

Tentamen Data Structures and Algorithms

19 Oktober 2009

This exam consists of 7 tasks and one optional task.

Task 1.

(a) Given is the following algorithm:

```
Loop(n):  
  s = 0  
  for i = 1 to n do  
    for j = 1 to 3n do  
      s = s + 1  
    done  
  done
```

What is the time complexity of this algorithm in O-notation?

(5 points)

$O(n^2)$

(b) Give a pseudo-code for implementing the operations **enqueue** and **dequeue** of the queue ADT using two stacks S_1 and S_2 for storing the elements.

(5 points)

```
enqueue(o):  
  S1.push(o)  
  
dequeue():  
  if ¬S2.isEmpty() then return S2.pop()  
  if S1.isEmpty() then throw EmptyQueueException  
  while ¬S1.isEmpty() do S2.push(S1.pop())  
  return S2.pop()
```

(c) What is the worst-case time complexity of your **enqueue** and **dequeue** from (a) in big O-notation? (assuming that the stack operations are $O(1)$) Answer suffices.

(4 points)

enqueue is $O(1)$
dequeue is $O(n)$ (not required info: the amortized costs are $O(1)$)

(d) Fill in the question mark with a possible simple, but not overestimating expression:

- (i) $n^2 + n^3 + n \in O(?)$
- (ii) $2 \cdot n + \log_2 n \in O(?)$
- (iii) $4^n + 7 \cdot n \cdot \log_2 n \in O(?)$

(6 points)

(i) $n^2 + n^3 + n \in O(n^3)$
(ii) $2 \cdot n + \log_2 n \in O(n)$
(iii) $4^n + 7 \cdot n \cdot \log_2 n \in O(4^n)$

Task 2.

(a) We implement a priority queue using an **unsorted** array. What is the time complexity of **enqueue** and **dequeue** in terms of big O-notation? Justify your answer.

(5 points)

enqueue is $O(1)$: we can insert the element at the end of the list
dequeue is $O(n)$: searching the minimal element is $O(n)$, removing it requires moving the elements behind $O(n)$

(b) Again, consider a priority queue using an unsorted array. What is the time complexity for sorting an (unsorted) list using this priority queue? Justify your answer.

(5 points)

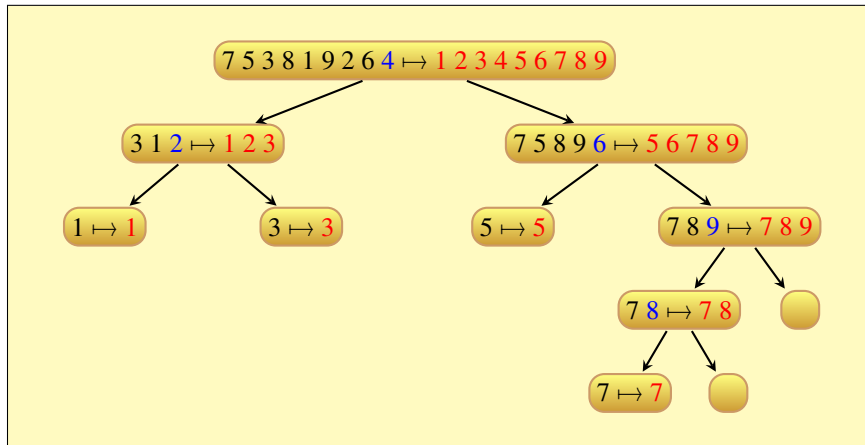
The time complexity is $O(n^2)$. We need to **enqueue** n elements and then **dequeue** n elements, that is, n -times $O(1)$ plus n -times $O(n)$.

(c) Give the quick-sort tree for sorting the following list:

7, 5, 3, 8, 1, 9, 2, 6, 4

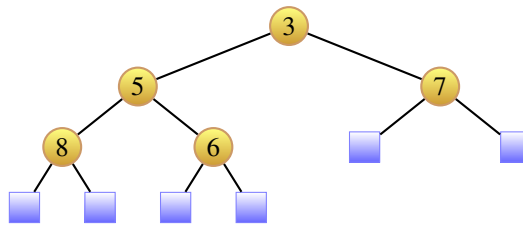
As the pivot elements pick always the rightmost element of the list.

(5 points)



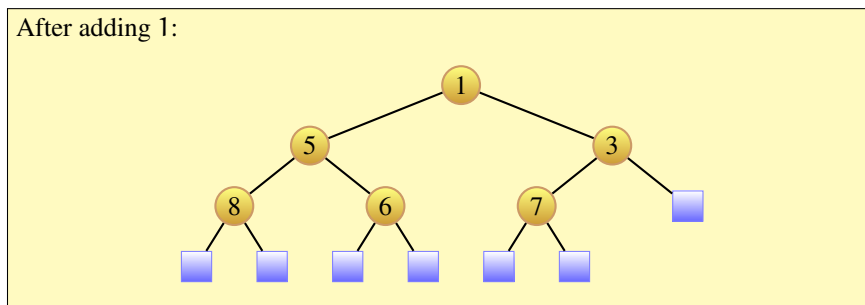
Task 3.

