Student Number:

The University of Melbourne
COMP20007: Design of Algorithms
Semester 1 Sample Exam Solutions 2023

Reading Time	15 minutes.
Writing Time	Three hours.

This paper has 24 pages including this cover page.

Authorised Materials: None.

Instructions to Invigilators:
- Students will write all of their answers on this examination paper.
- Students may not remove any part of the examination paper from the examination room.

Instructions to Students:
- This paper counts for 60% of your final grade.
- All questions must be answered in the indicated answer boxes provided on this examination paper.
- Answer each of the following questions by writing a brief response or explanation (no essays please!).
- Only material written inside the boxes will be marked.
- If you need to make rough notes, or prepare draft answers, you may do so on the reverse of any page.

Paper to be held by Baillieu Library: No.

Common Facts and Notation

We know from the lectures that for $0 < \varepsilon < 1 < c$: $1 \prec \log n \prec n^{\varepsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n}$.

Remember that $\log n$ is an abbreviation for $\log_2 n$, the binary logarithm.

Remember also that for $a > 0$: $\sum\limits_{i=0}^{n} (a^i) = \dfrac{1 - a^{n+1}}{1 - a}$.

We remind you also of the following conventions:

$$\sum_{i=m}^{n} (s_i) = s_m + s_{m+1} + \ldots + s_{n-1} + s_n \quad \text{and} \quad \prod_{i=m}^{n} (p_i) = p_m \cdot p_{m+1} \cdot \ldots \cdot p_{n-1} \cdot p_n.$$

Logarithm laws

$$\log a + \log b = \log ab, \quad \log a - \log b = \log a/b \quad \text{and} \quad \log a^n = n \log a.$$
$$a^{\log_b c} = c^{log_b a} \quad a > 0,\ b > 0,\ c > 0.$$

Master Theorem

For integer constants $a \geq 1$ and $b > 1$, and function f with $f(n) \in \Theta(n^d), d \geq 0$, the recurrence

$$T(n) = aT(n/b) + f(n)$$

(with $T(1) = c$) has solutions, and

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Note that we also allow a to be greater than b.

Question 1 [6 marks]

(a) For the following pairs of functions, $f(n), g(n)$, determine if $f(n) \in \Theta(g(n))$, $f(n) \in O(g(n))$, or $f(n) \in \Omega(g(n))$, making the strongest statement possible. That is, if both $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ are true, you **must** answer with the form $f(n) \in \Theta(g(n))$. You must answer with $f(n)$ on the left and $g(n)$ on the right, for example, you may not answer $g(n) \in O(f(n))$. You need to justify your answer.

i. $f(n) = \log(n)$ and $g(n) = \log(\log(n))$.

ii. $f(n) = \log_3(n)$ and $g(n) = \log_2(n^2)$.

iii. $f(n) = 100\sqrt{n} + 0.01n^2$ and $g(n) = n$.

iv. $f(n) = \log n + n^2$ and $g(n) = n$.

(b) For the following recurrence relations, use the Master Theorem to make the strongest statement possible about the complexities of the closed form solutions.

i. $T(n) = 5T(\frac{3n}{7}) + n, T(1) = 5165$.

ii. $T(n) = T(\frac{n}{10}) + \sum_{i=0}^{8} n^i, T(1) = 30$.

Question 1 Solutions

(a) i. Using L'Hôpital's rule,

$$\lim_{n\to\infty} \frac{\log(n)}{\log(\log(n))} = \lim_{n\to\infty} \frac{\frac{d}{dn}\log(n)}{\frac{d}{dn}\log(\log(n))} = \lim_{n\to\infty} \frac{n^{-1}}{\frac{n^{-1}}{\log(n)}} = \lim_{n\to\infty} \log(n) = \infty$$

So $f(n) \in \Omega(g(n))$. [1 mark]

ii. $g(n) = \log_2(n^2) = 2\log_2(n) = \frac{2}{\log_3(2)}\log_3(n)$ so $f(n) \in \Theta(g(n))$. [1 mark]

iii. $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ so $f(n) \in \Omega(g(n))$ [1 mark]

iv. $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ so $f(n) \in \Omega(g(n))$ [1 mark]

(b) i. Using Master Theorem, $a = 5, b = \frac{7}{3}, d = 1$, $b^d = (\frac{7}{3})^1$, so $a > b^d$ and hence the strongest statement about the bound of the closed form solution is:

$$\Theta(n^{\log_b(a)}) = \Theta(n^{\log_{\frac{7}{3}}(5)}) \approx \Theta(n^{1.899})$$

