DAA Lab 8

1) 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.

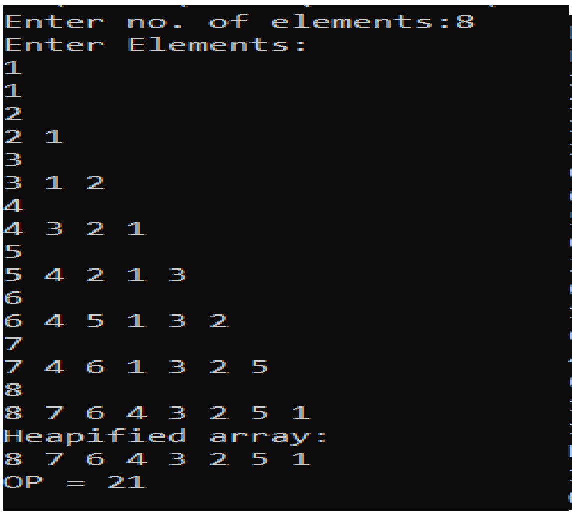
Code –

```
#include <stdio.h>
#include <stdlib.h>
int op = 0;
void heapify(int arr[], int currIndex)
{
   int parent = currIndex/2; //if parent is i, children are 2i and 2i+1
   op++;
   while(parent > 0) //heapification for each insertion
   {
          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 h[20], n;
   printf("Enter no. of elements:");
   scanf("%d", &n);
```

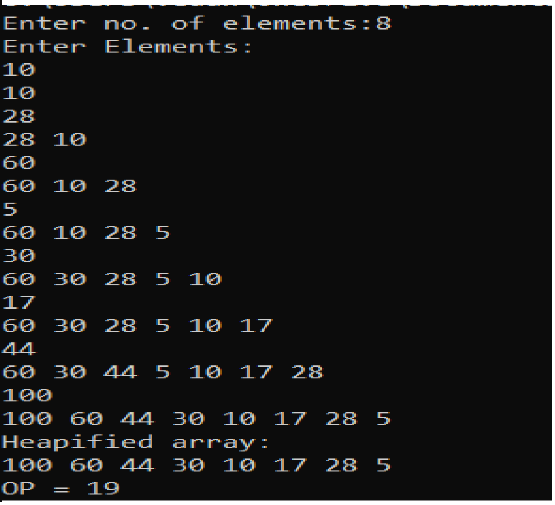
```
    printf("Enter Elements:\n");


    for(int i = 1; i<=n; i++)
    {
      scanf("%d", &h[i]);
      heapify(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;
  }
```

Execution –



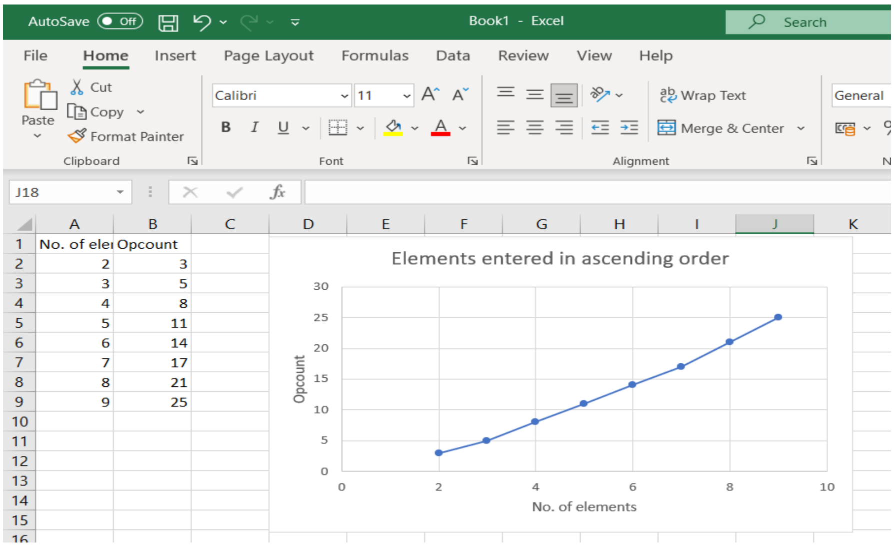
```
Enter no. of elements:8
Enter Elements:
1
1
2
2 1
3
3 1 2
4
4 3 2 1
5
5 4 2 1 3
6
6 4 5 1 3 2
7
7 4 6 1 3 2 5
8
8 7 6 4 3 2 5 1
Heapified array:
8 7 6 4 3 2 5 1
OP = 21
```



```
Enter no. of elements:8
Enter Elements:
10
10
28
28 10
60
60 10 28
5
60 10 28 5
30
60 30 28 5 10
17
60 30 28 5 10 17
44
60 30 44 5 10 17 28
100
100 60 44 30 10 17 28 5
Heapified array:
100 60 44 30 10 17 28 5
OP = 19
```

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.



