CS F364: Design & Analysis of Algorithms
Mid-Semester Examination (Solutions)

BITS Pilani, Hyderabad Campus

June 20, 2026

Time: 90 minutes — Max Marks: 90

Instructions: Answer all six questions. Each question carries marks as indicated. Show all steps for full credit. Unsupported answers receive no credit. Closed book. No electronic devices.

1. (15 points) **Asymptotic Notation**

   (a) (5 points) Arrange the following functions in **increasing order** of growth rate (slowest-growing first):

   $$f_1(n) = n^2, \quad f_2(n) = n!, \quad f_3(n) = n, \quad f_4(n) = \sqrt{n}, \quad f_5(n) = 2^n, \quad f_6(n) = n \log n$$

   Solution: The correct increasing order is:

   $$f_4(n) \prec f_3(n) \prec f_6(n) \prec f_1(n) \prec f_5(n) \prec f_2(n)$$

   Which is:

   $$\sqrt{n} \prec n \prec n \log n \prec n^2 \prec 2^n \prec n!$$

   (b) (5 points) For each function below, select the **tightest** Big-Oh bound $O(f(n))$ from the list:

   $$\{O(1), O(\log n), O(n), O(n \log n), O(n^{1.5}), O(n^2), O(2^n), O(n^{105})\}$$

   i. $500n + 5n^{1.5} + 50n \log n$
   ii. $n^2 \log n + n(\log n)^2$
   iii. $3 \log_8 n + \log_2 \log_2 \log_2 n$
   iv. $2^n + n^{105} + 0.5n^{1.25}$

   Solution:

   i. $500n + 5n^{1.5} + 50n \log n = O(n^{1.5})$

   ii. $n^2 \log n + n(\log n)^2 = O(n^{105})$

   iii. $3 \log_8 n + \log_2 \log_2 \log_2 n = O(\log n)$

   iv. $2^n + n^{105} + 0.5n^{1.25} = O(2^n)$

(c) (5 points) Prove the following statement **from the definition** of Big-Oh:

$$2^{n+5} = O(2^n)$$

Solution: By definition, $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Here, $f(n) = 2^{n+5}$ and $g(n) = 2^n$. We can rewrite $f(n)$ as:

$$2^{n+5} = 2^5 \cdot 2^n = 32 \cdot 2^n$$

Let $c = 32$ and $n_0 = 1$. Then:

$$2^{n+5} \leq 32 \cdot 2^n \quad \text{for all } n \geq 1$$

Therefore, $2^{n+5} = O(2^n)$.

2. (20 points) **Recurrences — Master Theorem**
Prove the Master Theorem using the recursion tree method by the following steps:

(a) (5 points) State the three cases of the Master Theorem for recurrences of the form $T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$. For each case, specify the relationship between $f(n)$ and $n^{\log_b a}$ and give the resulting asymptotic bound for $T(n)$.

Solution:

- **Case 1:** If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- **Case 2:** If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
- **Case 3:** If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n (regularity condition), then $T(n) = \Theta(f(n))$.

(b) (5 points) Analyze the recursion tree for $T(n) = aT(n/b) + f(n)$. Determine: (i) the number of subproblems at level k (root = level 0), (ii) the size of each subproblem at level k, and (iii) the total cost contributed by level k.

Solution: At level k:

- (i) The number of subproblems is a^k.
- (ii) The size of each subproblem is n/b^k.
- (iii) The total cost at level k is $a^k f(n/b^k)$.

(c) (5 points) Determine: (i) the number of levels in the tree, (ii) the number of leaf nodes, and (iii) the total cost contributed by the leaf level.

Solution:

- (i) The number of levels is $\log_b n + 1$ (from level 0 to level $\log_b n$).
- (ii) The number of leaf nodes is $a^{\log_b n} = n^{\log_b a}$.
- (iii) Since each leaf node costs $T(1) = \Theta(1)$, the total leaf cost is $\Theta(n^{\log_b a})$.

