ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Final Exam, Algorithms 2017-2018

- 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 pseudocode 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.
- Attached at the end of the exam is a French translation.
- **Do not touch until the start of the exam.**

Good luck!

Name: _____ **N° Sciper:** _____

Problem 1 / 23 points	Problem 2 / 14 points	Problem 3 / 16 points	Problem 4 / 17 points	Problem 5 / 18 points	Problem 6 / 12 points

Total / 100

1 *(23 pts)* **Basic questions.** This problem consists of five subproblems (1a-1e) for which you **do not** need to motivate your answers.

1a *(5 pts)* Arrange the following functions in increasing order according to asymptotic growth.

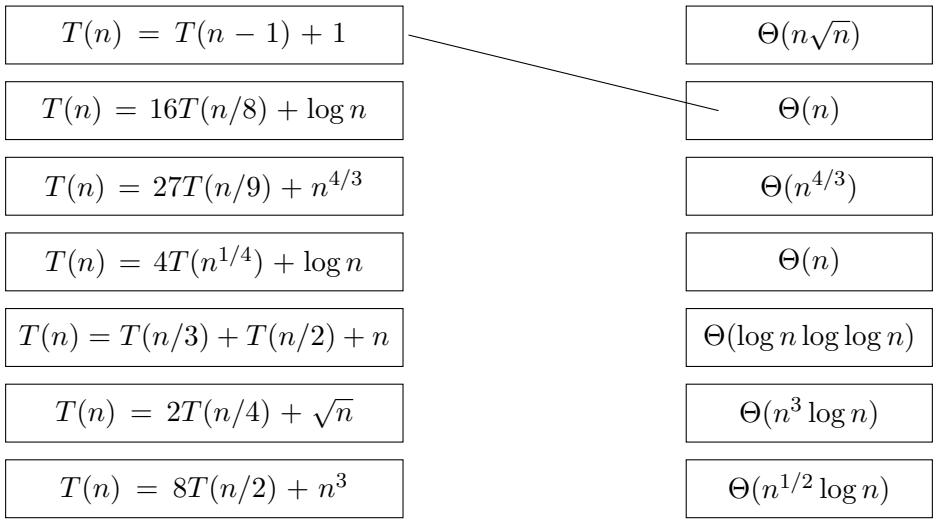
$$n^{\sqrt{n}},\ \log n,\ 3^n,\ n(\log n)^5,\ n^4 \log n,\ \log(n!)$$

(In this problem, you only need to give the right order, i.e., you do not need to explain your answer.)

Solution:

1b *(6 pts, 1 pt for each correct pair)* Pair up each of the recurrence relations on the left side with the correct asymptotic growth function on the right side (assuming that $T(1) = 1$). For example the recurrence $T(n) = T(n-1) + 1$ has asymptotic growth $\Theta(n)$.

Solution:



1c *(4 pts)* Consider the recurrence $T(n) = T(\alpha n) + T(\beta n) + n^2$, $T(1) = \Theta(1)$, for some constants $\alpha, \beta \in (0, 1)$ that satisfy $\alpha^2 + \beta^2 = 1$. Give a tight asymptotic bound for $T(n)$. You do not need to motivate your answer.

Solution:

The answer is $T(n) = \Theta(\_\_\_\_\_)$

1d *(4 pts)* Consider the recurrence $T(n) = T(\alpha n) + T(\beta n) + n^2$, $T(1) = \Theta(1)$, for some constants $\alpha, \beta \in (0, 1)$ that satisfy $\alpha^2 + \beta^2 < 1$. Give a tight asymptotic bound for $T(n)$. You do not need to motivate your answer.

Solution:

The answer is $T(n) = \Theta(\_\_\_\_\_)$

1e *(4pts)* Suppose that you want to store n keys in a hash table with m slots. Assume simple uniform hashing, and define a 4-collision as a set of four **distinct** keys $\{k_1, k_2, k_3, k_4\}$ that are hashed into the same slot.

Give a tight asymptotic expression for the expected number of 4-collisions.

Solution:

The expected number of 4-collisions is $\Theta(\_\_\_\_\_)$

2 *(14 pts)* **Maintaining the second smallest element.** In this question you should design and analyze a data structure that supports the following operations:

- INSERT(x): add an integer x to the data structure
- FINDSECONDSMALLEST(): return the second smallest element if the data structure contains more than one element and NULL otherwise
- DELETESECONDSMALLEST(): remove the second smallest element (if the data structure contains more than one element)

For full credit the operations must have the following time complexities: $O(\log n)$ for INSERT, $O(1)$ for FINDSECONDSMALLEST, $O(\log n)$ for DELETESECONDSMALLEST.