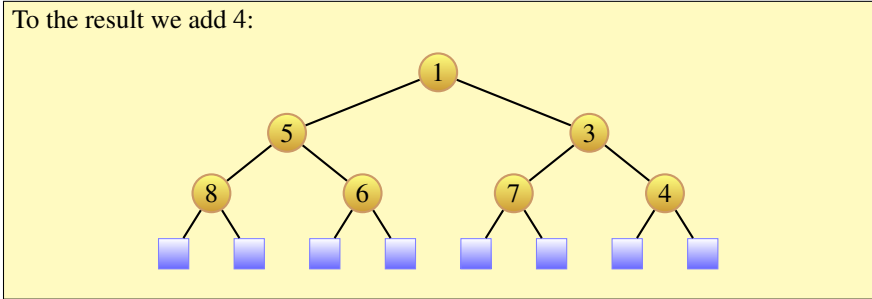
(a) Add 1 to the following heap (final heap suffices):



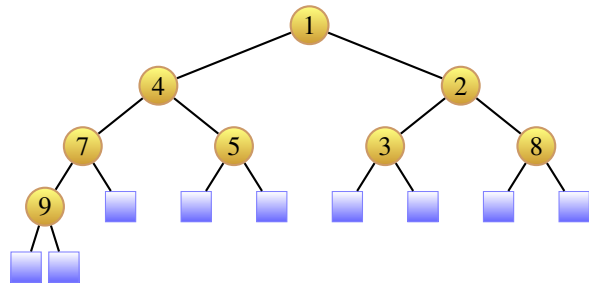
To the resulting heap add 4 (final heap suffices).

(5 points)

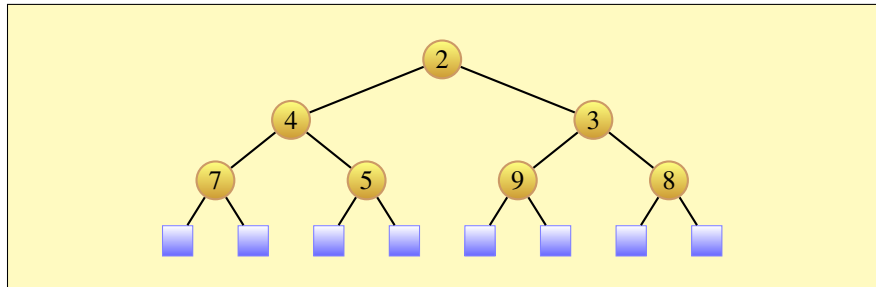




(b) Remove the key 1 from the following heap (final heap suffices):

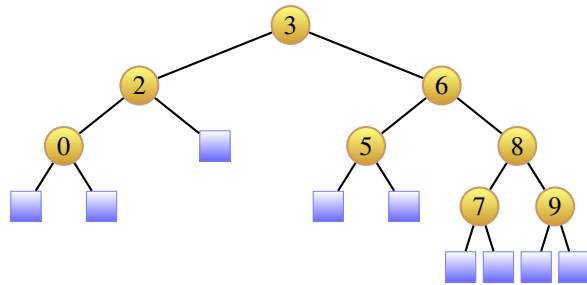


(5 points)



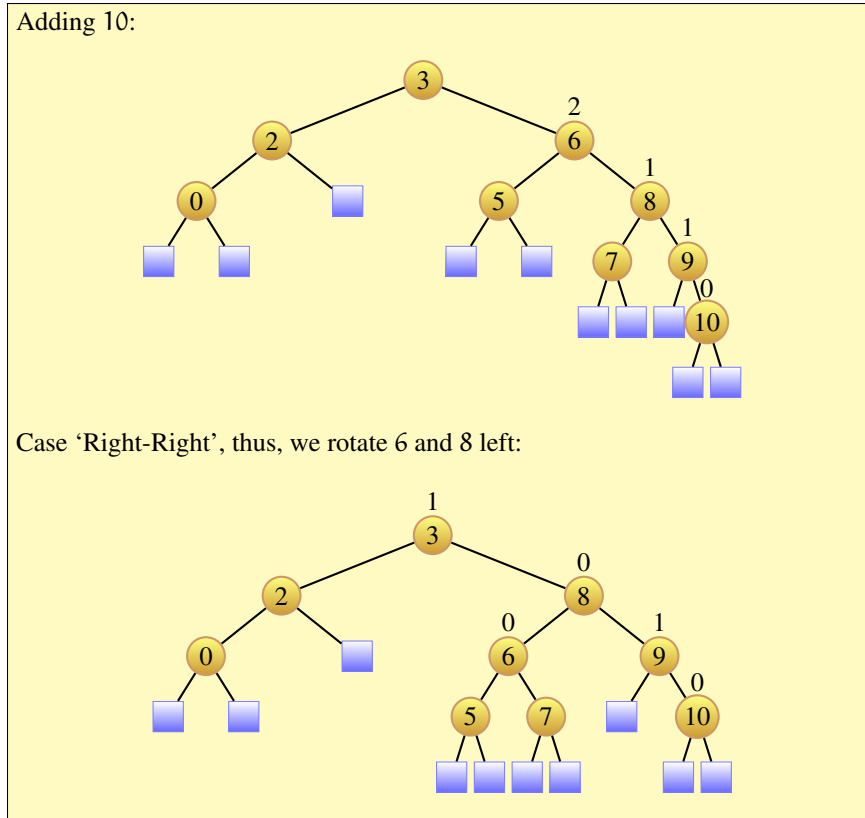
Task 4.

