EPFL

Midterm Exam, Algorithms 2019-2020

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudo-code without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to algorithms covered in class without reproving their properties.
- **Do not touch until the start of the exam.**

Good luck!

Name: _____ **N° Sciper:** _____

Problem 1 / 10 points	Problem 2 / 34 points	Problem 3 / 26 points	Problem 4 / 16 points	Problem 5 / 14 points

Total / 100

1 *(10 pts)* **Basic questions.**

1a *(4 pts)* Answer whether the following statements are **true** or **false**.

The k-th largest element of an array A can be found in $O(n + k \log n)$ time by running the first k iterations of HeapSort. True or false?

$n^{1+O(1/\log n)} = O(n \log n)$. True or false?

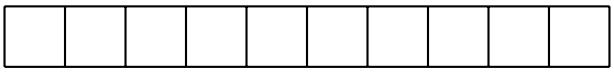
The best case runtime of InsertionSort is $\Omega(n^2)$. True or false?

If $f(n) = O(g(n))$, then $10^{f(n)} = O(10^{g(n)})$ True or false?

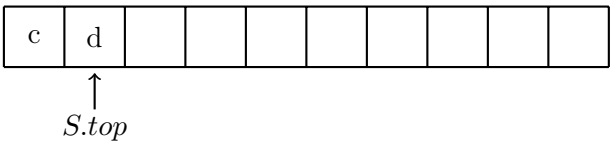
1b *(3 pts)* Consider the queue Q below (assume the implementation shown in class):

What is the resulting queue Q after the following operations: Enqueue(Q, c), Enqueue(Q, d), Dequeue(Q), Enqueue(Q, e)? Specify the content of the array used to implement the queue as well as the values of the head and tail pointers.

Solution:



1c *(3 pts)* Consider the following stack S implemented in the same way as seen in class:



What is the resulting stack S after the following operations: Push(S, e), Push(S, f), Pop(S), Push(S, e), Push(S, g), Pop(S)?

Solution:

2 *(34 pts)* **Recurrences.** Consider the following algorithms XYZ and ABC that take as input an array A and two indices low and $high$ in the array:

```
XYZ(A, low, high)
1.  if low ≥ high
2.        return
3.  else
4.        for i = low to high
5.              print A[i]
6.        mid = ⌊(low + high)/2⌋
7.        XYZ(A, low, mid)
8.  return
```

```
ABC(A, low, high)
1.  XYZ(A, low, high)
2.  if low ≥ high
3.        return
4.  else
5.        p = low + ⌊(high − low)/4⌋
6.        q = ⌊(low + high)/2⌋
7.        ABC(A, low, p)
8.        ABC(A, p + 1, high)
9.        ABC(A, p + 1, q)
10.       ABC(A, low, p)
11.       ABC(A, q + 1, high)
12. return
```

2a *(10 pts)* Let $S(n)$ be the time it takes to execute ABC$(A, low, high)$ with $n = high - low + 1$, and let $T(n)$ denote the time it takes to execute XYZ$(A, low, high)$ with $n = high - low + 1$. **Give the recurrence relations** for $S(n)$ and $T(n)$. To simplify notation, you may ignore floors and ceilings in your recurrence.

Solution:

2b *(24 pts)* **Prove** tight asymptotic bounds on $S(n)$ and $T(n)$.

Solution:

3 *(26 pts)* **Can you find the palindrome?** In this problem your task is to find the longest palindromic subsequence of a given sequence of characters. Recall that a sequence is called a palindrome if it does not change when reversed. For example, `ABBA` and `ABDBA` are palindromes, whereas `AABB` is not.

Consider a sequence $s = (s_1, s_2, \ldots, s_n)$ of n characters. We say that a sub-sequence $s' = (s[i_1], s[i_2], \ldots, s[i_k])$, where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, is a palindromic subsequence if the subsequence does not change if we reverse it. Formally, s' is palindromic if $s[i_j] = s[i_{k-j+1}]$ for every $j = 1, \ldots, k$. In this problem your task is to design a dynamic programming algorithm that given a sequence s of n characters as input, finds a longest palindromic subsequence of s. For example, if $s =$ `ACBDBA`, the longest palindromic subsequence is `ABDBA`.