In this problem you are required to (a) describe your data structure and (b) analyze runtime of the three operations above.

Solution:

3 *(16 pts)* **String similarity.** In this problem you will design an algorithm that, given two strings s and t, outputs the length of the **longest common substring** between s and t.

- **Input:** The input consists of two strings s and t of length n and m respectively.
- **Output:** The length of the longest common substring: the largest k such that there exist $1 \leq i \leq n-k+1, 1 \leq j \leq m-k+1$ such that $s[i : i+k-1] = t[j : j+k-1]$. Here $s[a : b]$ stands for the substring of s corresponding to indices $a, a+1, \ldots, b$.

For example, the maximum common substring between $s =$ `ABAABABBAB` and $t =$ `AAABBAA` is `ABBA`, of length 4. Indeed, $s[6 : 9] = t[3 : 6] =$ `ABBA`, and there is no common substring of length larger than 4.

Give a dynamic programming solution to the problem with runtime $O(nm)$. **Note: this is similar, but not the same as the longest common subsequence problem covered in class.**

In this problem you are required to (a) define the subproblems and the recurrence, (b) analyze the runtime of a bottom-up implementation of your recurrence.

Solution:

Continuation of the solution to 3:

4 *(17 pts)* **Algorithms on a grid.** You are given an $n \times n$ grid of cells, with every grid cell containing one of the letters `A`, `L`, `G`, `O`. Your task is to find as many non-overlapping spellings of the word `ALGO` on the grid as possible (see Fig. 1). Formally, a spelling of `ALGO` on the grid is a sequence of grid cells $(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4) \in \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ such that the letter in position (i_1, j_1) is `A`, the letter in position (i_2, j_2) is `L`, the letter in position (i_3, j_3) is `G`, the letter in position (i_4, j_4) is `O`, and consecutive cells are adjacent.

We say that two cells (i, j) and (i', j') are adjacent on the grid if either $i = i'$ and $|j - j'| = 1$ or $|i - i'| = 1$ and $j = j'$. Two spellings are called non-overlapping if they do not share cells.

Input: An $n \times n$ array A with $A_{ij} \in$ `{A, L, G, O}` for every $1 \le i \le n, 1 \le j \le n$.

Output: The largest $k \ge 0$ such that there exist k **non-overlapping** spellings of `ALGO` on the grid.

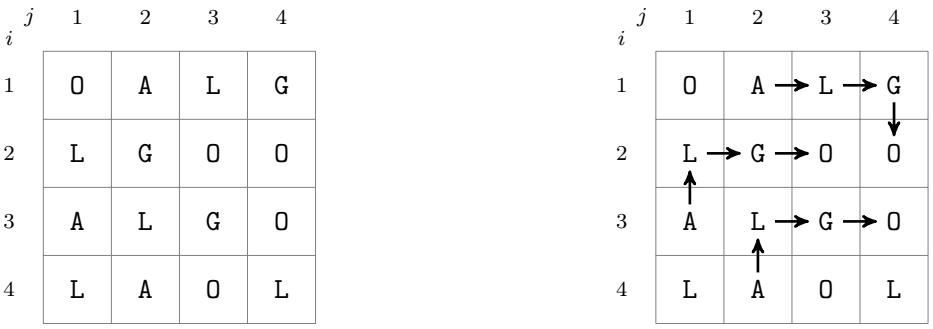


Figure 1. 4×4 instance of the problem (left) and the optimal solution (right).

Design and analyze an algorithm for the problem. Your algorithm should run in time polynomial in n. *Hint: construct a graph G with a source s and a sink t such that the value of max-flow from s and t in G is exactly the maximum number of non-overlapping spellings.*

In this problem you are required to (a) describe an efficient algorithm and (b) analyze its runtime.

Solution:

Continuation of the solution to 4:

5 *(18 pts)* **Minimum-weight forests.** In this problem you are given a connected undirected graph $G = (V, E)$ with non-negative edge weights w, and your task is to **design and analyze** an efficient algorithm that computes the minimum-weight forest in G with exactly two connected components.

Input: Connected undirected graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}$.

Output: Minimum-weight forest in G with exactly two connected components.

For full credit, your algorithm should run in time $O((|V| + |E|) \log |V|)$.

