**Question 1** [10 marks total]

Solve the following recurrence equations to get a tight asymptotic bound on the complexity of the corresponding functions. Justify your answers by using the master theorem. Show your working. You may assume that $T(n) = \Theta(1)$ for suitable constant $n$. (You may leave logarithmic expressions in your answers if they evaluate to nonintegral values.)

a. [5 marks] $T(n) = 8T(\frac{n}{2}) + n(n+1)$

b. [5 marks] $T(n) = 3T(\frac{n}{4}) + n \log_2 n$

**Question 2** [10 marks] Given the recurrence

$$\begin{array}{rcll} T(n) & = & 1 & \text{for } 1 \leq n < 8 \\ T(n) & = & T(\lfloor n/4 \rfloor) + T(\lfloor 3n/4 \rfloor) + n & \text{for } n \geq 8 \end{array}$$

prove that T (n) ∈ O(n log2 n) using the substitution method. Clearly describe your inductive assumptions and boundary conditions, and show your working. Justify that what you have proven is sufficient to show that T (n) ∈ O(n log2 n) by referring to the definition of O-notation.

**Question 3** [20 marks]

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

Consider the following implementation of a collection of integers that has an operation INSERT($x$) that inserts integer $x$ into the collection.

The implementation uses an array $A$ of size $n$, and an integer $size$ where $0 \leq size \leq n$. The array $A$ is indexed from zero (i.e. $A[0]$ is the first element of the array). The current elements of the collection are stored in the first $size$ positions of the array $A$. For the purpose of this question you may assume that $n$ is a power of two and that $n \geq 2$. Initially, a new collection of integers has no elements (i.e. $size == 0$).

```
INSERT($x$)

1  if $size == n$
2        REARRANGE($A$) // rearranges the elements of array $A$
3        $size = n/2$
4  $A[size] = x$
5  $size = size + 1$
```

If INSERT($x$) is called when $size < n$ then element $x$ is inserted into the collection. Otherwise, if INSERT($x$) is called when the array is already full (i.e. $size == n$), then $n/2$ elements of the array are first removed, and then $x$ is inserted into the collection. The removal is performed by first using operation REARRANGE to rearrange the elements of $A$ so that the elements to be removed are in the last half of the array $A$, and then updating $size$ to be $n/2$.

a. [10 marks] Exactly how many times will line 2 of INSERT be called in any sequence of $m$ consecutive INSERT operations, starting from an empty collection (i.e. one in which $size == 0$). Answer in terms of $m$ (the number of operations) and $n$ (the size of $A$).

b. [8 marks] Given that the procedure REARRANGE on line 2 of INSERT has a worst-case time complexity of $\Theta(n)$, use the *aggregate method* to determine an upper-bound on the worst-case time complexity of any sequence of $m$ consecutive INSERT operations, starting from an empty collection (i.e. one in which $size = 0$). Answer in terms of $m$ (the number of operations) and $n$ (the size of $A$). Make your bound as tight as possible, and show your working.

c. [2 marks] Based on your answer to part (b) of this question, what is the *amortised cost* per INSERT operation, over a sequence of $m$ consecutive INSERT operations, starting from an empty collection?

**Question 4** [25 marks]

The Bellman-Ford algorithm (with pseudo-code given below) solves the single-source shortest-paths problem on a weighted, directed graph $G$ with vertices $G.V$ and edges $G.E$ and weights $w$, as long as there are no negative weight cycles. The algorithm returns TRUE if and only if the graph contains no negative weight cycles that are reachable from the source vertex $s$. In that case the final shortest-path weight from the source $s$ to a vertex $v$ is stored in $v.d$ at the termination of the algorithm.

```
BELLMAN-FORD($G, w, s$)
 1  // $G$ is the graph, $w$ the weight function, $s$ the source vertex
 2  INIT-SINGLE-SOURCE($G, s$)
 3  // Relax each edge $|G.V|-1$ times to find shortest paths
 4  for $i = 1$ to $|G.V| - 1$
 5      for each edge $(u, v) \in G.E$
 6          RELAX($u, v, w$)
 7  // Check for negative-weight cycles
 8  for each edge $(u, v) \in G.E$
 9      if $v.d > u.d + w(u, v)$
10          return FALSE
11  return TRUE
```

```
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      $v.\pi = u$
```

Consider the weighted directed graph $G$ with vertices $G.V = \{a, b, c, d, e\}$ and weight function

$$w \quad = \quad \{(d, b) \mapsto -4, (d, e) \mapsto 1, (c, e) \mapsto 1, (c, d) \mapsto 1, (b, c) \mapsto 2, (a, b) \mapsto 1\},$$

which maps the directed edges in $G$, $G.E = \mathsf{domain}(w)$, to values.