Given is the following AVL tree:



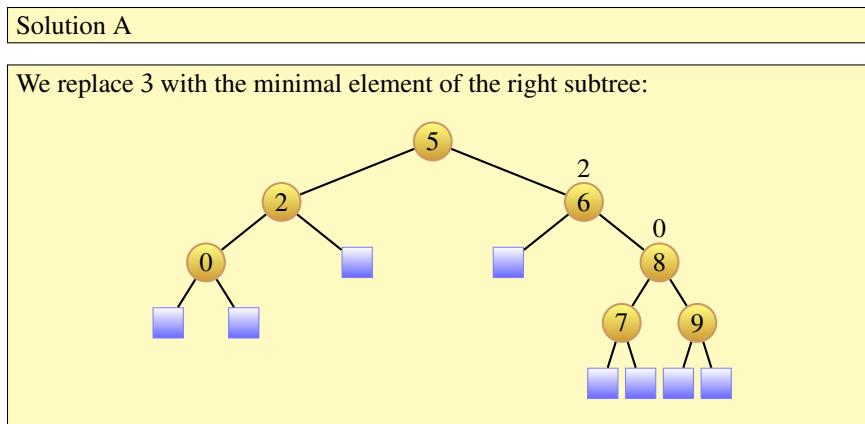
(a) Indicate step by step how a node with key 10 is added.

(5 points)

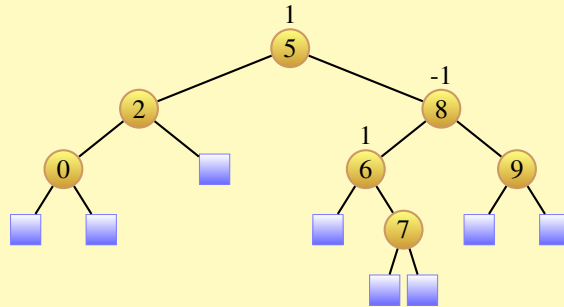


(b) Indicate step by step how the node with key 3 is removed.
(Work with the given tree, not the result of (a).)

(5 points)

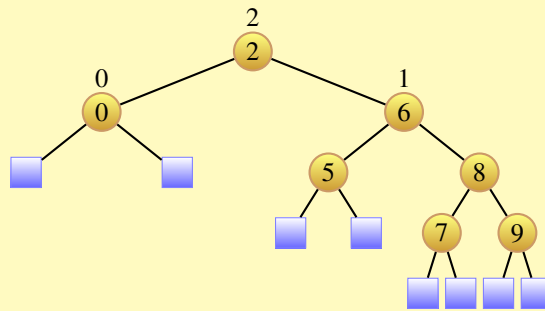


Case 'Right', thus, we rotate 6 and 8 left:

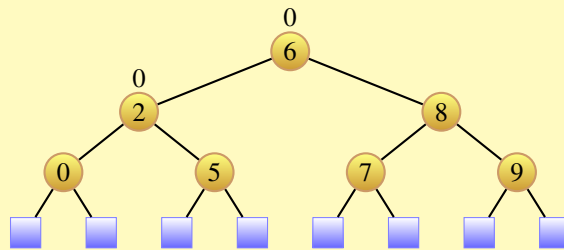


Solution B

We replace 3 with the maximal element of the left subtree:



Case 'Right-Right', thus, we rotate 2 and 6 left:

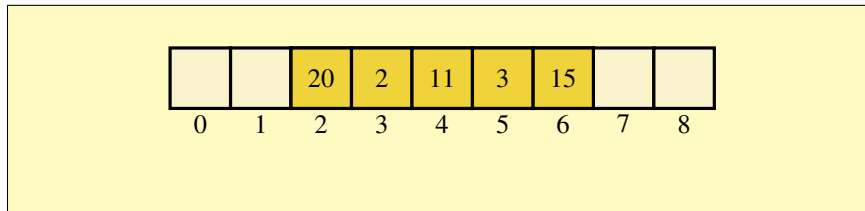


Task 5.