[1 mark]

ii. Using Master Theorem, $a = 1, b = 10, d = 8$ as n^8 is the largest order term in the finite sum, $b^d = 10^8$, so $a < b^d$ and hence the strongest statement about the bound of the closed form solution is:

$$\Theta(n^d) = \Theta(n^8)$$

[1 mark]

Question 2 **[4 marks]**

(a) Consider the following sorting algorithm:

```
function ChunkSort($A[0 \dots n-1], n$)
    if $n == 2$ and $A[0] > A[1]$ then
        swap $A[0]$ and $A[1]$
    else
        ChunkSort($A[0 \dots \frac{n}{2} - 1]$, $\frac{n}{2}$)
        ChunkSort($A[\frac{n}{2} \dots n-1]$, $\frac{n}{2}$)
        ChunkSort($A[\frac{n}{4} \dots \frac{3n}{4} - 1]$, $\frac{n}{2}$)
        ChunkSort($A[0 \dots \frac{n}{2} - 1]$, $\frac{n}{2}$)
        ChunkSort($A[\frac{n}{2} \dots n-1]$, $\frac{n}{2}$)
        ChunkSort($A[\frac{n}{4} \dots \frac{3n}{4} - 1]$, $\frac{n}{2}$)
```

Write the recurrence relation (including the base case) for the number of comparisons performed by ChunkSort.

(b) Find the *closed form* solution to the recurrence relation for the number of comparisons performed by ChunkSort given an initial input of size m.

Question 2 Solutions

(a) The recurrence relation is

$$T(n) = 6T(\frac{n}{2}) + 1, T(2) = 1$$

[1 mark]

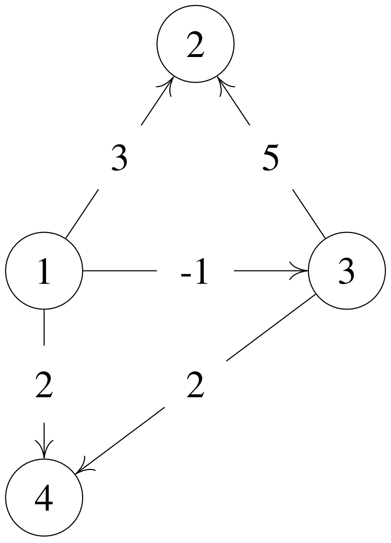
(b)

$$\begin{aligned}
T(n) &= 6T(\frac{n}{2}) + 1 \\
&= 6\big(6T(\frac{n}{2^2}) + 1\big) + 1 \\
&= 6\left(6\big(T(\frac{n}{2^3}) + 1\big) + 1\right) + 1 \\
&= 6^3T(\frac{n}{2^3}) + 6^2 \times 1 + 6^1 \times 1 + 1 \\
&\vdots \quad k \text{ times} \\
&= 6^kT(\frac{n}{2^k}) + \sum_{i=0}^{k-1} 6^i \\
&\vdots \quad \frac{n}{2^k} = 2 \implies k = \log_2 n - 1 \\
&= 6^{\log_2 n - 1}T(\frac{n}{2^{\log_2 n - 1}}) + \sum_{i=0}^{\log_2 n - 1 - 1} 6^i \\
&= 6^{\log_2 n - 1}T(2) + \sum_{i=0}^{\log_2 n - 2} 6^i \\
&= 6^{\log_2 n - 1} + \frac{1 - 6^{\log_2 n - 1}}{1 - 6} \\
&= 6^{\log_2 n - 1} + \frac{6^{\log_2 n - 1} - 1}{5} \\
&= \frac{1}{6} \times 6^{\log_2 n} + \frac{\frac{1}{6} \times 6^{\log_2 n} - 1}{5} \\
&= \frac{n}{6} \times 3^{\log_2 n} + \frac{\frac{n}{6} \times 3^{\log_2 n} - 1}{5} \\
&= \frac{n}{6} \times n^{\log_2 3} + \frac{\frac{n}{6} \times n^{\log_2 3} - 1}{5} \\
&= \frac{n \times n^{\log_2 3} - 1}{5}
\end{aligned}$$

[3 marks]

Question 3 **[8 marks]**

(a) Give the weights matrix for the following graph.



