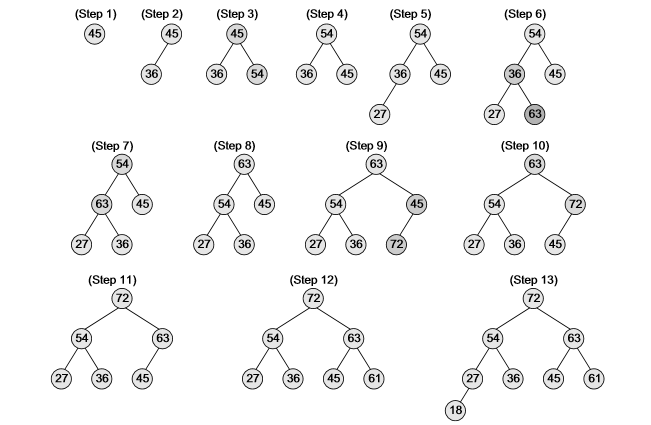
**AA LAB ASSIGNMENT | HEAP SORT**

**TITLE:**  Analysis, Proof of Analysis and Implementation of **Heap Sort**

**MECHANISM**

Heap Sort works by first creating a binary heap from the array, and then repeatedly extracting the maximum element and placing it at the end of the sorted array. To create a binary heap, we start from the last non-leaf node and move upwards, ensuring that each node is greater than or equal to its children. Once the binary heap is created, we repeatedly extract the maximum element (root node) and swap it with the last element of the unsorted array. We then decrease the heap size by 1 and heapify the remaining elements to maintain the heap property. We repeat this process until the entire array is sorted.



**ALGORITHM / PSEUDOCODE**

**HeapSort(A):**

1. Build a max heap from the input array A.
2. For i = n to 2: a. Swap A[1] with A[i]. b. Reduce heap size by 1. c. Heapify the root of the tree.
3. The sorted array is A.

**BuildMaxHeap(A):**

1. Initialize heap size to n.
2. For i = floor(n/2) to 1: a. Call MaxHeapify(A, i).

**MaxHeapify(A, i):**

1. Set l = 2i, r = 2i + 1, and largest = i.
2. If l ≤ heap size and A[l] > A[largest], then set largest = l.
3. If r ≤ heap size and A[r] > A[largest], then set largest = r.
4. If largest ≠ i, then swap A[i] and A[largest].
5. Recursively call MaxHeapify(A, largest) until the heap property is satisfied.

**IMPLEMENTATION**

*#include* <iostream>

using namespace std;

void heapify(int *arr*[], int *n*, int *i*) {

    int largest = *i*; *// Initialize largest as root*

    int left = 2 \* *i* + 1; *// left = 2\*i + 1*

    int right = 2 \* *i* + 2; *// right = 2\*i + 2*

*// If left child is larger than root*

*if* (left < *n* && *arr*[left] > *arr*[largest])

        largest = left;

*// If right child is larger than largest so far*

*if* (right < *n* && *arr*[right] > *arr*[largest])

        largest = right;

*// If largest is not root*

*if* (largest != *i*) {

        swap(*arr*[*i*], *arr*[largest]);

*// Recursively heapify the affected sub-tree*

        heapify(*arr*, *n*, largest);

    }

}

void heapSort(int *arr*[], int *n*) {

*// Build heap (rearrange array)*

*for* (int i = *n* / 2 - 1; i >= 0; i--)

        heapify(*arr*, *n*, i);

*// One by one extract an element from heap*

*for* (int i = *n* - 1; i >= 0; i--) {

*// Move current root to end*

        swap(*arr*[0], *arr*[i]);

*// call max heapify on the reduced heap*

        heapify(*arr*, i, 0);

    }

}

int main() {

    int arr[] = {12, 11, 13, 5, 6, 7};

    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array: ";

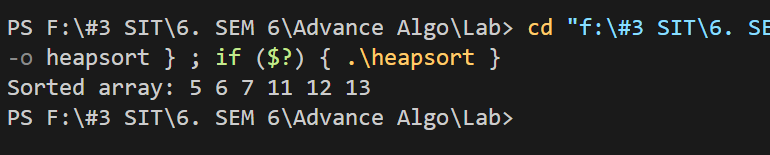
*for* (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

*return* 0;

}



**T(n) ANALYSIS WITH PROOF**

The time complexity of Heap Sort is **O(n\*logn).**

**SPACE COMPLEXITY ANALYSIS**

The space complexity of Heap Sort **is O(1**), which means that it uses a constant amount of extra space.

**ADVANTAGES / DISADVANTAGES**

|  |  |
| --- | --- |
| **ADVANTAGE** | **DISADVANTAGES** |
| Efficient for large datasets | Slower average case performance than Quick Sort and Merge Sort |
| Time complexity of O(n\*logn) | Not a stable sorting algorithm |
| Space complexity of O(1) |  |

**REAL LIFE APPLICATIONS:**

Database systems for sorting large amounts of data.

Operating systems for managing memory allocation and task scheduling.

Network routing algorithms for finding the shortest path between two nodes.

**OPTIMIZATIONS AND ADVANCEMENTS:**

Using a binary heap instead of a ternary or d-ary heap: Binary heaps have a simpler structure and can be implemented more efficiently, which can improve the overall performance of Heap Sort.

Using a heap data structure that supports efficient extraction of the maximum element: By using a heap data structure that supports efficient extraction of the maximum element, such as a Fibonacci heap, the time complexity of Heap Sort can be reduced from O(nlogn) to O(n + mlogn), where m is the number of times the heap is modified during the sorting process.