**Experiment No. – 2.3**

**Aim**: Write a program to implement Heap sort along with its complexity analysis.

1. **Problem Description:**

We have given an unsorted array of numbers, generate a sorted array of numbers by applying Heap Sort. Demonstrate knowledge of time complexity of Heap Sort and also find space complexity.

1. **Algorithm:**

HeapSort(arr)

BuildMaxHeap(arr)

for i = length(arr) to 2

    swap arr[1] with arr[i]

        heap\_size[arr] = heap\_size[arr] ? 1

        MaxHeapify(arr,1)

End

**BuildMaxHeap(arr)**

BuildMaxHeap(arr)

    heap\_size(arr) = length(arr)

    for i = length(arr)/2 to 1

MaxHeapify(arr,i)

End

**MaxHeapify(arr,i)**

MaxHeapify(arr,i)

L = left(i)

R = right(i)

if L ? heap\_size[arr] and arr[L] **>** arr[i]

largest = L

else

largest = i

if R ? heap\_size[arr] and arr[R] **>** arr[largest]

largest = R

if largest != i

swap arr[i] with arr[largest]

MaxHeapify(arr,largest)

End

1. **Complexity Analysis**

**Time complexity of Insertion Sort**

* **Best Case Complexity - O(n logn).** It occurs when the array is already sorted.
* **Average Case Complexity - O(n log n).** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending.
* **Worst Case Complexity -** **O(n log n).** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order.

**Space Complexity**: O(1). As no extra space is used while sorting.

1. **Pseudo Code**

HeapSort(A)

BuildHeap(A)

for i <- length(A) downto 2

exchange A[1] <-> A[i]

heapsize <- heapsize - 1

Heapify(A, 1)

BuildHeap(A)

heapsize <- length(A)

for i <- floor((heapsize - 1) / 2) downto 0

Heapify(A, i)

Heapify(A, i)

largest <- i

l <- 2 \* i + 1

r <- 2 \* i + 2

if l <= heapsize and A[l] > A[largest]

largest <- l

if r <= heapsize and A[r] > A[largest]

largest <- r

if largest != i

exchange A[i] <-> A[largest]

Heapify(A, largest)

1. **Source Code (C/C++):**

#include <bits/stdc++.h>

using namespace std;

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

{

    // Find largest among root, left child and right child

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

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

        largest = left;

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

        largest = right;

    // Swap and continue heapifying if root is not largest

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

    // Build max heap

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

        heapify(arr, n, i);

    // Heap sort

    for (int i = n - 1; i >= 0; i--)

    {

        swap(arr[0], arr[i]);

        // Heapify root element to get highest element at root again

        heapify(arr, i, 0);

    }

}

// Print an array

void print(int arr[], int n)

{

    for (int i = 0; i < n; ++i)

        cout << arr[i] << " ";

    cout << "\n";

}

int main()

{

    cout << "Name: Saurabh Kumar \nUID: 23MAI10004\n";

    cout << "\nHeap Sort\n";

    int n;

    cout << "Enter Size of array\n";

    cin >> n;

    int arr[n];

    cout << "Enter array elements \n";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout << "\nArray before sorting: ";

    print(arr, n);

    heapSort(arr, n);

    cout << "Array after sorting : ";

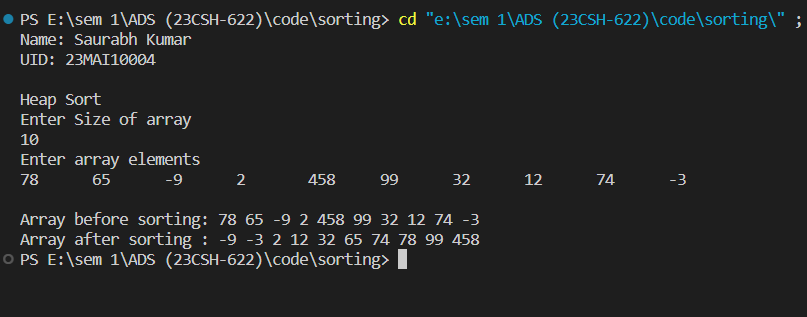
    print(arr, n);

    return 0;

    return 0;

}

1. **Screenshot of Outputs:**

****

1. **Learning Outcomes**
2. Learnt about how to sort an array.
3. Learnt about how to implement Heap sort.
4. Learnt about how to find time and space complexity
5. Learnt about the heapify maxheap minheap.
6. Learnt about how to use function and loop in c++.