Instructions: Please answer the following questions to the best of your ability. If you are asked to develop an algorithm, you must give a plain text description of the algorithm, show that it is correct and prove that it has the required complexity. When writing proofs, please strive for clarity and brevity (in that order). Your work can be either in English or in French.

The exam is closed-book, closed-note, closed-internet, etc. However, you may use one A4-sized sheet (front and back) of notes as reference.

You have 180 minutes to complete the exam. Good luck!
Q. 1 (12 points) Fast multiplication of polynomials. You do not need to prove the properties of complex numbers and you do not need to prove that the determinant of the Vandermonde matrix is non-zero.

Q. 2 (6 points) In order to maintain expected $O(1)$ performance for hash table operations in an $m$-bucket hash table that holds $n$ entries, the load factor $\alpha = n/m$ must be $O(1)$. However, what if we do not know in advance how many items the table needs to hold? Choosing a large $m$ means wasting memory, choosing a small $m$ means poor performance when $n = \omega(m)$. Thus, $m$ will have to grow dynamically. We will analyze a hash table that uses chaining to resolve collisions. Thus, the table is an array of buckets, each of which is a linked list. Elements are inserted at the beginning of the list corresponding to the bucket they hash to; empty buckets are initialized to NIL. We assume the hash function $h(x, m)$ satisfies the assumptions of simple uniform hashing.

Throughout this problem, assume $h(x, m)$ takes $c_h = O(1)$ units of time to compute, while arithmetic operations, memory accesses, and memory allocations take 1 unit.

Our goal will be to maintain the property that $\alpha \leq 2$. We will not worry about reclaiming memory if too many elements are deleted. We say the hash table is full when $n = 2m$. If we try to insert into a full hash table, we need to grow the table in order to maintain the constraint on $\alpha$. Thus, prior to inserting we will do the following:

1. allocate a new array of buckets with $m' = 2m$;
2. for each element $x$ in the hash table, insert $x$ into new bucket $h(x, m')$
3. free the old hashtable (ignore this cost in your analysis).

The second and third steps above are done by iterating over the linked list within each bucket.

(a) (4 points) Analyze the performance of an insertion using the amortized analysis method of your choice. Ignore the cost of deallocating memory in your analysis.

(b) (1 point) Explain why we cannot simply grow the bucket array every time we need to expand the hash table, without rehashing all the elements in the table.

(c) (1 point) Are we restricted to using the same function $h$ when growing the hash table, or can we switch from $h(x, m)$ to some other hash function $h'(x, m')$, if $h'(x, m')$ satisfies the assumptions of simple uniform hashing? Explain.

Q. 3 (6 points) In this exercise, we look at the weighted independent set problem:

**INPUT:** A graph $G = (V, E)$ with a positive weight function $\omega$ on the vertices.

**OUTPUT:** A maximum weighted independent set, i.e. the subset of vertices of maximum total weight such that any two of them are not connected by an edge.

Consider Algorithm 1 that computes an independent set $S$.

(a) (2 points) Let $T$ be any independent set of $G$. Show that for every vertex $u \in T$, either $u \in S$, or $u$ has a neighbor $v$ in $S$ such that $\omega(v) \geq \omega(u)$.

(b) (4 points) Show that Algorithm 1 is a $\frac{1}{\Delta(G)}$-approximation algorithm for the weighted independent set problem, where $\Delta(G)$ is the maximum degree of $G$. 
Algorithm 1 Greedy-MaxInd(G, ω)
1: S ← ∅
2: while V ̸= ∅ do
3: Find the vertex v ∈ V with maximum weight
4: Add v to S
5: Delete v and its neighbors from G.
6: end while
7: return S

Q. 4 (10 points) The goal of this question is to study longest monotone subsequences of an array. Suppose that we are given an array A of size n containing integers A[1], A[2],..., A[n]. A subsequence of length m is a set of indices 1 ≤ i₁ < i₂ < ... < iₘ ≤ n. A subsequence is increasing if for any k, A[iₖ] ≤ A[iₖ₊₁], is decreasing if for any k, A[iₖ] ≥ A[iₖ₊₁], and monotone if it is either increasing or decreasing.

