**LAB – 8**

**NAME – TUSHAR MAROO**

**SECTION – A**

**REGISTRATION NUMBER – 190905238**

**ROLL NUMBER – 40**

**Q1.**

**CODE**

#include <stdio.h>

#include <stdlib.h>

int opcount = 0;

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void TD(int heap[], int curr)

{

int parent = (curr - 1) / 2;

while (parent >= 0)

{

opcount++;

if (heap[parent] < heap[curr])

{

swap(&heap[parent], &heap[curr]);

curr = parent;

parent = (curr - 1) / 2;

}

else

return;

}

}

int main()

{

int heap[100], n, i, j;

printf("Enter the number of elements : ");

scanf("%d", &n);

printf("Enter the elements : ");

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

{

scanf("%d", &heap[i]);

TD(heap, i);

}

printf("Heapified array : ");

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

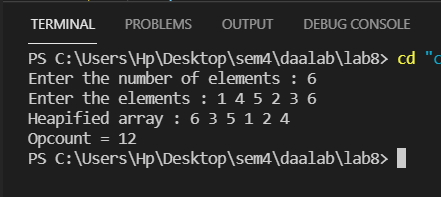
printf("%d ", heap[i]);

printf("\nOpcount = %d\n", opcount);

return 0;

}

**OUTPUT**

****

**GRAPH**

**ANALYSIS**

The time complexity of the Top Down (TD) function would be O(logn) as in the worst case the element on which top down is applied can be maximum and then we need to get it to the root of the heap. To do that we need to swap numbers h times h being the height of the tree. Since height of a binary tree is logn therefore the time complexity is O(logn). The best case for top down would be O(1) the case in which the element added is smaller than its parent.

To make a heap we need to call top down for all the elements of the array thus the worst-case time complexity for the creation would be O(nlogn) and the best-case tome complexity would be O(n).

**Q2.**

**CODE**

#include <stdio.h>

#include <stdlib.h>

int opcount = 0;

void heapify(int h[], int l, int n)

{

int i, k, v, heapify, j;

for(i = (n/2); i>=l; i--)

{

k = i; v = h[k]; heapify = 0;

while(heapify == 0 && 2\*k <= n)

{

j = 2\*k;

opcount++;

if(j<n)

if(h[j]<h[j+1])

j = j+1;

if(v>=h[j])

heapify = 1;

else

{

h[k] = h[j];

k = j;

}

}

h[k] = v;

}

return;

}

void heapSort(int arr[], int n)

{

int k = 0;

for(int i = 1; i<=n; i++)

{

heapify(arr, 1, n - k);

int temp = arr[1];

arr[1] = arr[n-k];

arr[n-k] = temp;

k++;

}

}

int main()

{

int heap[100], n, i, j;

printf("Enter the number of elements : ");

scanf("%d", &n);

printf("Enter the elements : ");

for (i = 1; i <= n; i++)

scanf("%d", &heap[i]);

heapSort(heap, n);

printf("Sorted array : ");

for (i = 1; i <= n; i++)

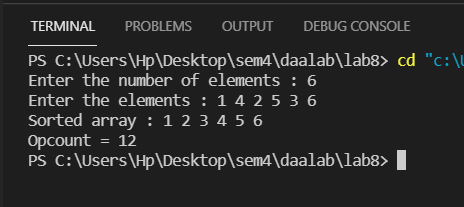
printf("%d ", heap[i]);

printf("\nOpcount = %d\n", opcount);

return 0;

}

**OUTPUT**

****

**GRAPH**

**ANALYSIS**

The time complexity for heapify function is O(logn). Since we need to call heapify n times to sort the heap thus the time complexity for heapsort is O(nlogn). For the creation of a heap from an array heapify is called n/2 times thus the time complexity for creation of heap is also O(nlogn).