a. [6 marks] Show how the Bellman-Ford algorithm runs on graph $G$ with weight-function $w$, using vertex $a$ as the source. You must assume that in the loop on line 5 of BELLMAN-FORD the edges are always visited in the following order (from left to right):

$$(d, b), (d, e), (c, e), (c, d), (b, c), (a, b)$$

Give your answer by showing the values of $v.d$ and $v.\pi$ for each vertex $v \in G.V$ after each iteration of the first for loop (line 4) in the BELLMAN-FORD algorithm. Also state whether or not the return value of the algorithm will be TRUE or FALSE.

b. [4 marks] For which vertices $v$ from $G$ (if any) does there exist a negative-weight cycle on some path from $a$ to $v$?

c. [15 marks] Modify the Bellman-Ford algorithm, so that (instead of TRUE/FALSE) the return value is the set of all vertices $v$ in $G.v$ such that there is a negative-weight cycle on some path from the source vertex $s$ to $v$. (If the return value is the empty set, this will mean that the graph contains no negative weight cycles.)

The modified algorithm should have the same asymptotic time complexity as the current algorithm.

Give clear pseudocode (including all algorithmic details) for the modified parts of the algorithm.

**Question 5** [25 marks total]

There are $n$ (where $n > 0$) players, $p_1, p_2, \cdots, p_n$, standing in a row, where the value of the $i^{th}$ player in the row, $p_i$, is $v[i]$ for $i$ in $1, 2, \cdots, n$. Each of the players needs to be allocated to one of two possible teams. There is a process for allocating the players to the teams.

The coaches of the two teams take turns selecting a player for their team. In each turn, the coach taking the turn must select (and allocate to their team) either the first player in the row who has not yet been allocated to a team, or the last player in the row who has not yet been allocated to a team. For example, the first coach to take a turn may select either $p_1$ or $p_n$. The allocation process finishes when there are no unallocated players left.

The *total value* of the players allocated to a team is the sum of the individual values of the players allocated to the team. More generally, for any $1 \leq i \leq j \leq n$, the *total value* of the players from $p_i, p_{i+1}, \cdots p_j$ allocated to a team is the sum of the individual values of the players from $p_i, p_{i+1}, \cdots p_j$ who are allocated to the team.

Note that if the total value of the players from $p_i, p_{i+1}, \cdots p_j$ allocated to one team is $X$, then the total value of the players from $p_i, p_{i+1}, \cdots p_j$ allocated to the other team must be $(\sum_{k=i}^{j} v[k]) - X$.

The goal of each coach is to maximise the total value of the players allocated to their own team (and hence minimise the total value of the players allocated to the other team).

The coach who takes the first turn wants to know the maximum total value of the players that they can allocate to their team, given that each coach always makes a selection that will result in the maximum total value of the players allocated to their own team.

This problem can be solved by dynamic programming.

a. [15 marks] Let $M(i, j)$ be the maximum total value of the players from $p_i, p_{i+1}, \cdots, p_j$ that could be allocated by a coach to their team given that (i) *it is currently their turn*, and (ii) the players in that range (i.e. $p_i, p_{i+1}, \cdots, p_j$) are the only ones (from the $n$ players standing in row) who have not yet been allocated, and (iii) the other coach always makes a selection that will result in the maximum total value of the players allocated to their own team. The solution we seek is $M(1, n)$.

   Give a recurrence defining $M(i, j)$ for $1 \leq i \leq j \leq n$.

   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, give pseudocode indicating the evaluation order.

**Question 6** [10 marks total]

Below we describe two computer science problems.

**The clique problem**

A *clique* $C$ of an undirected graph $G = (V, E)$ is a subset of the vertices of $G$ such that for each pair of vertices $u \in C$ and $v \in C$, if $u \neq v$ then $u$ and $v$ are directly connected by an edge in $G.E$. The size of a clique $C$ is the number of vertices in the set $C$.

Given an undirected graph $G = (V, E)$ and an integer $k \geq 1$, the clique decision problem is to decide if $G$ contains a clique of at least size $k$.

This problem is known to be NP-complete (assuming a standard encoding of inputs).

**The weighted-clique problem**

Given a clique $C$ from an undirected graph $G = (V, E)$, and an integer-valued weight function $w$ that maps each edge in $G.E$ to a corresponding integer-valued weight, the *weight of clique* $C$ is the sum of the weights of the edges in $G$ that connect vertices in $C$.

Given an undirected weighted graph $G = (V, E)$ with integer-valued weight function $w$, and an integer $c \geq 0$, the weighted-clique decision problem is to decide if $G$ contains a clique of at least weight $c$.

a. [5 marks] Prove that the weighted-clique problem is NP-hard.

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