(b) Run Floyd's algorithm to find the shortest paths between each pair of vertices in the graph from part (a). Write the matrices after each step of the algorithm.

(c) Describe what the time complexity of Floyd's algorithm is (assume the graph has n vertices and m edges).

(d) Using $\text{DIJKSTRA}(G, u, v)$ as a helper function which returns the cost of the shortest path between u and v in a graph G, write an algorithm in pseudocode for computing the *all pairs shortest paths* in a graph G with non-negative edge weights. The graph G has n vertices and m edges and is sparse, *i.e.,* $m \in O(n)$.

Your algorithm must have a time complexity better (*i.e.,* smaller) than Floyd's algorithm.

Question 3 Solutions

(a) The weights matrix is given by

$$\begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

[1 mark]

(b) Running Floyd's algorithm yields the following matrices. The row and column in question will appear blue, while any elements considered for updating will appear in red.

$$W^0 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^1 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} 0 & 3 & -1 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^4 = \begin{bmatrix} 0 & 3 & -1 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

[3 marks]

(c) Floyd's algorithm is a dynamic programming algorithm which solves n subproblems (the intermediate weights matrices), each of which requiring up to $\Theta(n^2)$ updates. So Floyd's algorithm is $\Theta(n^3)$.

[1 mark]

(d) The following algorithm will run in $O(n^2 \log n)$ time given the graph is sparse, because in a sparse graph, Dijkstra's algorithm takes $O((n+m)\log n) = O((n+n)\log n) = O(n \log n)$ time, and we run this as a helper function n times.

```
function MyASPS($G = (V, E)$)
    paths ← $n \times n$ matrix
    for $u$ in $V$ do
        uPaths ← Dijkstra(G, u)
        for $v$ in $V$ do
            paths$[u][v]$ ← uPaths$[v]$
    return paths
```

[3 marks]

Question 4 **[6 marks]**

(a) A string of parentheses (”(” and ”)”) is *valid* if no pair of parentheses is ”closed” before it is ”opened”. For example, ”(())()” is valid while ”())(” is not.

The 5 valid strings of 6 parentheses are:

$$\text{"()()()", "(())()", "()(())", "(()())" and "((()))".}$$

Give 3 (three) examples of valid strings of parentheses containing 8 (eight) parentheses.

(b) Write a recursive formula and base case(s) for the subproblem N_i, where N_i is defined as the number of different valid strings containing exactly i parentheses.

(c) What is the space complexity of your algorithm? Give an answer in Big-Theta notation and justify your answer.

Question 4 Solutions

(a) Some examples are: ”()()()()”, ”(((())))”, ”(())()()”. [1 mark]

(b) We can determine the number of valid strings there are by considering the following scenario: place one set of parentheses and then determine how many ways we can fill the gaps.

For example, for n parentheses, we’ve already placed 2 parentheses, so we will put some even number within these (let’s say k) and the remaining parentheses after the pair we’ve already placed:

$$(\underbrace{\qquad\qquad}_{N_k \text{ ways}})\underbrace{\qquad\qquad}_{N_{n-2-k} \text{ ways}}$$

The recursive formula for N_n is then given by,

$$N_0 = 1$$
$$N_n = \begin{cases} 0, & \text{if } n \text{ odd} \\ \sum_{i\in\{0,2,4,\ldots,n-2\}} N_i \times N_{n-2-i}, & \text{if } n \text{ even} \end{cases}$$

[4 marks]

(c) The algorithm requires an additional array with $n+1$ elements, so the space complexity is $\Theta(n)$.

[1 mark]

Question 5 [12 marks]

(a) Consider a hash table with 8 slots and a hash function $h(k) = k \mod 8$, which uses open addressing with a step size of 1. Show the contents of the table after inserting 16, 23, 7, 10 and 12.

(b) How many comparisons are required to lookup the key 7? Indicate which comparisons are performed. You can assume that we can check if a slot is empty without making any comparisons.

(c) How many comparisons are required to lookup the key 15? Indicate which comparisons are performed. You can assume that we can check if a slot is empty without making any comparisons.