	A	B
1	No. of ele	Opcount
2	2	3
3	3	5
4	4	8
5	5	11
6	6	14
7	7	17
8	8	21
9	9	25

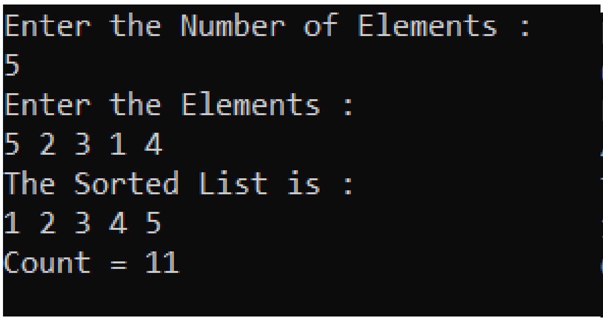
2) Write a program to sort the list of integers using heap sort with bottom up max heap construction and analyze its time efficiency. Prove experimentally that the worst case time complexity is O (n log n)

Code –

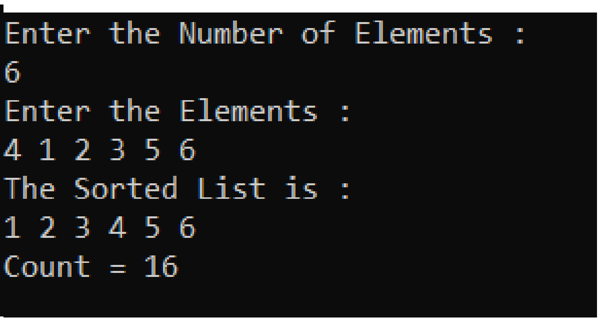
```
#include <stdio.h>
#include <stdlib.h>
int op = 0;
void heapify(int h[],int n)
{
  int i,k,v,heapify,j;
  for(i=(n/2);i>=1;i--)
  {
    k=i;v=h[k];heapify=0;
    while(heapify==0&&2*k<=n)
    {
      op++;
      j=2*k;
      if(j<n)
        if(h[j]<h[j+1])
          j=j+1;
      if(v>=h[j])
        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, n - k);
        int temp = arr[1];
        arr[1] = arr[n-k];
        arr[n-k] = temp;
        k++;
        op++;
    }
}
int 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);
    return 0;
}
```

Execution –



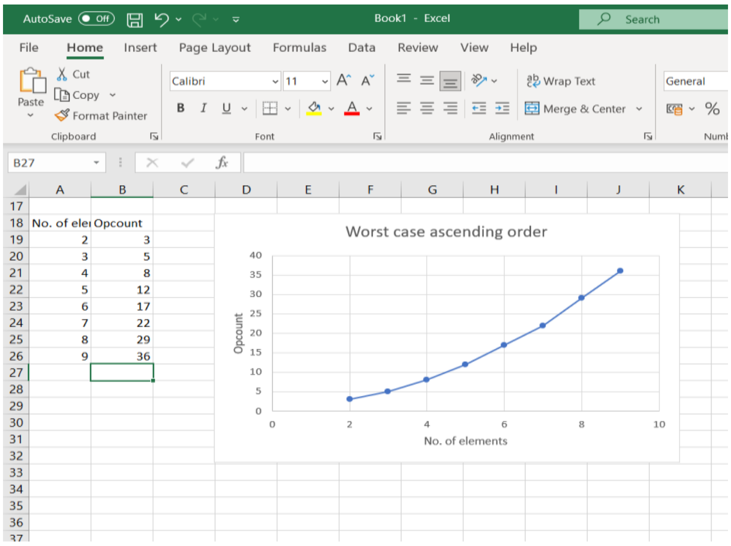
```
Enter the Number of Elements :
5
Enter the Elements :
5 2 3 1 4
The Sorted List is :
1 2 3 4 5
Count = 11
```



```
Enter the Number of Elements :
6
Enter the Elements :
4 1 2 3 5 6
The Sorted List is :
1 2 3 4 5 6
Count = 16
```

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 ele	Opcount
2	3
3	5
4	8
5	12
6	17
7	22
8	29
9	36