ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Final Exam, Algorithms 2018-2019

- 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 / 22 points	Problem 2 / 14 points	Problem 3 / 18 points	Problem 4 / 18 points	Problem 5 / 14 points	Problem 6 / 14 points

Total / 100

1 *(22 pts)* **Basic questions.** This problem consists of five subproblems (1a-1e) for which you **do not** need to motivate your answers.

1a *(6 pts)* **Sorting**

Insertion Sort and Quick Sort have the same worst case running time. True or False?

Both Heap Sort and Merge Sort require linear extra space to sort, i.e. are **not** in place. True or False?

Let $A[1\ldots 6] = \begin{array}{|c|c|c|c|c|c|}\hline 1 & 3 & 7 & 4 & 5 & 6 \\ \hline \end{array}$ be an array consisting of 6 numbers. If we use randomized quick sort to sort A, the probability that $A[2] = 3$ and $A[5] = 5$ are compared is 1/2. True or False?

1b *(4 pts)* **Uniform Hashing**
Suppose you are hashing n elements into m slots using uniform hashing. Asymptotically, what is the value of m in terms of n that ensures that

(A) Expected number of collisions is $\Theta(\sqrt{n})$.

$m = \Theta(\underline{\hspace{2cm}})$

(B) Expected number of elements mapped to any given slot of the table is $\Theta(1)$.

$m = \Theta(\underline{\hspace{2cm}})$

1c *(8 pts)* **Recurrences**

Consider the functions Foo(n) and Bar(n), whose pseudocodes are given below. The functions take as input an integer n.

```
Foo(n)
if n > 100
    u= ⌊n/3⌋
    Foo(u)
    Foo(n − u)
    Bar(⌊√n⌋)

Bar(n)
if n > 1000
    Bar(n − 1)
    for i = 1 to n
        Print ('Ok')
```

Denote the running time of Foo(n) by $T(n)$, denote the running time of Bar(n) by $S(n)$.

Write down the recurrence relation for $T(n)$ in terms of $S(n)$.

$T(n) =$______________________________________

Write down the recurrence relation for $S(n)$.

$S(n) =$______________________________________

What is the runtime of BAR(n) asymptotically as a function of n?

$S(n) = \Theta($__________$)$

What is the runtime of FOO(n) asymptotically as a function of n?

$T(n) = \Theta($__________$)$

1d *(2 pts)* Consider the recurrence $T(n) = T(\alpha n) + T(\beta n) + T(\gamma n) + n^3$, $T(1) = \Theta(1)$, for some $\alpha, \beta, \gamma \in (0, 1)$ that satisfy $\alpha^3 + \beta^3 + \gamma^3 = 1$. Give a tight asymptotic bound for $T(n)$. You do not need to motivate your answer.

Solution: The answer is $T(n) = \Theta($__________$)$

1e *(2 pts)* Consider the recurrence $T(n) = T(\alpha n) + T(\beta n) + T(\gamma n) + n^2$, $T(1) = \Theta(1)$, for some $\alpha, \beta, \gamma \in (0, 1)$ that satisfy $\alpha^2 + \beta^2 + \gamma^2 = 0.999$. Give a tight asymptotic bound for $T(n)$. You do not need to motivate your answer.

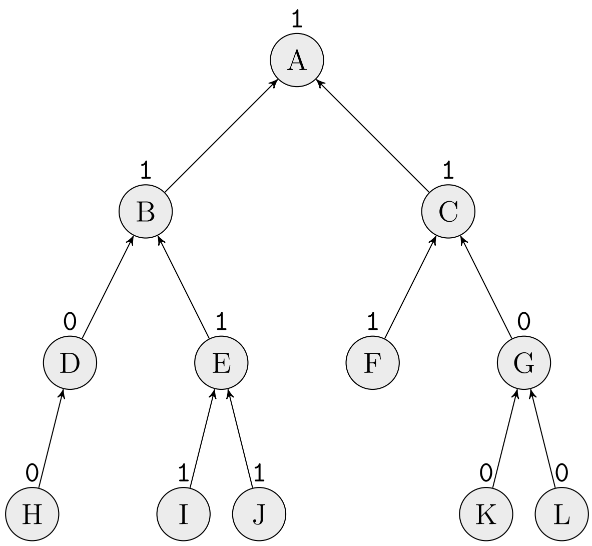
Solution: The answer is $T(n) = \Theta($__________$)$

2 *(14 pts)* **Monochromatic subtrees.** For a rooted binary tree $T = (V, E)$ with nodes labeled with zeros and ones we would like to count the number of nodes $u \in V$ whose subtrees are monochromatic (i.e. either all nodes in their subtree are labelled zero or all nodes in their subtree are labelled one).