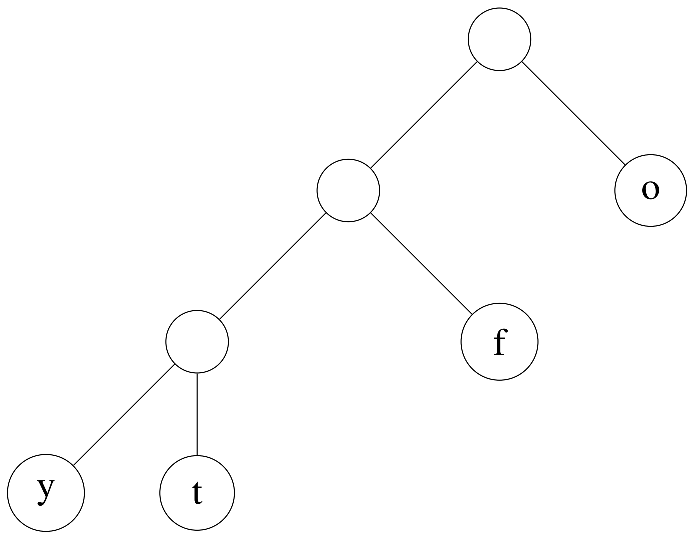
(d) Consider again a hash table with 8 slots and a hash function $h_1(k) = k \mod 8$. This time, the hash table uses separate chaining. Show the state of the hash table after inserting 9, 17, 4, 11 and 1 into an empty hash table.

(e) Now consider the same hash table (with separate chaining) but with a different hash function $h_1(k) = k \mod 5$. Explain why this hash function is not ideal for this hash table.

(f) Use Huffman's algorithm to construct a Huffman code tree for the string "free-coffee".

(g) What is the encoded version of this string, using your Huffman tree? Let each left child be 0 and each right child be 1. Give the total length of the encoding in bits.

(h) Using the following Huffman tree (again, using 0 for left and 1 for right), give the codes for f, o, t, and y and decode 0111001000.



Question 5 Solutions

(a)

0	16
1	7
2	10
3	
4	12
5	
6	
7	23

[2 marks]

(b) Three comparisons are required, at indices 7, 0, and 1. [1 mark]

(c) 15 hashes to 7. So indices 7, 0, 1, and 2 are checked (unsuccessfully). Since 3 is empty this process finishes without finding 15 after **4 comparisons**.

[1 mark]

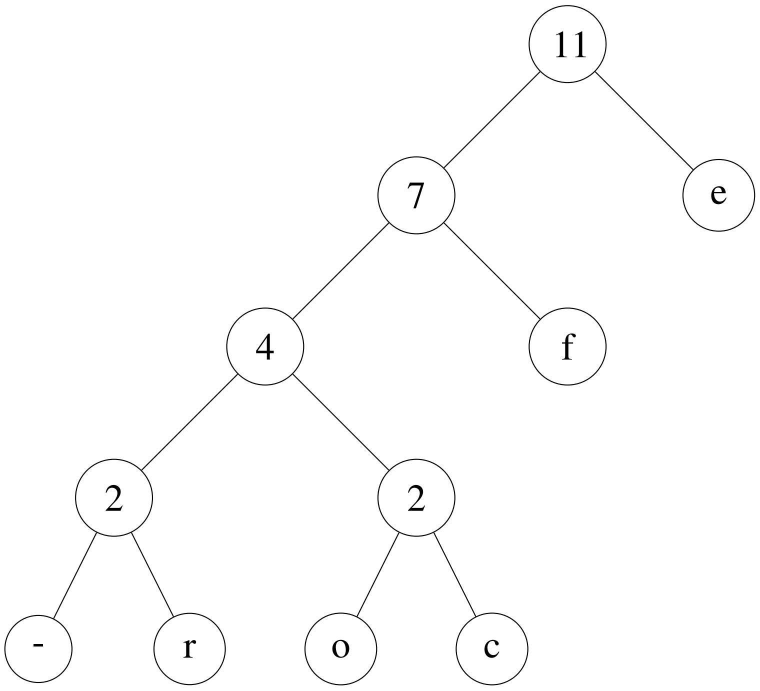
(d)

0	
1	$9 \to 17 \to 1$
2	
3	11
4	4
5	
6	
7	

[1 mark]

(e) Because this would result in positions 5, 6 and 7 never being filled. In general the argument of the mod function should be the same as the size of the hash table. [1 mark]

(f) The frequencies are $\{e:4, f:3, c:1, o:1, r:1, -:1\}$



[2 marks]

(g) Let left be 0 and right be 1. The encoded version of *"free-coffee"* is then

01 0001 1 1 0000 0011 0010 01 01 1 1

[2 marks]

(h) The decoded version of 0111001000 is footy.