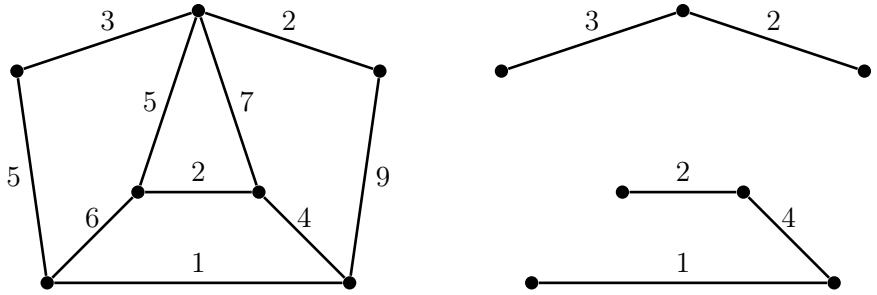


Figure 2. Example of input graph G with edge weights (left), and its minimum-weight forest with two components (right).

In this problem you are expected to (a) design an efficient algorithm (b) prove its correctness and (c) analyze its runtime.

Solution:

Continuation of the solution to 5:

6 *(12 pts)* **Route intersections revisited.** The infinite glacier in the Swiss Alps has the following structure: two perfectly parallel walls (North and South) with a river of ice flowing between them. We model the South wall as the line $y = 0$ in the plane, and the North wall as the line $y = 1$ (see Fig. 3). Now n companies want to establish Tyrolean routes across the glacier, and each company designated two climbers, who occupy positions on the two walls and have a rope between them. The ropes may cross, and in this problem your task is to design an efficient algorithm for **counting the number of crossings**.

Input: Positions $a_i, i = 1, \ldots, n$ of n climbers on the North wall. The first climber in the i-th pair occupies position $(a_i, 1)$ on the North wall, and the second occupies position $(i, 0)$ on the South wall (see Fig. 3). The rope between them is the line segment connecting $(a_i, 1)$ to $(i, 0)$ in the plane.

Output: Your task is to compute **the number of intersections** of the line segments.

You may assume that n is a power of two if this simplifies your analysis, as well as that no two climbers occupy the same position, and no three line segments intersect at the same point. You may use the fact that a segment connecting $(i, 0)$ to $(a_i, 1)$ intersects a segment connecting $(j, 0)$ to $(a_j, 1)$ if and only if $i < j$ and $a_i > a_j$ or $i > j$ and $a_i < a_j$.

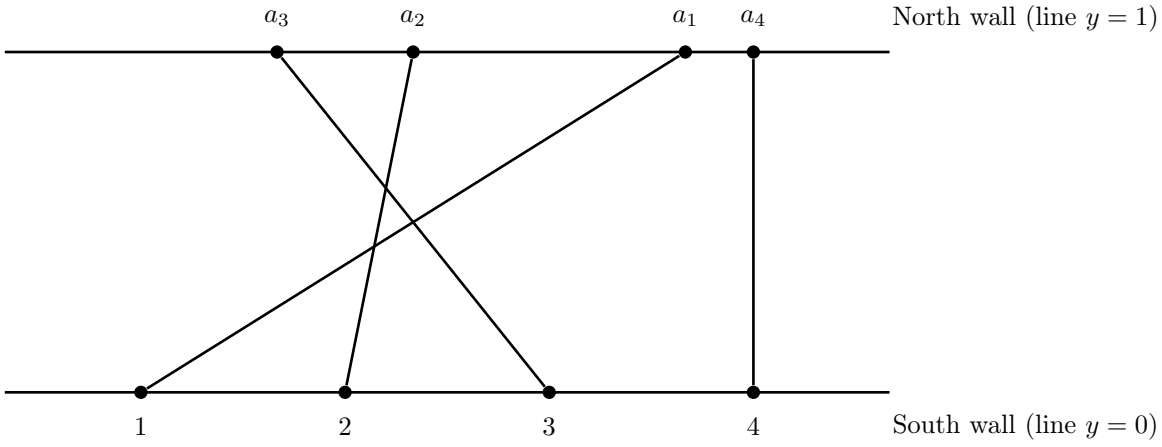


Figure 3. An example problem instance with $n = 4$. The number of intersection points is 3. Note that only the x-coordinates of the climbers are shown, as all climbers on the North wall have their y-coordinates equal to 1, and climbers on the South wall have their y-coordinates equal to 0.

For full credit your algorithm should run in $O(n \log n)$ time. *Hint: the problem can be solved using an adaptation of merge-sort.*

Solution:

Continuation of the solution to 6: