1. Definition of Heap

A **Heap** is a **complete binary tree** (all levels completely filled except possibly the last, and the last level has all keys as left as possible) which satisfies the **heap property**.

There are two main types of heaps:

- **Min Heap**: The parent is **less than or equal to** its children → the smallest element is at the root.
- **Max Heap**: The parent is **greater than or equal to** its children → the largest element is at the root.

Heap is typically implemented using an array.

2. Properties of Heap

Property	**Description**
Shape	Complete binary tree
Heap Order	Parent node is smaller (min heap) or larger (max heap) than children
Time for Insertion/Deletion	O(log n)
Time for Accessing Min/Max	O(1) (since it's the root)

3. Use Cases / Where Heaps Are Applied

Application	**Explanation**
Priority Queue	Heap is used to implement priority queues efficiently
Top K Elements	For finding top or smallest K elements in O(n log k)
Heap Sort	In-place sorting using heap (max heap)
Median of Stream	Min heap + max heap combo to maintain running median
Dijkstra's Algorithm	For efficiently getting the smallest distance vertex
Merge K Sorted Lists / Arrays	Min-heap helps merge k lists in

	O(N log k)

4. How to Apply Heaps in Problems

You apply heap when:

- You need **frequent access to the min or max** value.
- You need to maintain a **dynamic set of elements** where you perform **insertions + deletions**.
- You need to solve **top K / smallest K / largest K** problems.
- You're merging multiple sorted inputs efficiently.
- You want to simulate **priority-based processing**.

5. Java Implementation Tip

Java offers:

```
PriorityQueue<Integer> minHeap = new PriorityQueue<>();
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
```

7. Benefits of Using Heap

Benefit	Explanation
Fast Access	O(1) access to min/max
Efficient Updates	O(log n) insert/delete
Dynamic	Adapts to changes in the dataset
Memory Efficient	In-place heap sort possible
Versatile	Works for sorting, streaming, scheduling, and more

8. Example Problems

Problem	Heap Used	Why
Find K Largest Elements	Min Heap of size K	Maintain top K efficiently
Merge K Sorted Lists	Min Heap	Get smallest head each time

Running Median	Two Heaps (Min + Max)	Balance left/right halves
Task Scheduler	Max Heap	Pick most frequent task first
Top K Frequent Elements	Min Heap or Bucket Sort	Frequency prioritization
Dijkstra's Algorithm	Min Heap	Pick shortest path node quickly

Definition: What is K-Way Merge?
K-Way Merge is a pattern used to merge **K sorted lists/arrays** into a single **fully sorted list**. It extends the idea of merging two sorted arrays (like in Merge Sort) to **K arrays** using a **min-heap (or priority queue)**.

Core Idea (Intuition)

Instead of comparing all elements in all arrays, we maintain a **min-heap** of the **current smallest element from each list**.

- Always extract the **minimum element** from the heap.
- Push the **next element from the same list** (from where the min came) into the heap.
- Repeat until all elements are merged.

Example Problem:
"Merge K sorted arrays into one sorted array"

Step-by-step:

1. **Initialize min-heap** of size K
   Each heap entry stores: (value, arrayIndex, elementIndex)
2. **Push first element** from each of the K arrays into the heap
3. While heap is not empty:
   - Pop the **smallest element**
   - Add it to result array
   - Push the **next element** from that array into the heap (if exists)
4. Continue until all arrays are exhausted.

#	Problem Title	Type	Description
1	Kth Largest Element in an Array	Max Heap	Find the Kth largest using a max/min heap
2	Top K Frequent Elements	Min Heap	Use hashmap + heap to track frequency
3	Merge K Sorted Lists	Min Heap	Merge using priority queue of list nodes
4	Find Median from Data Stream	Min/Max Heap	Use two heaps to maintain median
5	K Closest Points to Origin	Max Heap	Use heap to keep top K closest points
6	Task Scheduler	Max Heap	Greedy + max heap of task frequencies
7	Reorganize String	Max Heap	Heap to always pick the most frequent char
8	Kth Smallest Element in a Sorted Matrix	Min Heap	Matrix traversal using heap
9	Minimum Cost to Connect Ropes	Min Heap	Greedy strategy using priority queue
10	Last Stone Weight	Max Heap	Smash top 2 stones

			repeatedly
11	Sliding Window Median	Min + Max Heap	Track medians as window moves
12	Sort Characters by Frequency	Max Heap	Sort based on frequency using heap
13	Kth Largest Element in a Stream	Min Heap	Maintain k-sized heap in stream
14	Trapping Rain Water II	Min Heap	Dijkstra-like BFS using heap
15	Smallest Range Covering Elements from K Lists	Min Heap	Use heap to track smallest range