(d) (5 points) Express the total cost as a summation over all levels (internal + leaf). Assuming the case where $f(n) = \Theta(n^{\log_b a})$, evaluate the sum and derive the asymptotic bound for $T(n)$.

Solution: The total cost is:

$$T(n) = \sum_{k=0}^{\log_b n - 1} a^k f(n/b^k) + \Theta(n^{\log_b a})$$

Assume $f(n) = \Theta(n^{\log_b a})$, so $f(n) \leq c \cdot n^{\log_b a}$ for some constant c. Then:

$$a^k f(n/b^k) \leq a^k \cdot c \left(\frac{n}{b^k}\right)^{\log_b a} = c \cdot n^{\log_b a} \left(\frac{a}{b^{\log_b a}}\right)^k = c \cdot n^{\log_b a}(1)^k = c \cdot n^{\log_b a}$$

Thus, the sum of internal nodes is:

$$\sum_{k=0}^{\log_b n - 1} a^k f(n/b^k) \leq \sum_{k=0}^{\log_b n - 1} c \cdot n^{\log_b a} = c \cdot n^{\log_b a} \log_b n = \Theta(n^{\log_b a} \log n)$$

Adding the leaf cost $\Theta(n^{\log_b a})$, the overall asymptotic bound is:

$$T(n) = \Theta(n^{\log_b a} \log n)$$

3. (10 points) **Boat Rescue**
You are a rescue coordinator on a sinking island. There are n people, each with weight w_i. You have an unlimited number of identical boats, each with a maximum weight capacity of W. Each boat can carry **at most 2 people**. Your goal is to rescue everyone using the **minimum number of boats**.

(a) (5 points) Describe an algorithm that solves this problem. Write it in 3–4 clear steps or pseudocode.

Solution: A greedy approach works:

1. Sort the weights of the people in non-decreasing order: $w_1 \leq w_2 \leq \cdots \leq w_n$.
2. Initialize two pointers: `left = 1` (lightest person) and `right = n` (heaviest person).
3. While `left <= right`:

- If `left == right`, assign this last person to a boat and terminate.
- If $w_{\text{left}} + w_{\text{right}} \leq W$, they can share a boat. Increment `left` and decrement `right`.
- Else, the heaviest person cannot share a boat with anyone (even the lightest person is too heavy). Assign them their own boat. Decrement `right`.
- Increment the boat counter.

(b) (5 points) Apply your algorithm to the following instance. Show each step.

$$n = 5, \quad \text{weights: } [30, 50, 60, 70, 80], \quad W = 100$$

How many boats does your algorithm use?

Solution: The sorted weights are $[30, 50, 60, 70, 80]$.

- **Step 1:** `left = 30`, `right = 80`. $30 + 80 = 110 > 100$. 80 gets its own boat. `right` moves to 70. (Boat 1)
- **Step 2:** `left = 30`, `right = 70`. $30 + 70 = 100 \leq 100$. They share a boat. `left` moves to 50, `right` moves to 60. (Boat 2)
- **Step 3:** `left = 50`, `right = 60`. $50 + 60 = 110 > 100$. 60 gets its own boat. `right` moves to 50. (Boat 3)
- **Step 4:** `left = 50`, `right = 50` (`left == right`). 50 gets its own boat. (Boat 4)

The algorithm uses **4 boats**.

4. (15 points) **Majority Element**
Given an array $A[1..n]$ of integers, an element x is called a **majority element** if it appears **more than** $n/2$ times. Not every array has a majority element.

(a) (8 points) Design an algorithm that returns a majority element if one exists, and returns `None` otherwise. Describe your algorithm in 2–3 clear sentences. Then trace your algorithm on the array $A = [2, 3, 3, 2, 3, 3, 3]$, showing the recursive calls and the result at each level.