f	01
o	1
t	001
y	000

[2 marks]

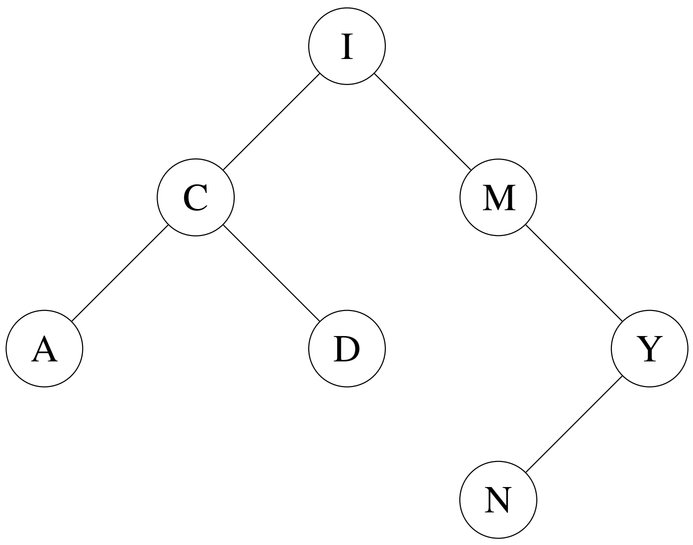
Question 6 [9 marks]

(a) Perform mergesort on the array [25, 23, 17, 29, 7, 16, 33, 21], showing the result after each recursive call. Indicate which range of elements are being considered during each recursive call by outlining those elements.

(b) Which two operations must a queue implement? Give the name and a brief description of these two operations.

(c) Insert the characters $C, O, M, P, L, E, X, I, T, Y$ into an initially empty 2-3 tree. You must show the state of the 2-3 tree after every insertion.

(d) Describe which two rotations must be done to balance the following tree. Give the node and the direction of the rotation (left or right).



(e) Describe how performing a right rotation of the tree above would change the in-order traversal of the tree. You should give the resultant in-order traversal as part of your answer.

Question 6 Solutions

(a) Perform mergesort on the array [25, 23, 17, 29, 7, 16, 33, 21], showing the result after each recursive call. Indicate which range of elements are being considered during each recursive call by outlining those elements.

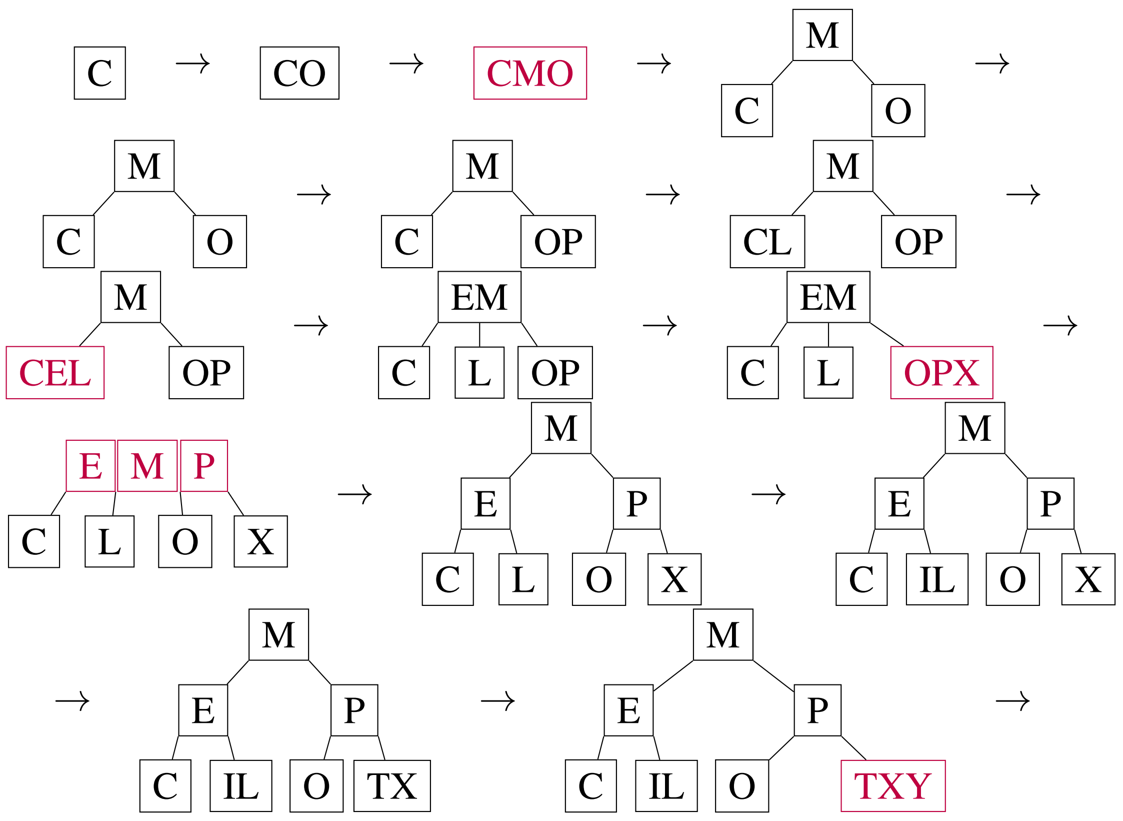
$$[25, 23, 17, 29, 7, 16, 33, 21]$$
$$[25, 23, 17, 29] \quad [7, 16, 33, 21]$$
$$[25, 23] \quad [17, 29] \quad [7, 16] \quad [33, 21]$$
$$[23, 25] \quad [17, 29] \quad [7, 16] \quad [21, 33]$$
$$[17, 23, 25, 29] \quad [7, 16, 21, 33]$$
$$[7, 16, 17, 21, 23, 25, 29, 33]$$

