Heap Sort

Heap Sort is a **comparison-based sorting algorithm** that uses a special data structure called a **Heap**.
It is based on the idea of using a **Binary Heap** (usually a Max Heap) to repeatedly extract the maximum element and build a sorted array.

Heap Sort is efficient and works in-place, but unlike Merge Sort and Quick Sort, it is **not stable**.

- A **Heap** is a complete binary tree that satisfies the **Heap Property**.
- **Max Heap:** The parent node is always greater than or equal to its children.
- **Min Heap:** The parent node is always smaller than or equal to its children.

Heap Sort generally uses a **Max Heap** to sort numbers in ascending order.

🔹 Steps in Heap Sort

1. **Build Max Heap** → Convert the array into a Max Heap (largest element at the root).
2. **Swap Root with Last Element** → Swap the maximum element (root) with the last element of the heap.
3. **Reduce Heap Size & Heapify** → Exclude the last (sorted) element and heapify the remaining elements to maintain heap property.
4. **Repeat** → Continue until all elements are sorted.

🔹 Example

Sort the array: `[4, 10, 3, 5, 1]`

- Step 1: Build Max Heap → `[10, 5, 3, 4, 1]`

- Step 2: Swap root with last → `[1, 5, 3, 4, 10]`
- Step 3: Heapify → `[5, 4, 3, 1, 10]`
- Step 4: Repeat process → `[1, 3, 4, 5, 10]`

Final Sorted Array: **[1, 3, 4, 5, 10]**

🔹 Time Complexity

- **Building Heap:** `O(n)`
- **Heapify (per element):** `O(log n)`
- **Overall:** `O(n log n)`
- **Best Case:** `O(n log n)`
- **Average Case:** `O(n log n)`
- **Worst Case:** `O(n log n)`

Unlike Quick Sort, Heap Sort's worst case is **not bad.**

🔹 Space Complexity

- `O(1)` → In-place sorting (only requires constant extra memory).

🔹 Advantages of Heap Sort

- **Guaranteed O(n log n)** time complexity.
- **In-place sorting** (no extra space like Merge Sort).

- Useful for **Priority Queue operations** (using heaps).

🔹 Disadvantages of Heap Sort

- **Not Stable:** Equal elements may not maintain their relative order.
- In practice, Heap Sort is often **slower than Quick Sort** due to cache inefficiency.