Solution: We can use a Divide and Conquer strategy. **Algorithm Description:** Split the array into two halves, recursively find the majority candidate for the left and right halves. If the candidates match, return that candidate. If they differ, count the frequency of both candidates in the current array segment and return the one that occurs more than $n/2$ times (or return `None` if neither does).
Trace on $A = [2, 3, 3, 2, 3, 3, 3]$**:**

- Level 0: Split into $L = [2, 3, 3, 2]$ and $R = [3, 3, 3]$.
- Level 1 (L): Split into $L_L = [2, 3]$ and $L_R = [3, 2]$.

- L_L returns `None` (no majority).
- L_R returns `None`.
- Combine: Since both are `None`, count candidates 2 and 3 in L: both appear twice (not $> 4/2$). L returns `None`.

- Level 1 (R): Split into $R_L = [3]$ and $R_R = [3, 3]$.
  - R_L returns 3.
  - R_R returns 3.
  - Combine: Since they match, R returns 3.
- Level 0 Combine: L returned `None`, R returned 3. Count occurrences of 3 in the entire array A: 3 appears 5 times, which is $> 7/2 = 3.5$.
- Final Result: **3**.

(b) (7 points) Write the recurrence relation for your algorithm's worst-case running time. [3 marks] Solve it using the Master Theorem. [4 marks]

Solution: The algorithm divides the problem of size n into two subproblems of size $n/2$ and does linear work $O(n)$ to count the occurrences of the candidates in the combine step. Recurrence relation:

$$T(n) = 2T(n/2) + O(n)$$

Master Theorem Solution: Here $a = 2$, $b = 2$, and $f(n) = O(n)$. We compare $f(n)$ with $n^{\log_b a} = n^{\log_2 2} = n^1 = n$. Since $f(n) = \Theta(n)$, it falls under Case 2 of the Master Theorem. Thus, the solution is:

$$T(n) = \Theta(n \log n)$$

5. (15 points) **Longest Common Subsequence**
Given two strings $X[1..m]$ and $Y[1..n]$, the Longest Common Subsequence (LCS) problem asks for the length and structure of the maximum length subsequence common to both.

(a) (6 points) **Methodology:** Define the subproblems $c[i, j]$ and provide the complete recurrence relation, including base cases. State the time and space complexity of your algorithm.

Solution: Let $c[i, j]$ be the length of the LCS of the prefix $X[1..i]$ and $Y[1..j]$. The recurrence relation is:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \\ \max(c[i-1, j], c[i, j-1]) & \text{if } X[i] \neq Y[j] \end{cases}$$

Time Complexity: $O(mn)$ as we fill an $m \times n$ table. Space Complexity: $O(mn)$ to

store the table.

(b) (5 points) **Table Construction:** For strings $X = \text{ABCD}$ and $Y = \text{AXCD}$, complete the DP table below:

Solution: The completed DP table is:

$c[i,j]$	$j=0$ ($\emptyset$)	$j=1$ (**A**)	$j=2$ (**X**)	$j=3$ (**C**)	$j=4$ (**D**)
	$\emptyset$	**A**	**X**	**C**	**D**
$i=0$ ($\emptyset$)	0	0	0	0	0
$i=1$ (**A**)	0	1	1	1	1
$i=2$ (**B**)	0	1	1	1	1
$i=3$ (**C**)	0	1	1	2	2
$i=4$ (**D**)	0	1	1	2	3

(c) (4 points) **Traceback:** Perform a traceback on your filled DP table to identify the LCS. Clearly show your path, state the final LCS string, and report the length of the result.

Solution: Start at $c[4,4] = 3$:

- $X[4] = Y[4] =$ 'D', so we add 'D' to the LCS and move diagonally to $c[3,3] = 2$.
- $X[3] = Y[3] =$ 'C', so we add 'C' to the LCS and move diagonally to $c[2,2] = 1$.
- $X[2] =$ 'B' $\neq Y[2] =$ 'X'. We look at $c[1,2] = 1$ and $c[2,1] = 1$. Since they are equal, we can go to either. Let's move to $c[1,2] = 1$.
- $X[1] =$ 'A' $\neq Y[2] =$ 'X'. Move to $c[1,1] = 1$.
- $X[1] = Y[1] =$ 'A', so we add 'A' to the LCS and move diagonally to $c[0,0] = 0$.

