```
Input array: 10 64 45 9 15 1
Total element: 6

Round-1
10 64 45 9 15 1
C1. 10 > 64 False Do Nothing {arr[0] > arr[1]}
C2. 64 > 45 True Swap 64 and 45 {arr[1] > arr[2]}
10 45 64 9 15 1
C3. 64 > 9 True Swap 64 and 9 {arr[2] > arr[3]}
10 45 9 64 15 1
C4. 64 > 15 True Swap 64 and 15 {arr[3] > arr[4]}
10 45 9 15 64 1
C5. 64 > 1 True Swap 64 and 1 {arr[4] > arr[5]}
10 45 9 15 1 64 [Heaviest element is at the last]

Total Comparison: 5
Sorted elements: 1

#outer loop for rounds
for i range(len(arr)-1): 0,1,2,3,4
        #innner loop is for comparison
        for j in range(0, len(arr) - i - 1):
                i = 0 (Round-1), j = 0,1,2,3,4 (Comparison-5)
                i = 1 (Round-2), j = 0,1,2,3 (Comparison-4)
                i = 2 (Round-3), j = 0,1,2 (Comparison-3)
                i = 3 (Round-4), j = 0,1 (Comparison-2)
                i = 4 (Round-5), j = 0 (Comparison-1)

Round-2
10 45 9 15 1 64
C1. 10 > 45 False Do Nothing {arr[0] > arr[1]}
C2. 45 > 9 True Swap 45 and 9 {arr[1] > arr[2]}
10 9 45 15 1 64
C3. 45 > 15 True Swap 45 and 15 {arr[2] > arr[3]}
10 9 15 45 1 64
C4. 45 > 1 True Swap 45 and 1 {arr[3] > arr[4]}
10 9 15 1 45 64

Total Comparison: 4
Sorted elements: 2

Round-3
10 9 15 1 45 64
C1. 10 > 9 True Swap 10 and 9
9 10 15 1 45 64
C2. 10 > 15 False Do Nothing
C3. 15 > 1 True Swap 15 and 1
9 10 1 15 45 64

Total Comparison: 3
Sorted elements: 3

Round-4
9 10 1 15 45 64
C1. 9 > 10 False Do Nothing
C2. 10 > 1 True Swap 10 and 1
```

```
9 1 10 15 45 64

Total Comparison: 2
Sorted elements: 4

Round-5
9 1 10 15 45 64
C1. 9 > 1 True Swap 9 and 1
1 9 10 15 45 64

Total Comparison: 1
Sorted elements: 5 (first is lowest so all are sorted)

for n elements, we are going to have n - 1 rounds
Total elements: n
Round:r Comparison:n-r

Total elements: 6
Round:1 Comparison: 5 (6 - 1)
Round:2 Comparison: 4 (6 - 2)
Round:3 Comparison: 3 (6 - 3)
Round:4 Comparison: 2 (6 - 4)
Round:5 Comparison: 1 (6 - 5)
============================================================================
====================
```

```
def merge(arr, x, m ,r):
        #get the total elements in first part
        n1 = m - x + 1
        #get the total elements in second part
        n2 = r - m

        #create temp array
        L = [0]*n1
        R = [0]*n2

        #copy the data to temp array
        for i in range(0,n1):
                L[i] = arr[x + i]

        for i in range(0,n2):
                R[j] = arr[m + x + j]

        #create variable to be used for merging
        i,j,k = 0,0,x

        #code to merge data in sorted order
        while i < n1 and j < n2:
                if L[i] < R[j]:
                        arr[k] = L[i]
                        i+=1
                else:
                        arr[k] = R[j]
                        j+=1
                k+=1

        #code remaining data
        while i < n1:
                arr[k] = L[i]
                i+=1
                k+=1

        while j < n2:
                arr[k] = R[j]
                j+=1
                k+=1

def merge_sort(arr, x, r):
        if x < r:
                #code for middle index
                m = x + (r - x)/2
                merge_sort(arr,x,m)
                merge_sort(arr,m+1,r)
                merge(arr,x,m,r)

arr = [2,4,3,5,7,1]
merge_sort(arr, 0, len(arr) - 1)
for ele in arr:
        print(ele, end = ' ')
======================================================================
====================
Counting Sort
arr
4               2               2               8               3
3       1
0               1               2               3               4
5               6
Total elements: 7
Max = 8

count
0               1               3               5               6
6               6               6               7
0               1               2               3               4
5               6               7               8

1               2               2               3               3
4               8
0               1               2               3               4
5               6

Total elements: n
max: k
space complexity: O(n + k)

0       40
Time complexity: O(n)
```