Lab Exercises:

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.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int opCount = 0;
void HeapTopDown(int nameArray[], int startingIndex)
{
   int parentalDominance = (startingIndex - 1) / 2;
   while (parentalDominance >= 0)
   {
      opCount++;
      if (nameArray[parentalDominance] < nameArray[startingIndex])
      {
         int temp = nameArray[parentalDominance];
         nameArray[parentalDominance] = nameArray[startingIndex];
         nameArray[startingIndex] = temp;

         startingIndex = parentalDominance;
         parentalDominance = (startingIndex - 1) / 2;
      }
      else
         return;
   }
}
void main()
{
   int heapArray[20];
   int size;
   int i;
   int j;
   printf("Enter the size of elements : ");
   scanf("%d", &size);
   printf("Enter the elements : ");
   for (i = 0; i < size; i++)
   {
      scanf("%d", &heapArray[i]);
      printf("\n");
      printf("The Array is getting in Heap : \n\n");
      HeapTopDown(heapArray, i);
      for (j = 0; j <= i; j++)
```

```
            printf("%d ", heapArray[j]);
        printf("\n");
    }
    printf("\n\n");
    printf("Orgnized Heap Array is : \n\n");
    for (i = 0; i < size; i++)
        printf("%d\t", heapArray[i]);
    printf("\n\n");
    printf("Opcount of HeapTopDown is = %d ", opCount);
    printf("\n\n\n");
    exit(0);
}
```

Analysis:

Stage 1: Build heap for a given list of n keys using top-down heap construction

worst-case

$$C(n) \in \Theta(n \log n)$$

Stage 2: Repeat operation of root removal n-1 times (fix heap)

worst-case

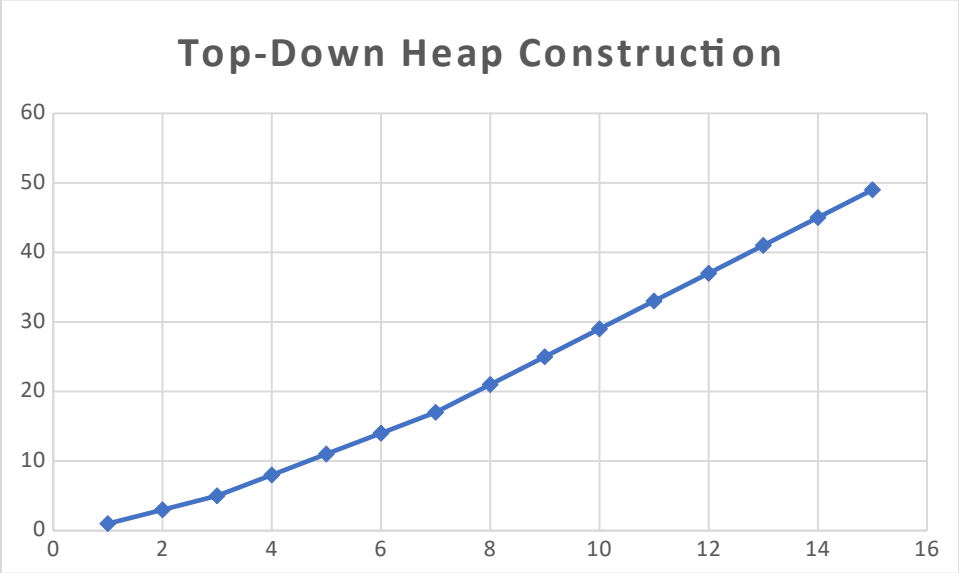
$$C(n) = \sum_{i=1}^{n-1} 2\log_2 i \in \Theta(n \log n)$$

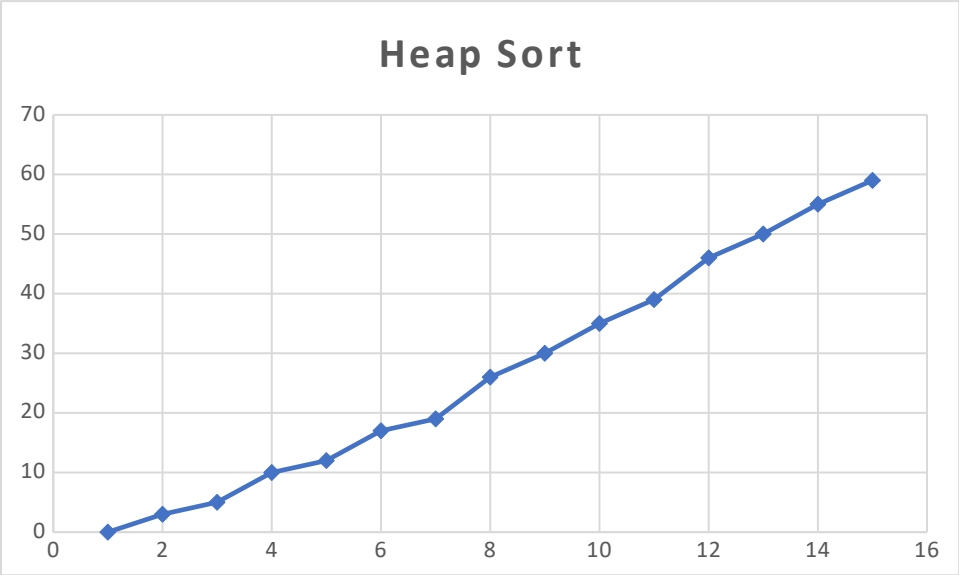
Both worst-case and average-case efficiency: $\Theta(n \log n)$

In-place: yes

Stability: no (e.g., 1 1)

Order of Growth:





Output:

