Exam

3h. Two A4 sheets (manuscript). We denote $[n] := \{1, \ldots, n\}$.

- Exercice 1 - NP-completeness. We recall that 3-SAT is NP-complete. In this exercise, we assume that the input is an undirected graph G and an integer k.

1. Show that deciding if a graph G has a clique (a set of vertices which are pairwise joined by an edge) of size at least k is NP-complete.
2. Show that deciding if a graph G has a stable set (a set of vertices which are pairwise not joined by an edge) of size at least k is NP-complete.
3. A *vertex-cover* in a graph $G = (V, E)$ is a subset X of V such that every edge $ab \in E$ satisfies that $a \in X$ or $b \in X$. Show that deciding if a graph G has a vertex cover of size at most k is NP-complete.
4. A *feedback vertex set* in a graph $G = (V, E)$ is a subset X of V such that every cycle of G intersects X. Show that deciding if a graph G has a feedback vertex set of size at most k is NP-complete.

- Exercice 2 - Bonus points. The four following exercises correspond to four algorithmic paradigmes : greedy, divide and conquer, randomized algorithm, dynamic programming (not in this order). Find which paradigme is suited for each of them.

- Exercice 3 - Paradigme 1. An array $C[1, \ldots, m]$ is a *sub-array* of $A[1, \ldots, n]$ if there is a strictly increasing function f from $[m]$ to $[n]$ such that $C[i] = A[f(i)]$ for all $i = 1, \ldots, m$. If moreover $f(i) = i + k$ for some constant k, we say that C is a *factor* of A.

1. Propose an algorithm running in $O(n^2)$ time which admits as input two arrays A and B with size n and returns the length of a longest common factor.
2. Same question for sub-array.
3. Deduce an algorithm which computes a longest increasing subarray in an integer array.

- Exercice 4 - Paradigme 2. An *inversion* in an integer array $A[1, \ldots, n]$ (with pairwise distinct entries) is a couple $i < j$ such that $A[i] > A[j]$.

1. Propose an algorithm in $O(n. \log n)$ time which computes the number of inversions of an input A.

- Exercice 5 - Paradigme 3. In this problem, the input is a connected graph $G = (V, E)$. Moreover, there is an edge-label tagging some edges of E as *fragile*, while the other are *reliable*.

1. Propose a polynomial algorithm which returns as output a spanning tree of G with as many reliable edges as possible.
2. * We assume now that G has at least as many edges as vertices. Discuss if your algorithm can be adapted to compute a spanning cycle-tree (i.e. a connected graph with a unique cycle) with a maximum number of reliable edges.

- Exercice 6 - Paradigme 4. Your intership advisor has just invented (again...) a new number generator. An integer n produced by this machine is :

1. either prime. **Case 1**.
2. or admits an integer a which is coprime with n such that $a^{n-1} \neq 1 \mod n$. **Case 2**.

Your task is to write an algorithm which admits as input a number n obtained from his generator and output in which case (1 or 2) n falls. While your advisor is not really interested in your code, he needs it to be efficient (your algorithm should be polynomial in $\log n$).

1. Show that in Case 2, for every integer b, at least one element x among $\{b, ab\}$ satisfies $x^{n-1} \neq 1 \mod n$.
2. Recall that if a is coprime with n, then the multiplication by a is a bijection among integers modulo n. Moreover, if n is prime and n does not divide a, then $a^{n-1} = 1 \mod n$. Propose an algorithm which satisfies your advisor (and such that he will very likely never prove it wrong before the end of your intership).

- Exercice 7 - Search trees. We have seen two kinds of search trees in a connected graph: breadth first search tree (BFS), and depth first search tree (DFS). Their computations involve respectively a queue (FIFO) and a stack (LIFO).

1. Show that if T is a BFS of a connected graph G rooted at r, then every edge xy of G satisfies that the height of x and y (distance to the root) differ by at most 1;
2. Show that if T is a DFS of a connected graph G rooted at r, then every edge xy of G satisfies that x is a descendent of y, or y is a descendent of x in T.
3. Use one of the two notions to test efficiently if a graph is bipartite.
4. Use one of the two notions to orient the edges of a 2-edge connected graph (removing any edge of G leaves G connected) in such a way that there is always a directed path from every vertex x to every vertex y.