Name: Achyut Shukla Batch: AA – A1 PRN: 20070122005

Heap Sort Algorithm

Heap sort is a comparison-based sorting algorithm that operates by first building a binary heap from the array of elements to be sorted, and then removing the largest element (in a max-heap) or the smallest element (in a min-heap) from the heap and placing it at the end of the array. This process is repeated for the remaining elements until the entire array is sorted.

The time complexity of heap sort is O(n log n), where n is the number of elements to be sorted. This is because building a heap takes O(n) time, and we need to perform n-1 heap removals and heapifications, each of which takes O(log n) time. The space complexity of heap sort is O(1) because it sorts the array in place.

Pseudo Code for HeapSort function:

```
First convert the array into heap data structure using Heapify
One by one delete the root node of the Max-heap and replace it with the last node in the heap
Heapify the root of the heap.
Repeat this process until size of heap is greater than 1.
```

Pseudo code for Heapify

```
Root = array[0]
Largest = largest( array[0], array [2 * 0 + 1]/ array[2 * 0 + 2])

if(Root != Largest)
  Swap(Root, Largest)
```

Code:

```
#include <iostream>

using namespace std;


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

{

   int largest = i;

   int l = 2 * i + 1;

   int r = 2 * i + 2;


   if (l < N && arr[l] > arr[largest])

      largest = l;
```

```
    if (r < N && arr[r] > arr[largest])
        largest = r;

    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, N, largest);
    }
}

void heapSort(int arr[], int N)
{

    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    for (int i = N - 1; i > 0; i--)
    {
        swap(arr[0], arr[i]);
        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 is \n";
    for (int i = 0; i < N; ++i)
```

```
    cout << arr[i] << " ";
  cout << "\n";
}
```

Complexity analysis of Quicksort

$$O(n)\ for\ heapify\ and\ O(1)\ for\ the\ sort$$

Worst Case - $\theta(nlogn)$

Best Case - $\theta(nlogn)$

Average Case - $\theta(nlogn)$

Space Complexity - $\theta(1)$

Advantages of heapsort:

- Efficiency: Heap sort has a time complexity of O(nlogn) for both worst and average cases. This makes it faster than many other sorting algorithms like bubble sort or insertion sort.
- Memory Usage: Memory usage is minimal because apart from what is necessary to hold the initial list of items to be sorted, it needs no additional memory space to work
- Simplicity: It is simpler to understand than other equally efficient sorting algorithms because it does not use advanced computer science concepts such as recursion.

Disadvantages of Heap Sort:

- Costly: Heap sort has a relatively high time complexity of O(nlogn) for the best case, which is when the input array is already sorted. This means that heap sort may not be the best choice when sorting an already sorted array.
- Unstable: Heap sort is unstable. It might rearrange the relative order.
- Efficient: Heap Sort are not very efficient when working with highly complex data.