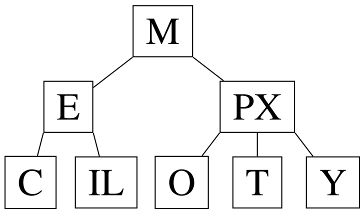
[2 marks]

(b) **Enqueue** inserts an element to the "back" of a queue. **Dequeue** removes and returns an element from the "front" (*i.e.*, the element which was inserted into the queue the earliest).

[2 marks]

(c) The intermediate states of the 2-3 tree (when nodes are over-full and before the problem is fixed) are shown in purple





[2 marks]

(d) The two rotations that must be performed are **rotate Y right** (so that N becomes the parent of Y and the right child of M) and then **rotate M left** (so that I's left child becomes N, with left and right child M and Y, respectively).

[1 mark]

(e) Performing a rotation on a binary tree does not change the in-order traversal. So the in-order traversal must be A, C, D, I, M, N, Y.

[2 marks]

Question 7 [7 marks]

A scout wants to raise funds by selling cookies in the neighbourhood's main street. The scout plans to visit houses in order and knows in advance how many cookies each house will buy. However, there is one caveat: **the scout cannot sell cookies in two consecutive houses**. The goal is to maximise the number of cookies sold in total. For example, if there are 6 houses in the street and the profit for each house is, in order:

$$[10, 20, 5, 15, 40, 10]$$

then, the maximum profit is obtained by selling cookies to houses 2 and 5 ($20 + 40 = 60$).

(a) You want to develop a Dynamic Programming (DP) solution for this problem. Define the base case for this problem.

(b) Define the recurrence relation for the DP solution.

(c) Develop the full DP algorithm and write it in pseudocode format. The algorithm should receive an array with the house profits and return the maximised total profit (in the example above, it would be the total from selling to houses 2 and 5 - 60).

Question 7 Solutions

(a)

(b) The two parts go together, though the choice of base case affects the recurrence. The two subproblems we are looking at to solve for the n^{th} house is either selling to house n or not. Not selling to house n allows us to take the maximum of selling to the $n-1^{th}$ house and not.

$$H_{1,\text{sell}} = H[1] = 10$$
$$H_{1,\neg\text{sell}} = 0$$

The recurrences for these base cases are:

$$H_{(n,\text{sell})} = H_{(n-1,\neg\text{sell})} + H[n]$$
$$H_{(n,\neg\text{sell})} = \max(H_{(n-1,\text{sell})}, H_{(n-1,\neg\text{sell})})$$

[3 marks]

(c)
```
function FindProfit($H[1,\ldots,n], n$)
    lastSell ← H[1]
    lastNSell ← 0
    for $i$ ← 2 to $n$ do
        newSell ← lastNSell + H[$i$]
        newNSell ← lastSell
        if lastNSell > newNSell then
            newNSell ← lastNSell
    if lastSell > lastNSell then
        return lastSell
    else
        return lastNSell
```