Input:	Sequence $s = (s_1, \ldots, s_n)$ of n characters.
Output:	A longest palindromic subsequence of s.

Design and analyze a dynamic programming solution for the problem. For full credit your algorithm should run in time $O(n^2)$.

3a For $1 \leq i \leq j \leq n$ let $d(i, j)$ denote the length of the largest palindromic subsequence in the substring $s[i], s[i+1], \ldots, s[j]$. Write a recursive formula for $d(i, j)$.

Solution:

3b Give a bottom up implementation of your recursion from **3a** and analyze its running time.

Solution:

(Solution to problem 3 continued)

4 *(16 pts)* **Merging Binary Search Trees.** Suppose that you are given k (not necessarily balanced) binary search trees $T_1, T_2 \cdots, T_k$ each containing n/k integers. Give an $O(n \log k)$ time algorithm for merging the trees $T_1, \ldots, T_k$ into a single *balanced* binary search tree T^*: the height of T^* must be $O(\log n)$. See Fig. 1 for an example.

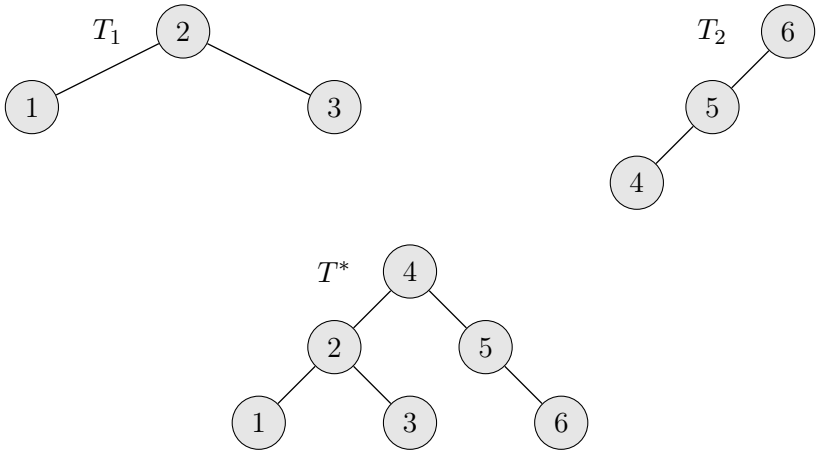


Figure 1. Input instance with $k = 2$ and $n = 6$. The tree T^* is the result of merging T_1 and T_2.

Input: Binary search trees T_i, $i = 1, \ldots, k$ containing n/k integers each. The T_i's are not necessarily balanced.

Output: A binary search tree T^* containing all n integers. The height of T^* must be bounded by $O(\log n)$.

Solution:

(Solution to problem 4 continued)

5 *(14 pts)* **Median of two sorted arrays.** In this problem you are given two sorted arrays with n distinct integers $a_1 < a_2 < \ldots < a_n$ and $b_1 < b_2 < \ldots < b_n$, and your task is to find the median in the union of the arrays. Let $c = \{a_1, a_2, \ldots, a_n, b_1, \ldots, b_n\}$ denote the union of the two arrays and let $c_1 \leq c_2 \leq \ldots \leq c_{2n-1} \leq c_{2n}$ denote the elements of c in sorted order. Recall that the median of c is $(c_n + c_{n+1})/2$. For example, suppose that $n = 4$, $a_1 = 1, a_2 = 3, a_3 = 5, a_4 = 7$ and $b_1 = 2, b_2 = 4, b_3 = 6, b_4 = 8$. Then $c_i = i$ for $i = 1, \ldots, 8$, and the median is $(c_4 + c_5)/2 = 4.5$.

Input: Two sorted arrays $a_1 < a_2 < \ldots < a_n$ and $b_1 < b_2 < \ldots < b_n$ of n distinct elements each.
Output: The median of the union of a and b.

For simplicity you may assume that the two arrays do not share any elements, i.e. all elements in c are distinct.

Design and analyze an algorithm for the problem. For full credit your solution should run in $O(\log^2 n)$ time.

Solution: