**Question 1** [15 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. Show your working.

a. [5 marks] $T(n) = 6T(\frac{n}{3}) + n^2$

b. [5 marks] $T(n) = 16T(\frac{n}{4}) + n$

c. [5 marks] $T(n) = 4T(n-1) + 1$

**Question 2** [20 marks]

Given an undirected graph $G = (V, E)$, with vertices $G.V$ and edges $G.E$, and a non-empty subset $X \subseteq G.V$ of the vertices in $G$, give an algorithm in pseudocode to set attribute $v.nearest$, for every vertex $v \in G.V$, to be a vertex $x \in X$, such that the *shortest-path weight* from $x$ to $v$ is less than or equal to the *shortest-path weight* from any other vertex $x' \in X$ to $v$. Note that $v$ need not be in $X$.

The *shortest-path weight* from $x$ to $v$ is defined to be the minimum length of any path from $x$ to $v$ in $G$, if there is at least one such path, or $\infty$ otherwise. The length of a path is the number of edges in the path. For any $x \in G.V$ there is always a path, $\langle x \rangle$, of length $0$ from $x$ to $x$.

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 3 [30 marks total]

A dignitary is scheduled to meet with n guests, one at a time in a predetermined order. There are m different rooms where the dignitary can choose to meet with each guest, however, the same room cannot be used for any two consecutive meetings. No other constraints are placed on the way in which rooms can be chosen for each of the meetings. For example, not all rooms need to be used, and the same room can be used on more than one occasion as long as it isn't used in any two consecutive meetings.

Each guest has a satisfaction rating for each of the m different rooms, where the satisfaction of the ith guest for room j is given by s[i][j], for i in 1, 2, $\cdots$, n, and j in 1, 2, $\cdots$, m. Given an assignment of the first n guests to any of the m rooms, the overall satisfaction of the guests is defined to be the sum

$$\sum_{i \in 1,2,\cdots,n} s[i][r_i]$$

where $r_i \in 1, 2, \cdots, m$ is the room assigned to the $i^{th}$ guest.

The problem is to find the maximum overall satisfaction of the guests, given that the same room cannot be used for any two consecutive meetings. This problem can be solved by dynamic programming.

a. [15 marks] Let $M(i, j)$ be maximum overall satisfaction of the first $i$ guests, given that (i) for $k \in 1, 2, \cdots, i-1$, the room used to meet with the $k^{th}$ guest is not the same as the room used to meet with the next, $(k+1)^{th}$, guest, and (ii) if $i \neq 0$ and $j \neq 0$, then the $i^{th}$ guest **cannot** be assigned to room $j$. The solution we seek is $M(n, 0)$, noting that there is no room $0$.

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

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 $M$ 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. What is the time complexity of the dynamic programming solution in terms of $n$ and $m$?

c. [5 marks] What properties would this problem have to satisfy for there to exist a greedy algorithm that solves it? If a greedy algorithm existed to solve this problem, would it be likely to be faster or slower than a dynamic programming solution? Explain your answer.

**Question 4** [20 marks]

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

Consider the following implementation of a min-priority queue that has two operations, ENQUEUE($e$) that adds an element $e$ with priority $e.key$ to the queue, and DEQUEUE that removes and returns an element with the highest priority (the minimum key) from the queue.

The implementation uses two linked lists, called $list1$ and $list2$, which are initially empty. The elements of $list2$ are kept in sorted order (from minimum key to maximum key), but the elements in $list1$ are not.

The linked lists have operators SIZE, that returns the number of elements in the list; REMOVEFIRST(), that removes and returns the first element in the linked list; and ITERATOR(), that returns an iterator over the linked list. An iterator over a linked list can be used to traverse the list in the forward direction from the front of the list to the end of the list. It has operators HASNEXT(), that returns true if this list iterator has more elements when traversing the list in the forward direction; NEXT() that returns the next element in the list and advances the cursor position to the next element; PREVIOUS() that moves the cursor position of the iterator backwards one place; ADD($e$) that inserts element $e$ into the list immediately before the element that would be returned by NEXT(), if there is one, or at the end of the list otherwise.

```
ENQUEUE(e)
1  list1.APPEND(e)

DEQUEUE()
 1  while list1.SIZE ≠ 0
 2      // remove the first element of list1 and insert it into list2 in sorted order
 3      e = list1.REMOVEFIRST()
 4      // find the place for e in list2
 5      searching = true
 6      list2Iterator = list2.ITERATOR()
 7      while searching AND list2Iterator.HASNEXT()
 8          if (list2Iterator.NEXT().key > e.key)
 9              list2Iterator.PREVIOUS()
10              searching = false
11      list2Iterator.ADD(e)
12  // list2 now contains all of the elements in the priority queue in sorted order
13  return list2.REMOVEFIRST()
```

Let $S1$ be the size of $list1$ and $S2$ be the size of $list2$. Then the size of the priority queue, $Q$, at any time is $S1 + S2$.

In the following analysis, measure the complexity of the operations in terms of the number of method calls to the iterator procedure HASNEXT() only. Using this complexity measure, the ENQUEUE operation has an *actual cost* of $0$.

a. [5 marks] Given that $S1$ is the size of $list1$ and $S2$ is the size of $list2$, what is the worst-case *actual cost* of the DEQUEUE operation (in terms of the number of method calls to HASNEXT)? Answer in terms of the sizes of lists: $S1$ and $S2$. Your answer should be a precise count in the form of a summation, not an order of magnitude.

b. [15 marks] Give a *potential function* $\Phi$ for the queue, as would be used in analysing the complexity (measured in terms of the number of calls to HASNEXT) with the *potential method*, and show that the value of the potential function after any sequence of $m$ operations, starting from an empty queue, is bound below by the initial value of the potential function. Calculate the amortised cost of operations ENQUEUE and DEQUEUE from $\Phi$ (show your working).

**Question 5** [15 marks total]

Below we describe two computer science problems.

**The hamiltonian cycle problem** The hamiltonian-cycle decision problem is to decide if a given undirected graph, $G = (V, E)$, contains a simple cycle that contains every vertex in $G.V$. This problem is known to be NP-complete (assuming a standard encoding of inputs).

**The longest-simple-cycle problem** Given an undirected graph $G = (V, E)$, and vertex $v$ from $G.V$ (the vertices of the graph), and an integer $k \geq 0$, the longest-simple-cycle decision problem is to decide if there exists a simple cycle in $G$ that contains $v$ and has at least $k$ edges.

a. [5 marks] Prove that the longest-simple-cycle problem is NP-hard.

b. [5 marks] Show that the longest-simple-cycle problem is NP-complete and clearly state any assumptions that you make.

c. [5 marks] Your friend has a proof that there exists no polynomial-time algorithm to solve a particular NP-complete problem. What is the significance of that result, if any? Explain your answer.