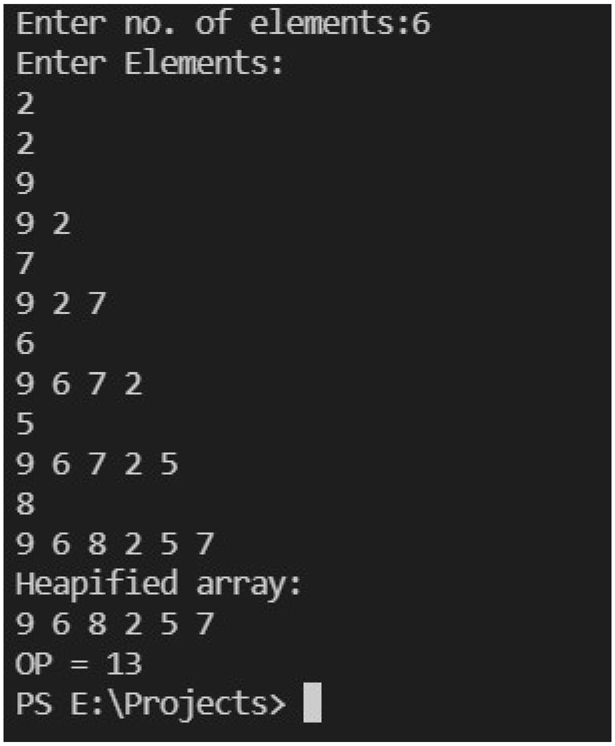
LAB 8

Aniket sambher
Section-A
Roll no-58

Q1. Write a program to create a heap for the list of integers using top-down heap construction algorithm and analyze its time efficiency. Obtain the experimental results for order of growth and plot the result.

```
#include <stdio.h>
#include <stdlib.h>


int op = 0;


void topDown(int arr[], int currIndex)
{
  int parent = currIndex/2; //if parent is i, children are 2i and 2i+1, dividing child index by 2 gives parent
  op++;
  while(parent > 0)
  {
  op++;
    if(arr[parent]<arr[currIndex])
    {
      int temp = arr[parent];       //swap if child > parent
      arr[parent] = arr[currIndex];
      arr[currIndex] = temp;

      currIndex = parent;
      parent = currIndex/2;
    }
```

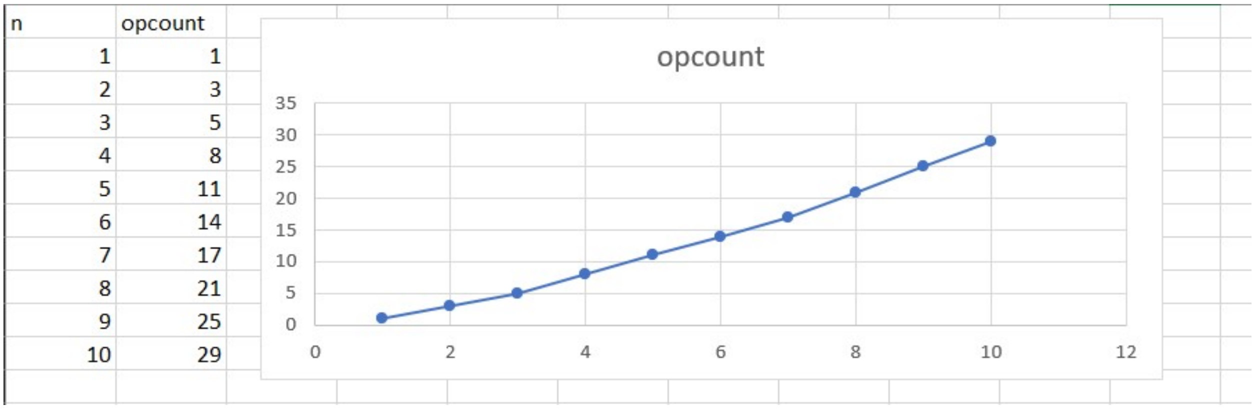
```
else return ;}

 int main()
 {
   int n;
   printf("Enter no. of elements:");
   scanf("%d", &n);
   int h[n+1];
   printf("Enter Elements:\n");

   for(int i = 1; i<=n; i++)
   {
     scanf("%d", &h[i]);
     topDown(h, i);
     for(int k = 1; k<=i; k++)
       printf("%d ", h[k]);
     printf("\n");
   }

   printf("Heapified array:\n");

   for(int i = 1; i<=n; i++)
     printf("%d ", h[i]);

   printf("\n");
   printf("OP = %d\n", op);

   return 0;
```



```
Enter no. of elements:6
Enter Elements:
2
2
9
9 2
7
9 2 7
6
9 6 7 2
5
9 6 7 2 5
8
9 6 8 2 5 7
Heapified array:
9 6 8 2 5 7
OP = 13
PS E:\Projects>
```

Graph and analysis:

For an input of array in ascending order, the opcount is close to n*logn.