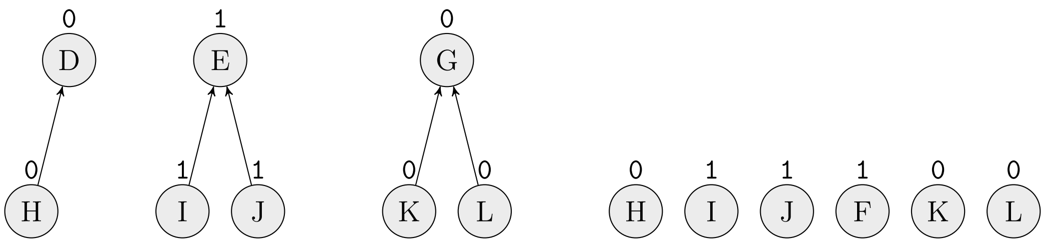
Input: a rooted binary tree $T = (V, E)$ with a label $u.label \in \{0, 1\}$ for every node $u \in T$.

Output: number of nodes $u \in V$ whose subtrees are monochromatic.

For example, consider the tree below (the arrows indicate the child-parent relation):



There are 9 nodes with monochromatic subtrees, namely D, E, G, F, H, I, J, K, L. The subtrees are shown below:



In this problem you are required to (a) explain a correct algorithm with the desired running time and to (b) analyze its running time. For full credit your algorithm should run in $O(V)$ time. You may assume that for every node u of T you have access to left child, right child and parent pointers $u.left, u.right, u.parent$ respectively, and $T.root$ is the root of T.

Solution:

Solution to problem 2 continued

3 *(18 pts)* **Packing dominos.** You bought a new board game: an $n \times n$ grid with every cell either square shaped or circular or inactive, together with an unlimited supply of dominos of a rather non-standard form: a circle attached to a square (see Fig. 1 below; inactive cells are shaded in grey). The domino fits onto the board if its circle part is in a circular cell of the board and its square is in a square cell of the board to the left/right, or above/below the circular part. Design an algorithm that determines, given the shape of the board, the maximum number of non-overlapping dominos that can be simultaneously placed on the board. You cannot use inactive cells.

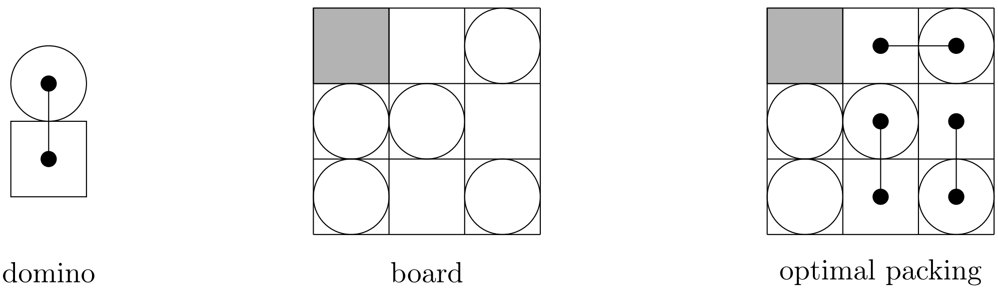


Figure 1. Illustration of the domino (left), a 3×3 board (center) and an optimal packing of 3 dominos onto the board (right).

- **Input:** an integer $n \geq 1$, an $n \times n$ array A with $A_{ij} = 1$ if the (i, j)-th cell is circular and $A_{ij} = 2$ if the (i, j)-th cell is square and $A_{ij} = 0$ if the (i, j)-the cell is inactive.
- **Output:** largest number of non-overlapping dominos that can be placed on the board.

In this problem you are required to (a) describe an efficient algorithm and (b) analyze its runtime. Hint: convert the problem to a graph problem and use an algorithm seen in class.

Solution:

Solution to problem 3 continued

4 *(18 pts)* **Archipelago.** You're trying to navigate a strange city. The city's map is given by a graph $G = (V, E)$ specifying the intersections and streets of the city. Each edge $e \in E$ has a positive integer weight w_e specifying the time it takes to walk along the street. You are currently located in vertex a and want to get to vertex b as quickly as possible.

Luckily we've seen many algorithms in class that are able to tell you the shortest path to b. Unluckily, the city is located on an archipelago, and the graph G is not connected.

To help the citizens get around, there are a number of ferries connecting the islands (or sometimes two points of the same island). You have access to their schedule: It lists T ferries, specifying for each the place of departure (a vertex), the place of arrival (also a vertex), the departure time and the arrival time. Times are given as non-negative integers; it is currently time 0. See Fig. 2 for an illustration.

Design and analyze an algorithm that, given the graph G and the ferry schedule, returns the shortest time needed to get from a to b if you start at a at time 0.

