Vergleiche benötigt MergeSort, gemäß der bewiesenen Formel?

d) Gib einen Vorteil von Mergesort gegenüber Heapsort an.

Lösung 2.1

a) $\log_2(64!) = 296$

b) $\lceil 2 \cdot 64 \ln(64) \rceil = 533$ oder $\lceil 1.39 \cdot 64 \log_2(64) \rceil = 533$

c) $1 - 64 + 64 \cdot \log_2(64) = 321$

d) Mergesort ist stabil oder als externes Verfahren für sehr große Daten nutzbar...

Aufgabe 2.2 *2 Punkte.*

Bei unserer Quicksort-Implementation wird immer das erste Element im Array als Pivotelement gewählt. Wie sieht eine Worst-Case-Eingabe der Länge 10 aus? Bestimmen Sie die Anzahl der Vegleiche, die der Algorithmus bei dieser Eingabe benötigt.

Lösung 2.2

Worst-Case ist ein bereits (rückwärts) sortiertes Array. Also zB $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$ Anzahl der Vergleiche: $9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45$.

Aufgabe 2.3 *1 Punkte.*

Gegeben sei folgendes Problem: Eingabe ist eine beliebige Permutation der Zahlen von 1 bis n. Ausgegeben werden sollen alle Zahlen der Eingabe in sortierter Reihenfolge. Welche Laufzeit (in $\mathcal{O}$-Notation) benötigt ein Programm (bestmöglich programmiert) für die Lösung dieses Problems.

Lösung 2.3

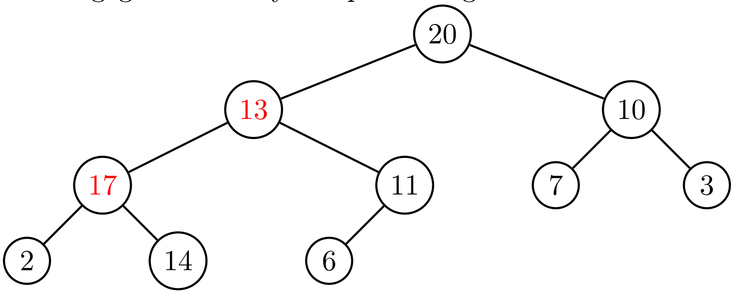
Die Lösing ist immer $\langle 1, 2, 3, \ldots, n \rangle$. Das kann in $\mathcal{O}(\,n\,)$ Zeit ausgegeben werden.

Aufgabe 2.4 *1+3+1+1+1 Punkte.*

(a) Folgendes Array $\langle 20, 13, 10, 17, 11, 7, 3, 2, 14, 6 \rangle$ repräsentiert einen Heap. Gibt es Elemente, die die Heap-Eigenschaft verletzen? Falls ja, welche?

Lösung

Das angegebene Array entspricht folgendem Baum:

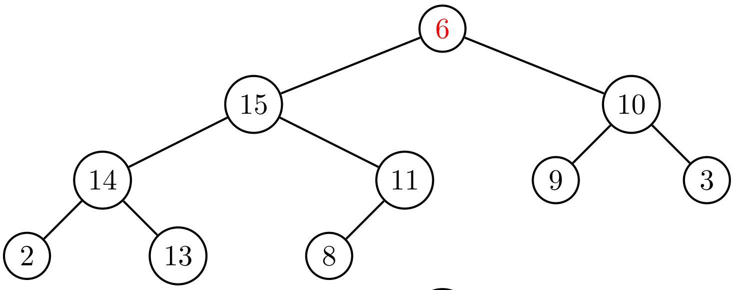


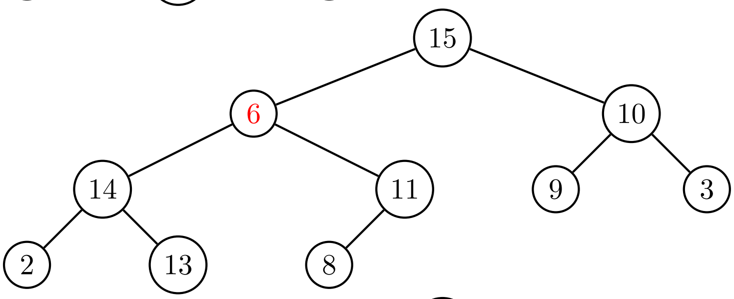
Damit verletzen die Knoten 13 und 17 die Heapeigenschaft.

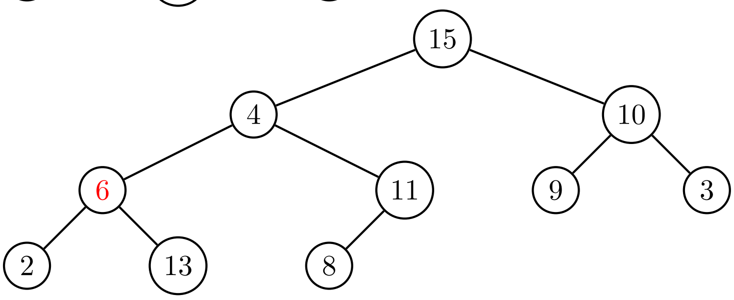
(b) Der Wert an der Wurzel eines Heaps wurde soeben auf 6 geupdated. Das Array sieht nun wie folgt aus: $\langle 6, 15, 10, 14, 11, 9, 3, 2, 13, 8 \rangle$. Stellen Sie mit der Prozedur Heapify die Heap-Eigenschaft wieder her. Geben Sie alle Zwischenschritte an.

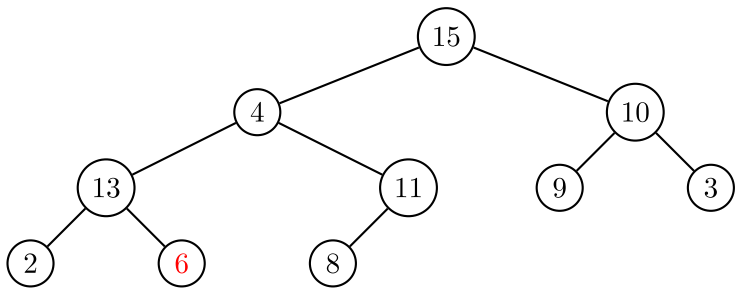
Lösung

$\langle 6, 15, 10, 14, 11, 9, 3, 2, 13, 8 \rangle$ - $\langle 15, 6, 10, 14, 11, 9, 3, 2, 13, 8 \rangle$ - $\langle 15, 14, 10, 6, 11, 9, 3, 2, 13, 8 \rangle$ - $\langle 15, 14, 10, 16, 11, 9, 3, 2, 6, 8 \rangle$ -









(c) In einem Heap ist das größte Element der dargestellten Zahlenmenge an der Wurzel. Zeigen Sie, dass eines der Kinder der Wurzel das zweitgrößte Element sein muss.

Lösung

Angenommen das zweitgrößte Element x_2 befindet sich im linken Teilbaum, dann ist x_2 das größte Element des linken Teilbaums. Wenn x_2 nicht die Wurzel des linken Teilbaums bildet, dann ist die Heap-Eigenschaft verletzt. Wenn x_2 im rechten Teilbaum ist, dann gilt eine analoge Eigenschaft.

(d) In welchen Ebenen kann sich das drittgrößte Element befinden?

Lösung

Entweder das andere Kind der Wurzel oder die Kinder des zweitgrößten Elements. Daher Ebenen 2 und 3.

(e) In welchen Ebenen kann sich das kleinste Element befinden?

Lösung

Es darf keine Kinder haben. Daher in der letzten oder vorletzen Ebene.

Aufgabe 2.5 *6 Punkte.*

Lösen Sie die Programmier-Aufgabe "Tickets".
Reichen Sie Ihren Code für jeden der 3 Test-Cases ein. Pro bestandenen Testcase gibt es 2 Punkte. **Geben Sie Ihren Domjudge-Teamnamen bei Ihrer Abgabe an, damit Ihnen Ihre Lösung zugeordnet werden kann.**

Lösung 2.5

In einem Maxheap der Größe k werden die k kleinsten Werte gespeichert. Das kt kleinste ist zu jedem Zeitpunkt an der Wurzel. Ist eine neue Zahl kleiner als die Wurzel wird sie in den heap aufgenommen. Der Heap wird mit unendlich Werten initialisiert.

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<unsigned int> heap;
unsigned int DUMMY_VAL = -1;

void heapify(unsigned int parent){

   unsigned int left_child  = (parent*2)+1;
   unsigned int right_child = (parent+1)*2;
   unsigned int greater_child;

   if(left_child >= heap.size()) //no childs
      return;

   if(right_child >= heap.size()) //only one child
      greater_child = left_child;
   else
      if(heap[left_child] > heap[right_child])
         greater_child = left_child;
      else
         greater_child = right_child;

   if(heap[parent] < heap[greater_child]){ //heap property broken
      //restore heap property
      unsigned int tmp = heap[parent];
      heap[parent] = heap[greater_child];
      heap[greater_child] = tmp;
      heapify(greater_child);
   }
}

int main() {
   unsigned int c,k,x;
   cin >> c;
   cin >> k;

   //initialise heap with big dummy values
   heap = vector<unsigned int>(k);
   for(unsigned int i = 0; i < k; i++)
      heap[i] = DUMMY_VAL;

   for(int i = 0; i < c; i++){
      unsigned int head = heap[0];
      cin >> x;
      if(x == 0)
         //Martin hat angerufen
         if(head == DUMMY_VAL)
            cout << -1 << endl;
         else
            cout << head << endl;

         else if(head == DUMMY_VAL || x < head){
            heap[0] = x;
            heapify(0);
         }
   }

   return 0;
}
```