Exam Data Structures and Algorithms 2018-2019

Tuesday October 23, 15.15-18.00

6 exercises

Exercise 1. *(5+5+5 points)*

This exercise is concerned with sorting.

(a) Give in terms of $\mathcal{O}$ the worst-case time complexity of insertion sort, selection sort, merge sort, quicksort, heapsort, and the average-case time complexity of bucket sort. No motivation needed.

(b) Consider the array $A = [5, 1, 2, 4, 3, 6]$.

Apply 'on the fly' heapsort to A. You may give your answer in pictures.

(c) Consider the pseudo-code for partition; the input is an array A of natural numbers, and indices p and r of A.

```
Algorithm partition(A, p, r):
    x := A[r]
    i := p − 1
    for j = p to r − 1 do
        if A[j] ≤ x then
            i := i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

Give pseudo-code for the algorithm quicksort that takes as input an array, and two indices in that array. You may use the algorithm partition.

Exercise 2. *(5+5 points)*

This exercise is concerned with sorting.

(a) Give an example of (i) a sorting algorithm that can be in-place; (ii) a sorting algorithm that in best-case performs in $\Theta(n)$ with n the size of the input-array; (iii) a sorting algorithm that is stable. No motivation needed.

(b) Statement: it is possible that an algorithm for max-heapify in $\Theta(1)$ will be found. Is this statement true or false? Explain your answer.

Exercise 3. *(4+6+5 points)*

This exercise is concerned with linear data structures.

(a) Suppose we wish to implement a max-priority queue. Give for the operations for inserting and deleting the worst-case time complexity in terms of $\mathcal{O}$, for the case we implement our max-priority queue using (i) an unordered array; (ii) an ordered array; (iii) a max-heap. No motivation needed.

(b) Give an implementation (use pseudo-code) of a stack with operations push and pop, both in $\mathcal{O}(1)$, using a singly linked list, where we have the following: for an element x in the list, we have operations $x.next$ and $x.key$ with the suggested meaning. For a list L we have operation $L.head$.

Also informally explain why your operations have the desired complexity.

(c) We consider a hash table of size m. We solve collisions by open addressing with double hashing. Explain informally how to add an item with key k.

Exercise 4. *(4+5+3+5 points)*

This exercise is concerned with binary search trees and AVL-trees.

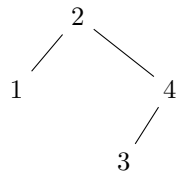
(a) What is the worst-case time complexity, in terms of $\mathcal{O}$, of adding an element to (i) a Binary Search Tree (BST) consisting of n elements, (ii) a min-heap consisting of n elements, (iii) a AVL-tree consisting of n elements? No motivation needed.

(b) Construct an AVL-tree by inserting one by one the keys

5 3 4 6 7 2 1

starting from the empty tree. After each insertion, rebalance the tree if needed. Give your answer in pictures.

(c) Give in a picture the result of removing the node with key 2 from the following binary seach tree:



(d) Consider binary search trees where all keys are different. Consider (informally) the operations for insertion and deletion. Give in (a) picture(s) an example showing that first inserting and then deleting can give another result than first deleting and then inserting.

Exercise 5. *(5+5+5 points)*

Consider the Fibonacci numbers defined by $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1}+F_{i-2}$ for $i \geq 2$.

(a) Give a naive recursive algorithm for computing the nth Fibonacci number.

(b) Give a recurrence equation describing the worst-case time complexity of your naive recursive program, and solve it to give the worst-case time complexity for computing the nth Fibonacci number.

(c) Give a dynamic programming algorithm for computing the nth Fibonacci number that is in $\mathcal{O}(n)$; explain informally why it is in $\mathcal{O}(n)$.

Exercise 6. *(5+4+5+4 points)*

Consider the dynamic programming algorithm for knapsack01:

Algorithm knapsack01(S, W):
  new $B[0 \ldots n, 0 \ldots W]$
  for $w := 0$ **to** W **do**
    $B[0, w] := 0$
  for $k := 1$ **to** n **do**
    $B[k, 0] := 0$
    for $w := 1$ **to** W **do**
      if $w_k \leq w$ **then**
        $B[k, w] := \mathsf{max}(B[k-1, w], B[k-1, w-w_k] + b_k)$
      else
        $B[k, w] := B[k-1, w]$

(a) Apply the algorithm to maximal weight $W = 5$ and the following set S with items with benefit and weight:

	b	w
s_1	3	2
s_2	2	1
s_3	4	3

Give your answer in the form of a table representing B, with from left to right w increasing, and from top to bottom k increasing.

(b) Explain why for the knapsack01 problem, the greedy choice for an item with the largest benefit does not necessarily lead to an optimal solution.

(c) Adapt the algorithm so that is uses only an array $B[0 \ldots W]$.

(d) We adapt the setting: every item has benefit 1. Can we simplify the algorithm, and if so, how? Explain informally.