Advanced Algorithms

W1) Circular Queue Trace (Wrap-around + Full/Empty)

A circular queue has `MAX = 6`, initially `front = -1`, `rear = -1`. Use modulo indexing for wrap-around.

Operations:

```
enq(12), enq(5), enq(19), deq(), enq(7), enq(3), deq(), enq(8), enq(6),
enq(10)
```

1. Create a table with one row per operation showing `front`, `rear`, and if any operation fails (full/empty).
2. Write the dequeued values in order.
3. Draw the final array slots `0..MAX-1` (show the stored values and empty slots).

W2) Queue Edge-case Reasoning (False Full)

Consider an array queue (not circular) with `SIZE=5`. A student claims: *"The queue is full iff rear == SIZE-1."*

1. Give a concrete sequence of `enqueue`/`dequeue` operations that makes the queue **incorrectly report full** even though there are unused slots.
2. State one correct fix (either data-structure change or algorithmic change).

W3) Priority Queue Scheduling

A priority queue stores `(jobID, priority)` where higher priority is served first. If priorities tie, serve earlier arrival first.

Arrival sequence:

$$(A, 2),\ (B, 4),\ (C, 4),\ (D, 1),\ (E, 3),\ (F, 4),\ (G, 2)$$

1. Write the complete service order (jobIDs only) if we repeatedly remove until empty.
2. After serving the first three jobs, list the remaining jobs in the order they will be served next.

W4) Sorting Stability Challenge

Each item is `(key, id)` and sorting is by `key` ascending:

$$[(2, a),\ (1, b),\ (2, c),\ (1, d),\ (3, e),\ (2, f)]$$

1. Write the final order if the algorithm is **stable**.
2. Give one example of a possible final order if the algorithm is **not stable**.
3. Name two common sorting algorithms that are stable and two that are not.

W5) Insertion Sort

Apply insertion sort to:

$$A = [5, 2, 4, 6, 1, 3]$$

1. Write the array after each outer-loop step ($i = 1$ to $n - 1$).
2. Count the total number of shifts of the form $A[j + 1] = A[j]$.

W6) Quick Sort Partition (Lomuto, pivot = last)

Perform exactly **one Lomuto partition** (do not recurse) with pivot = last element:

$$A = [7, 3, 7, 1, 5, 2, 7]$$

1. Show the values of i and j movements and each swap.
2. Write the array after partition and the final pivot index.
3. State which side contains elements $\leq$ pivot and which side contains $>$ pivot.

W7) Merge Sort Recursion Tree (Non power-of-two)

An array has $n = 10$ elements. Merge sort splits using:

$$mid = \left\lfloor \frac{lb + ub}{2} \right\rfloor$$

1. Draw the recursion splitting tree showing subarray index ranges until size 1.
2. How many merge operations occur in total?
3. What is the maximum recursion depth (levels including the root call)?

W8) Construct a Binary Tree from Traversals

A binary tree has:

Inorder: 4 2 5 1 6 3 7

Postorder: 4 5 2 6 7 3 1

1. Draw the unique binary tree.

2. Write the preorder traversal of your tree.

W9) Consider the undirected graph with vertices:

$$V = \{A, B, C, D, E, F, G\}$$

Edges:

$$E = \{(A, A), (A, C), (B, G), (B, E), (C, F), (E, F), (D, G), (F, G), (D, A), (G, G)\}$$

1. Run DFS starting from **A**. Write all the visit orders.
2. Write the spanning tree.

W10) Adjacency Matrix Reasoning

Given this adjacency matrix of an undirected graph with vertices 1..5:

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 0 & 1 \\ 4 & 1 & 1 & 0 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

1. List all edges.
2. Compute the degree of each vertex.
3. How many connected components does the graph have?

W11) Tree Properties from Node Count

A **perfect binary tree** has exactly $N = 511$ nodes.

1. Find the number of **levels** L in the tree.
2. Find the **height** h of the tree, where height is the number of **edges** on the longest root-to-leaf path.
3. Find the number of **leaf nodes**.
4. Find the number of **internal nodes** (non-leaf nodes).

Case Study: Emergency Dispatch System

Scenario. A city emergency dispatch center receives incident calls. Each call is recorded as:

$$(\text{CallID},\ \text{Severity (1–10)},\ \text{Location})$$

Calls first enter a **waiting line (Queue)**. The dispatcher processes the next call, and then:

- adds its **Severity** to a **Max-Heap** (priority dispatch),
- inserts its **CallID** into a **BST** of active incidents,
- uses **BFS** on the city map (unweighted graph) to compute the route in minimum hops.

Incoming calls (in arrival order):

$$(50, 6, C),\ (30, 9, E),\ (70, 5, B),\ (20, 4, D),\ (40, 7, F),\ (60, 8, C),\ (80, 3, E)$$

Exercise 1:

The waiting line is an **array-based queue** with `SIZE = 6`, initial `front = -1`, `rear = -1`. Use the standard rules you learned for enqueue/dequeue in an array queue.

Operations:

```
enqueue CallID 50, 30, 70, 20, 40, 60, dequeue, dequeue, enqueue 80
```

1. Fill the table after each operation: **front**, **rear**, and the queue content (from front to rear).

Operation	front	rear	Queue (front → rear)
Initial state			
enqueue(50)			
enqueue(30)			
enqueue(70)			
enqueue(20)			
enqueue(40)			
enqueue(60)			
dequeue()			
dequeue()			
enqueue(80)			

2. If an **overflow** happens due to `rear` reaching the end even though there is free space at the beginning, explain **why** in 1–2 sentences and write one change you would make to avoid it (no stack allowed).

Wishing you success!