(a) (2 points) Let φ : [1..n] → [1..n] × [1..n] be a function that maps each integer k, 1 ≤ k ≤ n, into a pair of integers: the length of the longest increasing subsequence that ends with index k and the length of the longest decreasing subsequence that ends with k. Show a recursive formula for φ(k). Derive a dynamic programming algorithm that computes the maximal length of a monotone subsequence of A.

(b) (2 points) Modify the algorithm so that it also outputs all monotone subsequences of maximal length.
In what follows, we will only be interested in increasing subsequences.

(c) (4 points) For every i, 1 ≤ i ≤ n, and every k we consider increasing subsequences of A[1], A[2],..., A[i] of length k. Let αᵢ(k) be the last index of such a subsequence chosen so that A[αᵢ(k)] is the smallest possible (if αᵢ(k) is not defined, we set it to plus infinity).
   i. Show that for all i, the sequence A[αᵢ(1)], A[αᵢ(2)],... is increasing.
   ii. Show that αᵢ₊₁ can be obtained from αᵢ by modifying just one element. Give an algorithm with time O(log n) to find this element.
   iii. Show an algorithm based on α that finds a longest increasing subsequence.

(d) (2 points) In TD5, we discussed van Emde Boas trees. Recall that a van Emde Boas tree is a data structure that can efficiently represent a subset of integers from [1, n] so that insertion, deletion, and successor queries take O(log log n) time. Suppose that A is a permutation of 1, 2,..., n. Show an algorithm that finds the longest increasing subsequence in O(n log log n) time using van Emde Boas trees.

Q. 5 In this exercise, we study Karp’s minimum mean-weight cycle algorithm. Let G = (V, E) be a directed graph with a weight function ω : E → R. We define the mean weight of a cycle C as:

\[ \mu(C) = \sum_{e \in E(C)} \frac{\omega(e)}{|C|} \]

Let \( \mu^*(G) = \min \mu(C) \) where C ranges among all cycles of G. A cycle C for which \( \mu(C) = \mu^*(G) \) is called a minimum mean-weight cycle. We are going to design an efficient algorithm for computing \( \mu^*(G) \).
A walk consists of an alternating sequence of vertices and edges consecutive elements of which are incident, that begins and ends with a vertex.\footnote{In other words a walk is like a path but it can use edges several times.}

Given an integer $k$ and two vertices $u$ and $v$, let $\delta_k(u, v)$ be the weight of a lightest walk from $u$ to $v$ with exactly $k$ edges (if this walk uses an edge several times, then the weight of this edge is counted several times). If no such walk exists then $\delta_k(u, v) = +\infty$. Let $\delta(u, v)$ be the weight of a lightest path from $u$ to $v$.

All along the exercise, $s$ denotes a fixed vertex of $G$.

(a) (1 point) Assume that there is an algorithm $ALG$ that computes a minimum mean-weight cycle in strongly connected graph in time $O(nm)$. Prove that in this case we can compute a minimum mean-weight cycle in any directed graphs in $O(nm)$.

From now on we assume that $G$ is strongly connected.

(b) (1 point) Show that if $\mu^*(G) = 0$, then $G$ has no negative-weight cycle and that for all $v \in V$, $\delta(s, v) = \min_{1 \leq k \leq n-1} \delta_k(s, v)$.

(c) (1 points) Show that if $\mu^*(G) = 0$, then for all $v \in V$,

$$\max_{0 \leq k \leq n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} \geq 0$$

(d) (2 points) Let $C$ be a 0-weight cycle and let $u, v$ be two vertices of $C$. Let $x$ be the weight of the path from $u$ to $v$ along $C$. Prove that $\delta(s, v) = \delta(s, u) + x$.

(e) (2 points) Assume that $\mu^*(G) = 0$ and let $C$ be a cycle such that $\mu(C) = 0$. Show that there exists a vertex $v$ in $C$ such that

$$\max_{1 \leq k \leq n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$

(f) (1 point) Assume $\mu^*(G) = 0$. Prove that,

$$\min_{v \in V} \max_{1 \leq k \leq n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} = 0$$

(g) (2 points) Show that if we add a constant $t$ to the weight of each edge, then $\mu^*$ increases by $t$. Show that:

$$\mu^*(G) = \min_{v \in V} \max_{1 \leq k \leq n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k}$$

(h) (2 points) Give a $O(nm)$-time algorithm to compute $\mu^*(G)$. 