Experiment-2.3

Aim of the Experiment :

Write a program to implement the Heap sort along with its complexity analysis.

1. Problem Description :

Heap Sort: Heap Sort is a popular and efficient sorting algorithm. Heap sort is a comparison-based sorting technique based on Binary Heap data structure. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list. Heap sort is the in-place sorting algorithm.

2. Algorithm :

HEAPSORT(A)

```
BUILD-MAX-HEAP(A)
for i ← length[A] down to 2
    do exchange A[1] ↔ A[i]
        MAX-HEAPIFY(A, 1, i - 1)
```

BUILD-MAX-HEAP(A)

```
n = length[A]
for i ← floor(n/2) down to 1
    do MAX-HEAPIFY(A, i, n)
```

MAX-HEAPIFY(A, i, n)

```
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ n and A[l] > A[i]
   then largest ←l
   else largest ←i
if r ≤ n and A[r] > A[largest]
   then largest ←r
if largest     i
   then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest, n)
```

3. Complexity Analysis:

Time Complexity: Heap Sort has a time complexity of O(n log n) in all cases. This makes it efficient for sorting large datasets. The log n factor comes from the height of the binary heap, and it ensures that the algorithm maintains good performance even with a large number of elements.

1) Best Case: O(n log(n)).

2) Worst Case: O(n log(n)).

3) Average Case: O(n log(n)).

Space Complexity: The space complexity of Heap Sort is O(1).

4. Pseudo Code :

```
procedure heapSort()
        // Array A, size N
        heapSort()
        For all non-leaf elements (i=N/2-1;i>=0;i--)
                Build Heap (Heapify)
        Initialize indexEnd
        While indexEnd>1
                Swap(A[0],A[indexEnd]
                indexEnd=indexEnd-1
        Build heap (apply heapify on the root node), considering array  from A[0] to A[indexEnd]
        Output the sorted array[]

        end heapSort()
end procedure
```

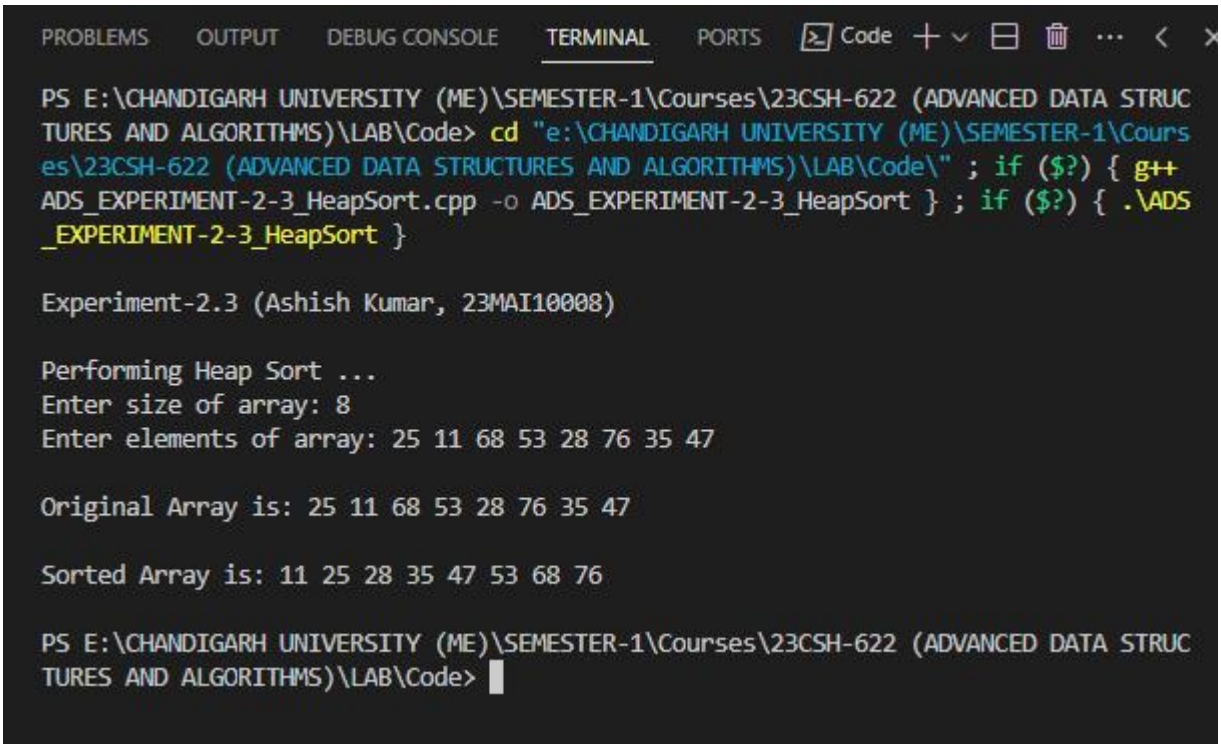
5. Source Code for Experiment :

```
#include <iostream>
using namespace std;

void printArray(int arr[], int n) {
   for(int i=0; i<n; i++) {
      cout<<arr[i]<<" ";
   }
   cout<<endl;
}
```

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

    // Initialize largest as root
    int largest = i;

    int l = 2 * i + 1;
    int r = 2 * i + 2;

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest]) {
        largest = l;
    }

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest]) {
        largest = r;
    }

    // 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() {

    cout<<"\nExperiment-2.3 (Ashish Kumar, 23MAI10008)"<<endl<<endl;
    cout<<"Performing Heap Sort ..."<<endl;

    int n;
    int arr[100];
    cout<<"Enter size of array: ";
    cin>>n;

    cout<<"Enter elements of array: ";
    for(int i=0; i<n; i++){
        cin>>arr[i];
    }

    cout<<"\nOriginal Array is: ";
    printArray(arr,n);
    cout<<endl;

    // Heap Sort Function
    heapSort(arr,n);

    cout<<"Sorted Array is: ";
    printArray(arr,n);
    cout<<endl;

    return 0;
}
```

6. Result/Output :



Learning outcomes (What I have learnt):

1. I learnt about how to input elements in an array.

2. I learnt about how to perform heap sort in an array.

3. I learnt about the binary heap data structure.

4. I learnt about differences between max heap and min heap.

5. I learnt about time and space complexity of heap sort.