Experiment 4

1) Heap Sort : Not stable, but can be made stable

$T(n) = O(n \log n)$

```
Heapsort(A) {
    BuildHeap(A)
    for i ← length(A) downto 2 {
        exchange A[1] ↔ A[i]
        heapsize ← heapsize - 1
        Heapify(A, 1)
    }

BuildHeap(A) {
    heapsize ← length(A)
    for i ← floor(length/2) downto 1
        Heapify(A, i)
    }

Heapify(A, i) {
    lef ← left(i)
    righ ← right(i)
    if (lef <= heapsize) and (A[lef] > A[i])
        largest ← lef
    else
        largest ← i
    if (righ <= heapsize) and (A[righ] > A[largest])
        largest ← righ
```

```
        if (largest != i) {
            exchange A[i] ↔ A[largest]
        Heapify (A, largest)
    }
}
```

2)
```
Quicksort (A, p, r) {
    if (p < r)
        q ← Partition (A, p, r)
        Quicksort (A, p, q)
        Quicksort (A, q+1, r)
    }
}
```

```
Partition (A, p, r) {
    pivot ← A[p]
    i ← p
    j = r
    while (i < j) {
        while (A[i] <= pivot)
        {    i++;   }
        while (A[j] > pivot)
        {   j--;   }
        if (i < j)
            exchange A[i] ↔ A[j]
    }
    exchange A[p] ↔ A[j]
    return j;
} // end.
```

$T(n) \Rightarrow$ Best case $O(n \log n)$

Worst case $O(n^2)$

3) Implementation of quick sort in Java

```
int [] array = {1, 11, 18, 9, 27, 4, 29, 19};
quickSort(arr, 0, 7);
```

↓

this func" is in java.util.Arrays package

4) Quick sort in C++ (STL)

```
int arr[7] = {1, 8, 9, 10, 19, 4, 27};
sort(arr, arr+7);
```

↓

```
#include<algorithm>
```