- **Input:** a graph $G = (V, E)$ with nonnegative weights $w : E \to \mathbb{Z}$ given in adjacency list representation. You have access to a function FERRYSCHEDULE. For every $u \in V$ and integer $i \geq 1$ a call to FERRYSCHEDULE(u, i) in $O(1)$ time returns a triple (v, s, t), where $v \in V$ is the destination, s the departure time and t the arrival time of the i-th ferry out of u, and NULL if there are fewer than i ferries out of u.
- **Output:** shortest time needed to get from a to b if you start at a at time 0.

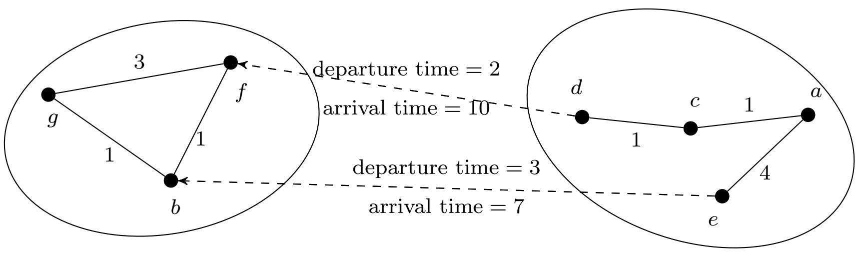


Figure 2. A graph G with two connected components (islands) and two ferries. The ferry from e to be b is faster, but one cannot get to vertex e from vertex a before the ferry's departure time, so the route a, c, d, f, b is optimal.

In this problem you are required to (a) describe an efficient algorithm and (b) analyze its runtime. For full credit your algorithm should run in time $O((E + T) \log V)$ time, where T is the total number of ferries. Hint: adapt an algorithm seen in class.

Solution:

Solution to problem 4 continued

5 *(14 pts)* **Maintaining a minimum spanning tree in a changing graph.** Suppose that you are given a graph $G = (V, E)$ with weights $w : E \to \mathbb{R}$ on edges, and a minimum spanning tree T in G. Now the graph $G' = (V', E')$ is obtained from G by adding a new vertex v together with incident edges (see Fig. 3 below). All edge weights in G' are distinct. Give an efficient algorithm for computing a minimum spanning tree T' in G'.

- **Input:** a graph $G = (V, E)$ with weights $w : E \to \mathbb{R}$ on the edges, a minimum spanning tree T in G. A new vertex v together with incident edges (with weights). All edge weights are distinct.
- **Output:** a minimum spanning tree in the graph G' obtained by adding v with all its edges to G.

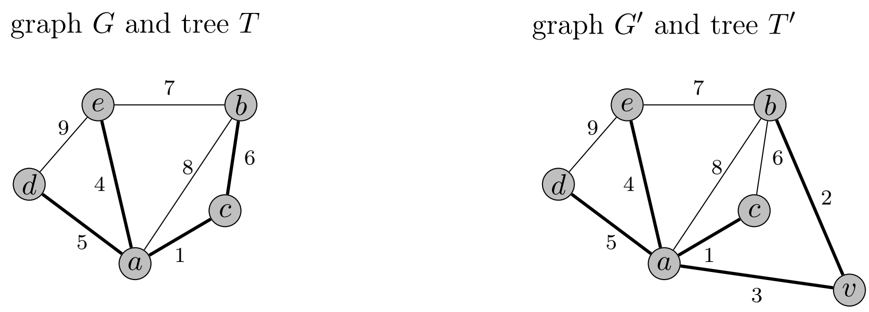


Figure 3. A graph G with its minimum spanning tree T (left) and the graph G' with its minimum spanning tree T' (right). The edges of the spanning trees are shown in bold.

In this problem you are expected to (a) design an efficient algorithm and (b) prove its correctness. For full credit your algorithm should run in $O(V \log V)$ time.

Solution:

6 *(14 pts)* **Largest square submatrix of ones.** Given an $m \times n$ matrix of zeros and ones, find the size of the largest **square** sub-matrix of 1's present in it.

- **Input:** an $m \times n$ matrix C with $C_{ij} \in \{0, 1\}$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$.
- **Output:** Your task is to compute the size of the largest **square** sub-matrix of 1's present in it. Formally, you should find the largest k such that there exist $1 \leq i \leq m - k + 1, 1 \leq j \leq n - k + 1$ such that $C_{i+a,j+b} = 1$ for all $a \in \{0, 1, \ldots, k - 1\}$ and $b \in \{0, 1, \ldots, k - 1\}$.

For example, the size of largest **square** sub-matrix of 1's is 3×3 in the matrix below.

1	0	0	1	1	1	0
1	1	0	1	1	0	1
0	0	1	1	1	1	0
1	1	1	1	1	0	1
1	0	1	1	1	1	0
1	1	0	1	0	0	1

Give a dynamic programming solution to this problem.

In this problem you are required to (a) describe the subproblems, (b) give a recurrence relation and (c) analyze the runtime of a bottom-up implementation of your recurrence. For full credit your solution should run in $O(mn)$ time.

Solution:

Solution to problem 6 continued