- (a) Make a hash table of size 9. Use the hash function $h(k) = k \bmod 9$. Add using linear probing to the initial empty hash table the following numbers in this order:

20, 15, 2, 11, 3

(5 points)



(b) What is the:

- (i) average (expected), and
- (ii) worst-case

time complexity (O-notation) of searching a key k in a hash table (using chaining)?

(5 points)

average: $O(1)$
 worst-case: $O(n)$

(c) What is the:

- (i) average (expected), and
- (ii) worst-case

time complexity (O-notation) of searching a key k in a AVL tree?

(5 points)

average: $O(\log n)$
 worst-case: $O(\log n)$

Task 6.

We consider the following text/string T:

a	b	a	c	a	b	a	b	a	c	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

together with the pattern P:

a	b	a	c	a	b	c
0	1	2	3	4	5	6

Use the following pattern matching algorithms to find the pattern P in T:

(a) Boyer-Moore *(5 points)*

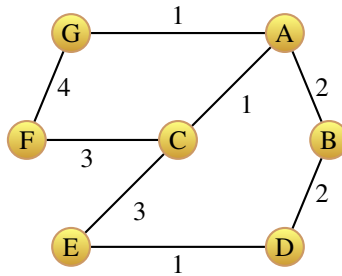
(b) Knuth-Morris-Pratt *(5 points)*

Indicate for every step which letter of the pattern is compared with which letter of the text, and how far the pattern is moved further. Example for the brute-force algorithm:

1. $P[0] = T[0]$
 2. $P[1] = T[1]$
 3. $P[2] = T[2]$
 4. $P[3] = T[3]$
 5. $P[4] = T[4]$
 6. $P[5] = T[5]$
 7. $P[6] \neq T[6]$
 8. moving pattern 1 further (now position 1)
 9. $P[0] = T[1]$
- ...

Task 7.

Given is the following graph:



- (a) Show step for step how the minimal spanning tree is computed using the Prim-Jarnik algorithm. Thereby start with the node G and indicate the order in which nodes are processed.

(5 points)

The nodes are processed in the order: G, A, C, B, D, E, F, and the result is:

- (b) What is the time complexity of Prim-Jarnik algorithm (O -notation)? Briefly justify your answer.

(5 points)

The complexity is $O(m \cdot \log n)$ where m is the number of edges, and n is the number of nodes of the graph. We store the distance of non-processed nodes to the set of processed nodes in a Heap-based priority queue. We need to remove n nodes from the queue, this is, n times $O(\log n)$, and for every edge we need to update the distance of a node, this is, m times $O(\log n)$. For connected graphs we have $m \geq n - 1$, thus the whole process is $O(m \cdot \log n)$.

Task 8. Optional Tasks for Extra Points

- (a) Describe a data structure where the dictionary operations (adding and searching of keys) run in expected time $O(1)$ and worst-case time $O(\log n)$. A brief indication of the idea suffices. Hint: think of combining two known data structures.

(5 points)

Think of a hash map with chaining: every cell of the table contains a reference to a linked list. Now instead of linked lists we use AVL trees. That is, we use a hash map where each cell contains a reference to an AVL tree containing the key-value pairs mapped to the corresponding cell of the table. Then the expected time for adding and searching is $O(1)$, and in worst-case, when all keys are mapped to the same cell, we have $O(\log n)$ for the respective operations on the AVL tree.

- (b) Prove that the worst-case height of an AVL tree is $O(\log_2 n)$ (where n is the number of nodes of the tree).

(5 points)

Let $n(h)$ denote the minimal number of nodes of an AVL tree of height h :

$$n(1) = 1$$

$$n(2) = 3$$

$$n(h) = 1 + n(h-1) + n(h-2) \quad \text{for } h > 2$$

Moreover $n(h-1) > n(h-2)$, hence we obtain:

$$n(h) \geq 2 \cdot n(h-2)$$

and by iterated substitution we get:

$$\begin{aligned} n(h) &\geq 2 \cdot n(h-2) \\ &\geq 2^2 \cdot n(h-2 \cdot 2) \\ &\geq \dots \\ &\geq 2^k \cdot n(h-k \cdot 2) \\ &\geq 2^{h/2} \cdot n(1 \text{ or } 2) \geq 2^{h/2} \end{aligned}$$

Thus $\log_2 n(h) \geq \log_2 2^{h/2}$, and hence $h \leq 2 \cdot \log_2 n \in O(\log_2 n)$.

The grade is the (total amount of points plus 10) divided by 10.