Heap Sort & The Heapify Method

A Comprehensive Technical Guide

1. Core Fundamentals of Heap Sort

Heap Sort is a highly efficient algorithm based on the **Heap Data Structure**. To understand how the entire process functions, one must master two prerequisite operations:

- **Build Heap:** The methodology of inserting data and organizing an unstructured array into a valid heap structure.
- **Deletion:** The process of extracting elements one by one from the heap while restructuring the tree to maintain its properties.

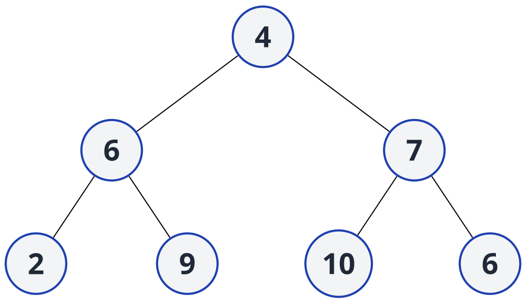
2. Tree Representation and Min-Heap Construction

Let us examine an unsorted array of elements:

$$A = [4, 6, 7, 2, 9, 10, 6]$$

To form the initial tree representation, elements are inserted level-by-level, from left to right, creating an **Almost Complete Binary Tree**.

Initial Tree (Before Heapify)



The Min-Heap Property

In a **Min-Heap**, the value at any parent node must always be *smaller than or equal to* the values of its child nodes.

The Bottom-Up Heapify Process

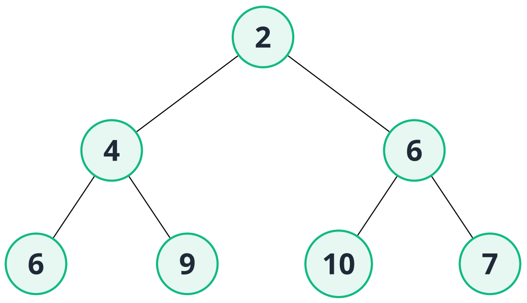
To convert our arbitrary tree into a valid Min-Heap, we skip the leaf nodes (which trivially satisfy the property) and apply the **Heapify** algorithm from the bottom up.

1. **Process Lowest Subtrees:** We evaluate nodes 6 and 7.

- Node 6 has children 2 and 9. Since $2 < 6$, we swap 6 and 2.
- Node 7 has children 10 and 6. Since $6 < 7$, we swap 7 and 6.

2. **Process Root Node:** Next, we look at the root node 4. Its new children are 2 and 6. Since $2 < 4$, we swap 4 and 2.

Final Valid Min-Heap



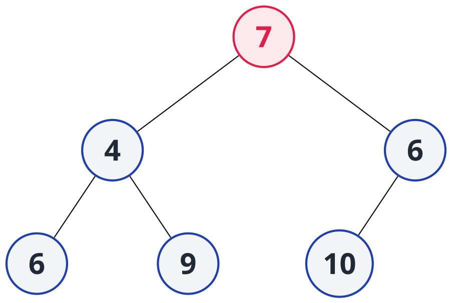
3. The Deletion Phase (Sorting Process)

Once the Min-Heap is successfully constructed, the sorting mechanism begins by continuously deleting elements.

Step 1. **Extract Root:** The root node (**2**), which is the smallest element, is removed and placed into the final sorted output array.

Step 2. **Promote the Last Element:** We take the rightmost element on the lowest level (**7**) and move it to the vacant root position.

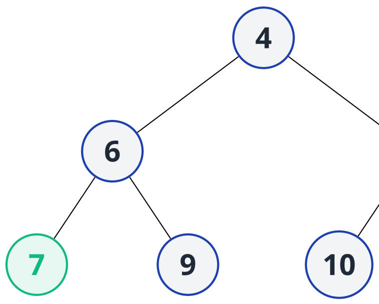
Tree After Root Deletion & Promotion



Because 7 is larger than its children (4 and 6), the Min-Heap property is violated. We must **Heapify** the root.

- Swap 7 with its smallest child (4).
- Node 7 is now the left child. Its children are 6 and 9. Swap 7 with 6.

Tree After Re-Heapify



We repeat this extraction and heapify cycle for every node until the tree is completely empty, yielding a perfectly sorted array.

4. Complexity Analysis

Time Complexity

The computational cost of Heap Sort operates in two distinct phases:

- **Building the Heap:** Constructing the initial heap using the optimized, bottom-up Heapify approach operates in strictly $\mathcal{O}(N)$ time. (*Note: The less efficient, sequential top-down element insertion would take* $\mathcal{O}(N \log N)$).
- **Deletion Phase:** Deleting a root element and restoring the heap takes $\mathcal{O}(\log N)$ time per node. Doing this for all N elements results in $\mathcal{O}(N \log N)$.

Since $N \log N$ is the dominating term, the overall **Time Complexity** is:

$$\mathcal{O}(\mathbf{N} \log \mathbf{N})$$

Space Complexity (In-Place)

Heap Sort modifies the elements directly within the bounds of the original array. It does not require any dynamic, external data structures.

In-Place Sorting

Because the algorithm demands zero extra allocated space to process the data, it is characterized as an **in-place** sorting algorithm. The **Space / Auxiliary Complexity** is constant: $\mathcal{O}(1)$.

5. Stability of Heap Sort

An algorithm is *stable* if it preserves the original relative order of elements that have identical values.

Verdict: Heap Sort is Unstable

Heap Sort is fundamentally an **unstable** sorting algorithm.

Proof via Duplicates Example:
Consider an initial array with duplicate items marked for identity:

$$A = [2, 6, 3_a, 3_b, 3_c, 4, 7]$$

When applying the heap formation and deletion steps, the aggressive physical swapping occurs. Specifically, when deleting the root, we continually rip elements from the bottom of the tree and sift them down.

Due to these non-adjacent, long-distance swaps, the original sequence of the identical values ($3_a \rightarrow 3_b \rightarrow 3_c$) gets irreversibly scrambled. The final output might group them as $3_c, 3_b, 3_a$. Because their relative initial positioning is destroyed, the algorithm is classified as mathematically unstable.