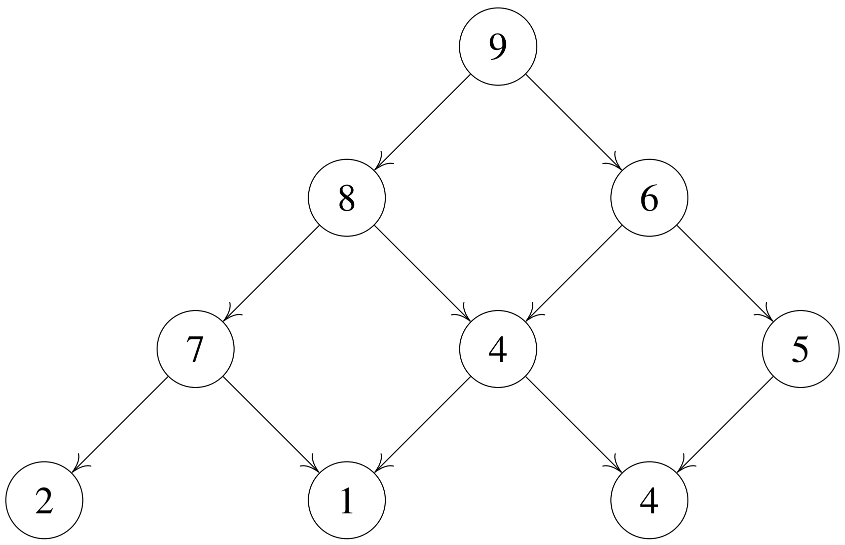
[4 marks]

Question 8 **[8 marks]**

A *web* is a data structure similar to a heap that can be used to implement a priority queue. As with a heap, a web is defined by two properties:

- **Structure property:** A web W consists of l levels, $l > 0$. The ith level, for $0 \leq i < l$, contains at most $i+1$ nodes, indicated as $W_{i,j}$ for $0 \leq j \leq i$. All levels but the last are completely filled, and the last level is left-justified.
- **Ordering property:** Any node $W_{i,j}$ has at most two *children*: $W_{i+1,j}$ and $W_{i+1,j+1}$, if those nodes exists. The priority of a node is always greater than or equal to the priority of either child node.

For example, the following diagram shows a web with 9 nodes and 4 levels.



A web W with n nodes can be stored in an array A of size n, similarly to an array-based heap. So, for example, if $n \geq 1$, the root node of the web $W_{0,0}$ will be stored at $A[0]$. This means we can access nodes in a web using a mathematical formula, in the same we use for heaps. Assume n, i and j are such that all indicated web nodes actually exist.

(a) For a web W with l levels ($l \geq 1$), give the upper and lower bounds for the number of nodes n. For example, a web with 3 levels has at least 4 and at most 6 nodes. Show your work in detail.

(b) Give the formula to access node $W_{i,j}$. Justify your answer. You can use the result from **(a)** to help justifying.

(c) Briefly describe how to eject the maximal element (the root node) from a web W. Find the complexity of your approach and justify your conclusion.

Question 8 Solutions

(a) Maximum: $\sum_{i=1}^{l} i = \frac{1}{2}l(l+1)$, Minimum: $1 + \sum_{i=1}^{l-1} i = 1 + \frac{1}{2}(l-1)l$

[3 marks]

(b) The index of the node will be the number of nodes on all levels before it plus the progress on the current level, since the i component is zero-indexed, this corresponds to the number of levels above the node of interest. Hence we will find $W_{i,j}$ in index $\frac{1}{2}i(i+1)+j$.

[2 marks]

(c) We can eject the root node trivially as we assume we have $O(1)$ access to that node. We can fill this hole by replacing it with the final index in the web, this then follows a similar process to SIFTDOWN, finding the larger of the two children and swapping with that child. This repeats until the value satisfies the ordering property for any children it has.

The worst case performance is that we must travel to the bottom layer of the web. For a web with n elements, we know from part (a) that $1 + \frac{1}{2}(l-1)l \leq n \leq \frac{1}{2}l(l+1)$, so $l \in \Theta(\sqrt{n})$. Since this repeated SIFTDOWN occurs between 1 and l times, the complexity is $O(\sqrt{n})$.

[3 marks]

The box here are for emergency use only. If you do need to use extra space for any answers, indicate CLEARLY in your previous answer that you have continued onto this page. Without such an indication, it is possible that this part of your answer will be overlooked.

END OF EXAM