```
Enter the size of elements : 8
Enter the elements : 5 77 8 63 1 1 99 0

The Array is getting in Heap :

5

The Array is getting in Heap :

77 5

The Array is getting in Heap :

77 5 8

The Array is getting in Heap :

77 63 8 5

The Array is getting in Heap :

77 63 8 5 1

The Array is getting in Heap :

77 63 8 5 1 1

The Array is getting in Heap :

99 63 77 5 1 1 8

The Array is getting in Heap :

99 63 77 5 1 1 8 0


Orgnized Heap Array is :

99      63      77      5       1       1       8       0

Opcount of HeapTopDown is = 12



Process returned 0 (0x0)   execution time : 26.094 s
Press any key to continue.
```

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)

Program:

```
#include <stdio.h>
#include <stdlib.h>
int opCount = 0;
void heapifyBottomUp(int heapArray[], int left, int size)
{
   int i;
   int k;
   int trace;
   int heapifyMenu;
   int j;
   for (i = (size / 2); i >= left; i--)
   {
      k = i;
      trace = heapArray[k];
      heapifyMenu = 0;
      while (heapifyMenu == 0 && 2 * k <= size)
      {
         j = 2 * k;
         opCount++;
         if (j < size)
            if (heapArray[j] < heapArray[j + 1])
               j = j + 1;
         if (trace >= heapArray[j])
            heapifyMenu = 1;
         else
         {
            heapArray[k] = heapArray[j];
            k = j;
         }
      }
      heapArray[k] = trace;
   }
   return;
}
void HeapSortUsingBottomUp(int heapArray[], int size)
{
   int j = 0;
   for (int i = 1; i <= size; i++)
   {
      heapifyBottomUp(heapArray, 1, size - j);
      int temp = heapArray[1];
      heapArray[1] = heapArray[size - j];
      heapArray[size - j] = temp;
      j++;
```

```
    }
}
void main()
{
    int heapArray[20];
    int size;
    int i;
    printf("Enter the size of Elemets : ");
    scanf("%d", &size);
    printf("\n\n");
    printf("Enter the Elements : ");
    for (i = 1; i <= size; i++)
        scanf("%d", &heapArray[i]);
    HeapSortUsingBottomUp(heapArray, size);
    printf("\n\n");
    printf("The Heap Sort Array is : \n");
    printf("\n");
    for (i = 1; i <= size; i++)
        printf("%d ", heapArray[i]);
    printf("\n\n");
    printf("The Opcount is = %d\n", opCount);
    printf("\n\n");
}
```

Analysis:

Stage 1: Build heap for a given list of n keys using bottom-up heap construction

worst-case

$$C(n) = \sum_{i=0}^{h-1} 2(h-i)2^i = 2\,(n - \log_2(n+1)) \in \Theta(n)$$

($2(h-i)$: # nodes at level i)

Stage 2: Repeat operation of root removal n-1 times (fix heap)

worst-case

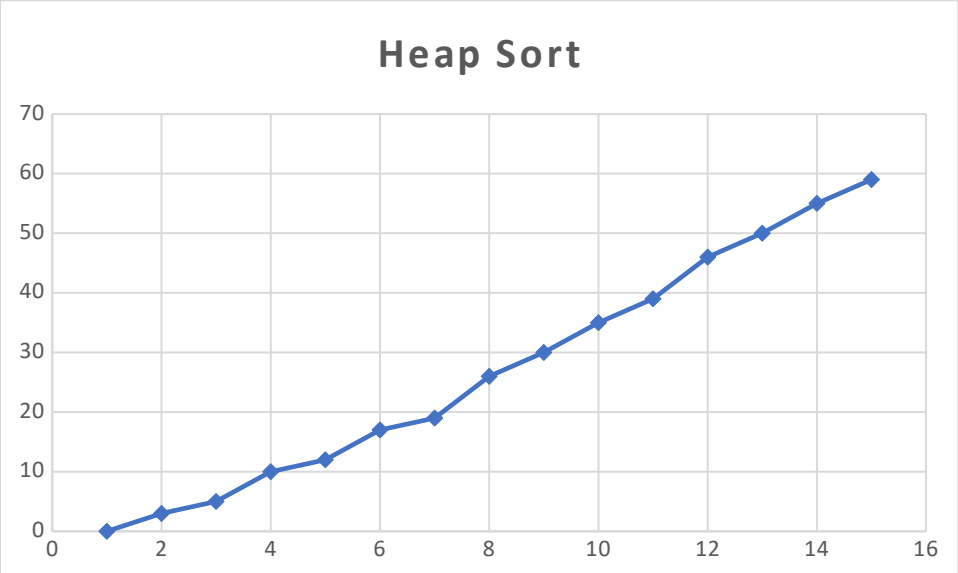
$$C(n) = \sum_{i=1}^{n-1} 2\log_2 i \in \Theta(n\log n)$$

Both worst-case and average-case efficiency: $\Theta(n\log n)$

In-place: yes

Stability: no (e.g., 1 1)

Order of Growth:



Output:

```
Enter the size of Elemets : 5


Enter the Elements : 5 6 1 3 4


The Heap Sort Array is :

1 3 4 5 6

The Opcount is = 8



Process returned 0 (0x0)   execution time : 39.367 s
Press any key to continue.
```