**WORKSHEET 2.3**

**1. Aim:**

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

**2. Problem Description:**

To implement heap sort along with its complexity analysis

**3. Algorithm:**

Step 1 : heapify method

heapify(arr , i)

leftChild = arr [2\*0 + 1];

rightChild = arr [2\*0 + 2];

maxIndex = max( arr[i], leftChild, rightChild)

if(i != maxIndex)

swap(arr[i], arr[maxIndex])  
Step 2 : BuildMaxHeap(arr)   
 buildMaxHeap(arr)

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

heapify(arr, i);  
Step 3 : HeapSort method

Heapsort(arr)

buildMaxHeap(arr)

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

swap(&arr[0], &arr[i]);

heapsize--;

maxHeapify(arr,0); }

Step 4 : End

**4. Computational Complexity:-**

* The heap sort algorithm’s best, worst, and average [time complexities](https://builtin.com/software-engineering-perspectives/time-complexity) are all the same —[O(n\*log(n))](https://builtin.com/software-engineering-perspectives/nlogn).
* The time it takes to sort the array increases logarithmically with the size of the array.
* However, some[optimized](https://link.springer.com/chapter/10.1007/978-3-319-11933-5_78) versions of the algorithm can provide a best-case time complexity of O(n) by checking if the array is already sorted and, in that case, making the algorithm run in linear time.
* Space complexity is: O(1), hence, No auxiliary space is required in Heapsort implementation that is we are not using any arrays, linked list, stack, queue, etc to store our elements

**5. 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

**6. Source 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);

}

}

void printArray(int arr[], int N)

{

for (int i = 0; i < N; ++i)

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

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

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

cout << "Array before sorting : \n";

printArray(arr, N);

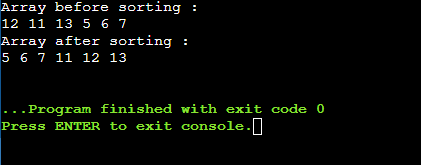
heapSort(arr, N);

cout << "Array after sorting : \n";

printArray(arr, N);

}

1. **Screenshot of Output:**



1. **Learning & Outcomes:**

* Learned about the heap sort algorithm, how it works, How much is the time complexity.
* Learned to create dynamic array using pointers.
* Learnt about max heap and min heap and does these work.