Question 1 [8 marks] Assuming that $T(n) \in \Theta(1)$ for all $n \leq n_0$ for a suitable constant n_0, solve each of the following recurrences to obtain an asymptotic bound on their complexity as a closed form. Make your bounds as tight as possible.

a. [4 marks] $T(n) = 4T(n/3) + n^2 \log_2 n$

b. [4 marks] $T(n) = 2T(n/4) + \sqrt{n}$

Question 3 [20 marks] Given a directed acyclic graph $G = (V, E)$ with vertices $G.V$ and edges $G.E$, and a subset $X \subseteq G.V$ of the vertices in G, write an efficient algorithm in pseudocode that computes the smallest subset $Y \subseteq X$ of X such that for each vertex $v \in X$, either (i) $v \in Y$ or (ii) there exists a $u \in Y$ such that v is reachable from u in G (i.e. there is a path from u to v in G).

You may assume the existence of a set data type where: creating a new empty set, adding or removing an element from a set, or checking for containment of an element in the set, are constant time operators.

Given that the graph G has v vertices and e edges, and that there are n vertices in X, your algorithm should run in worst-case $O(v + e)$ time.

Question 4 [19 marks]

This question involves performing an amortised analysis of a data structure.

Consider an array-based data structure that can be used to implement a resizable, $n \times n$-dimensional square matrix. In the data structure, the $n \times n$ dimensional matrix is stored in a $c \times c$-dimensional array, where $c \geq n$. Initially the dimension n of the square matrix is set to 1, and the dimension c of the square array is set to 1 (i.e. initially a 1×1-dimensional square matrix is stored in a 1×1-dimensional array).

The data structure has one operation to increment the dimension n of the resizable square matrix (i.e. to update an $n \times n$-dimensional matrix to be an $(n + 1) \times (n + 1)$-dimensional matrix).

If n is less than c when the increment operation is called, then the operation increments the dimension n by one, but the array used to store the matrix is not modified. However, if $n = c$ when the increment operation is called, then the array must be resized:

- a new larger array of dimension $(2c) \times (2c)$ is allocated,
- all of the elements of the old $c \times c$-dimensional array are copied to the new array, and the reference to the old array is updated to refer to the new array,

before the dimension n is incremented by one.

Let the <u>actual cost</u> of an increment operation be the number of elements that are copied from one array to another during the operation. For example, the first increment operation has an actual cost of 1, since one element must be copied from a 1×1-dimensional array to a 2×2-dimensional array. The second increment operation has an actual cost of 4, since four elements must be copied from a 2×2-dimensional array to a 4×4-dimensional array. The third increment operation has a cost of zero since the array does not need to be resized for that increment operation.

a. [4 marks] For arbitrary $i \geq 1$, write an expression c_i, for the actual cost of the i^{th} increment operation on the data structure.

b. [5 marks] Starting from a freshly initialized data structure with matrix dimension $n = 1$, how many array resizes are there in a sequence of m increment operations (for $m \geq 1$)? What is the actual cost of the the j^{th} array resize operation (for $j \geq 1$)?

c. [10 marks] Using the aggregate method, calculate the worst-case actual cost of performing m (for $m > 0$) consecutive increment operations on the data structure (starting from a freshly initialized data structure with matrix dimension $n = 1$).

Question 2 [10 marks]

The worst-case running time of Dijkstra's algorithm, given below, is dependent on the implementation of the priority queue data structure, Q, and the directed graph, G. There are two common ways to implement a directed graph: using an adjacency-list representation, or using an adjacency-matrix representation.

```
DIJKSTRA($G, w, s$)
1  // $G$ is the graph, $w$ the weight function, $s$ the source vertex
2  INIT-SINGLE-SOURCE($G, s$)
3  $S = \emptyset$
4  $Q = G.V$    // $Q$ is a priority queue maintaining $G.V - S$
5  while $Q \neq \emptyset$
6      $u$ = EXTRACT-MIN($Q$)
7      $S = S \cup \{u\}$
8      for each vertex $v \in G.Adj[u]$
9          RELAX($u, v, w$)
```

```
INIT-SINGLE-SOURCE($G, s$)
1  for each vertex $v \in G.V$
2      $v.d = \infty$
3      $v.\pi$ = NIL
4  $s.d = 0$
```

```
RELAX($u, v, w$)
1  if $v.d > u.d + w(u, v)$
2      $v.d = u.d + w(u, v)$
3      $v.\pi = u$
```

Assuming that the priority queue, Q, is implemented using an unsorted array, does the choice between either (i) the adjacency-list or (ii) the adjacency-matrix graph implementation affect the worst-case asymptotic time complexity of Dijkstra's algorithm? Justify your answer.

As part of your answer you should give, and explain, the worst-case asymptotic time complexity of the algorithm for both graph implementations, given that the priority queue is implemented using an unsorted array. Your time complexities should be expressed in terms of parameters v and e, the number of vertices and edges in the graph G, respectively.

When answering this question, you should assume that the graph G does not contain self-loops (i.e. there cannot be an edge from a vertex v back to itself) or multiple edges (i.e. for any vertices u and v there can be at most one edge from u to v).

Question 5 [25 marks total]

A sequence of integers $X = \langle x_1, x_2, \cdots, x_n \rangle$ is monotonically decreasing when $x_i \geq x_{i+1}$ for all indices i from 1 to $n-1$.

Given a sequence $X = \langle x_1, x_2, \cdots, x_m \rangle$, we have that $Z = \langle z_1, z_2, \cdots, z_k \rangle$ is a subsequence of X if there exists a strictly increasing sequence $\langle i_1, i_2, \cdots, i_k \rangle$ of indices of X such that for all $j = 1, 2, \cdots, k$, we have that $x_{i_j} = z_j$.

Given a sequence $X = \langle x_1, x_2, \cdots, x_m \rangle$ of integers, the problem is to design an efficient algorithm for finding a longest subsequence of X that is monotonically decreasing.

a. [15 marks] This problem can be solved by dynamic programming. Let $A(i, j)$ be the length of the longest subsequence of $\langle x_j, x_{j+1}, \cdots, x_m \rangle$ that (i) is monotonically decreasing and (ii) only contains integers that are less than or equal to x_i, if $1 \leq i \leq m$, or ∞ otherwise. The solution we seek is $A(0, 1)$.

   Give a recurrence defining $A(i, j)$ for $0 \leq i \leq m$ and $1 \leq j \leq m$.

   You do NOT have to give a dynamic programming solution, just the recurrence.

   Be sure to define the base cases of the recurrence as well as the more general cases.

b. [10 marks] For the dynamic programming solution indicate in what order the elements of the matrix A corresponding to the recurrence should be calculated. As part of answering this question you could either give pseudocode indicating the evaluation order or draw a table and indicate the dependencies of typical elements and which elements have no dependencies.

Question 6 [18 marks total]

Below we describe two computer science problems.

The longest-simple-path problem Given an undirected graph G, vertices u and v from $G.V$ (the vertices of the graph), and an integer $k \geq 0$, the longest-simple-path decision problem is to decide if there exists a simple path from u to v in G with at least k vertices. This problem is known to be NP-complete (assuming a standard encoding of inputs).

The most-expensive-simple-path problem Given a weighted, undirected graph G, vertices u and v from $G.V$ (the vertices of the graph), and an integer cost $c \geq 0$, the most-expensive-simple-path decision problem is to decide if there exists a simple path from u to v in G such that the cost of the path (the sum of the weights of the edges in the path) is at least c.

a. [8 marks] Prove that the most-expensive-simple-path problem is NP-hard.

b. [6 marks] Show that the most-expensive-simple-path problem is NP-complete and clearly state any assumptions that you make.

c. [4 marks] A programmer has been asked to write an algorithm that solves a problem. On closer inspection, the programmer recognises that the problem is NP-complete. What implications does this have for the programmer? Explain your answer.