Path of cells: $(4,4) \to (3,3) \to (2,2) \to (1,2) \to (1,1) \to (0,0)$. LCS: **ACD** Length of LCS: **3**.

6. (15 points) **Data Compression - Huffman Code**
The text below is from a well-known pop culture reference:

`maytheforcebewithyou`

(a) (3 points) Count the frequency of each distinct letter in the text. How many total characters does the string contain and what are their frequencies? Answer your results in the below format:

Solution: The string contains 20 total characters. Frequencies:

Letter	m	a	y	t	h	e	f	o	r	c	b	w	i	u
Freq	1	1	2	2	2	2	1	2	1	1	1	1	1	2

(b) (7 points) Construct an optimal prefix-free binary code for these characters using Huffman code. Show the initial sorted frequencies, each merge operation (which two nodes were merged and the resulting parent's frequency), and the final tree structure.

Solution: Sorted frequencies:

$$a:1, b:1, c:1, f:1, i:1, m:1, r:1, w:1, e:2, h:2, o:2, t:2, u:2, y:2$$

Merge operations (one valid sequence):

1. Merge $a:1, b:1 \rightarrow ab:2$
2. Merge $c:1, f:1 \rightarrow cf:2$
3. Merge $i:1, m:1 \rightarrow im:2$
4. Merge $r:1, w:1 \rightarrow rw:2$
5. Merge $e:2, h:2 \rightarrow eh:4$
6. Merge $o:2, t:2 \rightarrow ot:4$
7. Merge $u:2, y:2 \rightarrow uy:4$
8. Merge $ab:2, cf:2 \rightarrow abcf:4$
9. Merge $im:2, rw:2 \rightarrow imrw:4$
10. Merge $eh:4, ot:4 \rightarrow ehot:8$
11. Merge $uy:4, abcf:4 \rightarrow uyabcf:8$
12. Merge $imrw:4, ehot:8 \rightarrow imrwehot:12$
13. Merge $uyabcf:8, imrwehot:12 \rightarrow$ Root $:20$

(c) (5 points) Assign binary codes (left branch = 0, right branch = 1) to each letter and list them in a table. [2 marks] Encode the original string `"maytheforcebewithyou"` using your code and state the total number of bits. [3 marks]

Solution: Assigning codes based on the tree (left=0, right=1):

- `uyabcf` is left (0), `imrwehot` is right (1).
- Left subtree (0):
  - `uy` (00) → `u` (000), `y` (001)
  - `abcf` (01)
    - `ab` (010) → `a` (0100), `b` (0101)
    - `cf` (011) → `c` (0110), `f` (0111)
- Right subtree (1):
  - `imrw` (10)
    - `im` (100) → `i` (1000), `m` (1001)

        * `rw` (101) → `r` (1010), `w` (1011)
    - `ehot` (11)
        * `eh` (110) → `e` (1100), `h` (1101)
        * `ot` (111) → `o` (1110), `t` (1111)

Codes table:

Letter	Code	Letter	Code
m	1001	e	1100
a	0100	o	1110
y	001	r	1010
t	1111	c	0110
h	1101	b	0101
f	0111	w	1011
i	1000	u	000

Encoding of `"maytheforcebewithyou"`: `1001 0100 001 1111 1101 1100 0111 1110 1010 0110 1100 0101 1100 1011 1000 1111 1101 001 1110 000`

Total bits = $\sum$(Frequency × Code Length):

- 1-freq letters (a, b, c, f, i, m, r, w): $8 \times 4 = 32$ bits
- 2-freq letters (e, h, o, t, u, y):
    - e, h, o, t: 4 bits each ×4 letters ×2 freq = 32 bits
    - u, y: 3 bits each ×2 letters ×2 freq = 12 bits

Total bits = $32 + 32 + 12 =$ **76** bits.