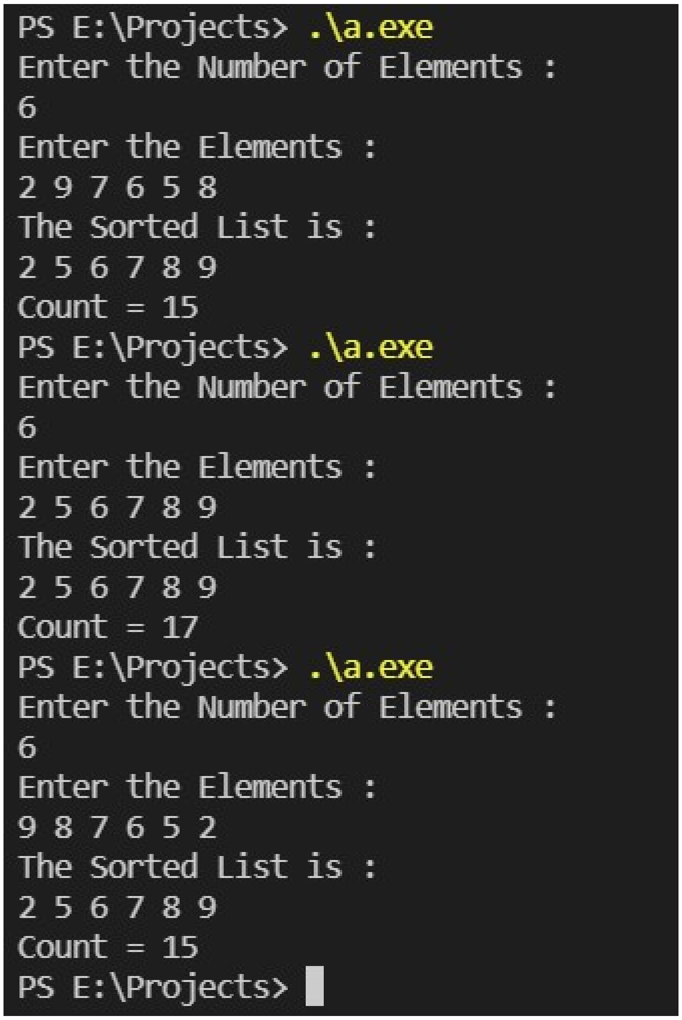
n	opcount
1	1
2	3
3	5
4	8
5	11
6	14
7	17
8	21
9	25
10	29

Q2. Write a program to sort the list of integers using heap sort with bottom-up max heap construction and analyze its time efficiency. Prove experimentally that the worst case time complexity is O(n log n)

```
#include  <stdio.h>
#include <stdlib.h>
int op = 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;
            op++;
            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;
        op++;
        k++;
    }
}
void main()
{
    int arr[20], n;
    printf("Enter the Number of Elements : \n");
    scanf("%d", &n);
    printf("Enter the Elements : \n");
```

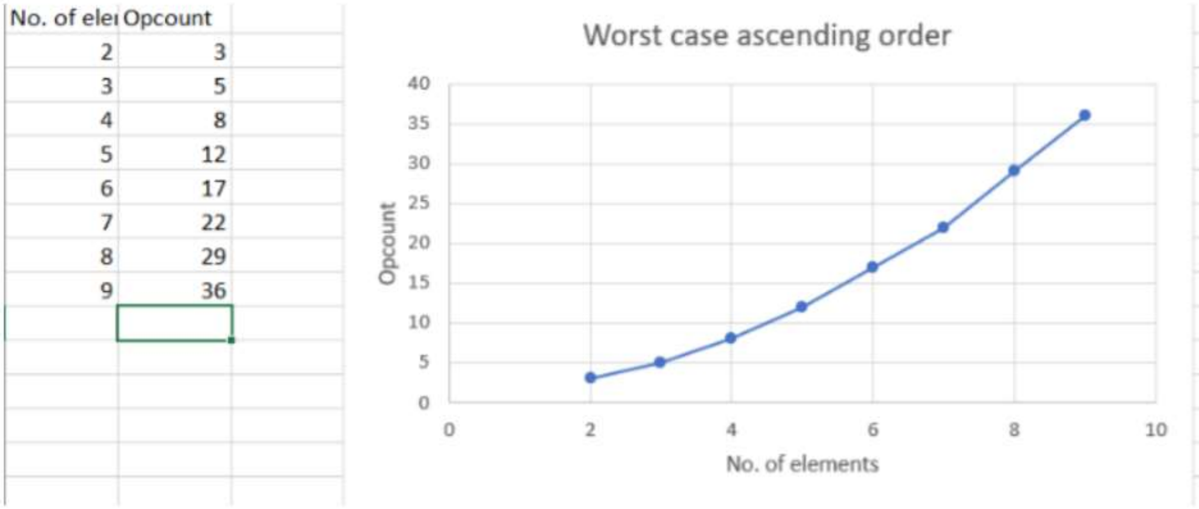
```
   for(int i = 1; i<=n; i++)
      scanf("%d", &arr[i]);
   HeapSort(arr, n);
   printf("The Sorted List is : \n");
   for(int i = 1; i<=n; i++)
      printf("%d ", arr[i]);
   printf("\n");
   printf("Count = %d\n", op);
}
```



```
PS E:\Projects> .\a.exe
Enter the Number of Elements :
6
Enter the Elements :
2 9 7 6 5 8
The Sorted List is :
2 5 6 7 8 9
Count = 15
PS E:\Projects> .\a.exe
Enter the Number of Elements :
6
Enter the Elements :
2 5 6 7 8 9
The Sorted List is :
2 5 6 7 8 9
Count = 17
PS E:\Projects> .\a.exe
Enter the Number of Elements :
6
Enter the Elements :
9 8 7 6 5 2
The Sorted List is :
2 5 6 7 8 9
Count = 15
PS E:\Projects>
```

Graph and analysis –

From the values we can see that for No. of elements n, the opcount is close to n*logn. The input is an array of ascending order. So we can see O(nlogn)for worst case.



No. of elei	Opcount
2	3
3	5
4	8
5	12
6	17
7	22
8	29
9	36