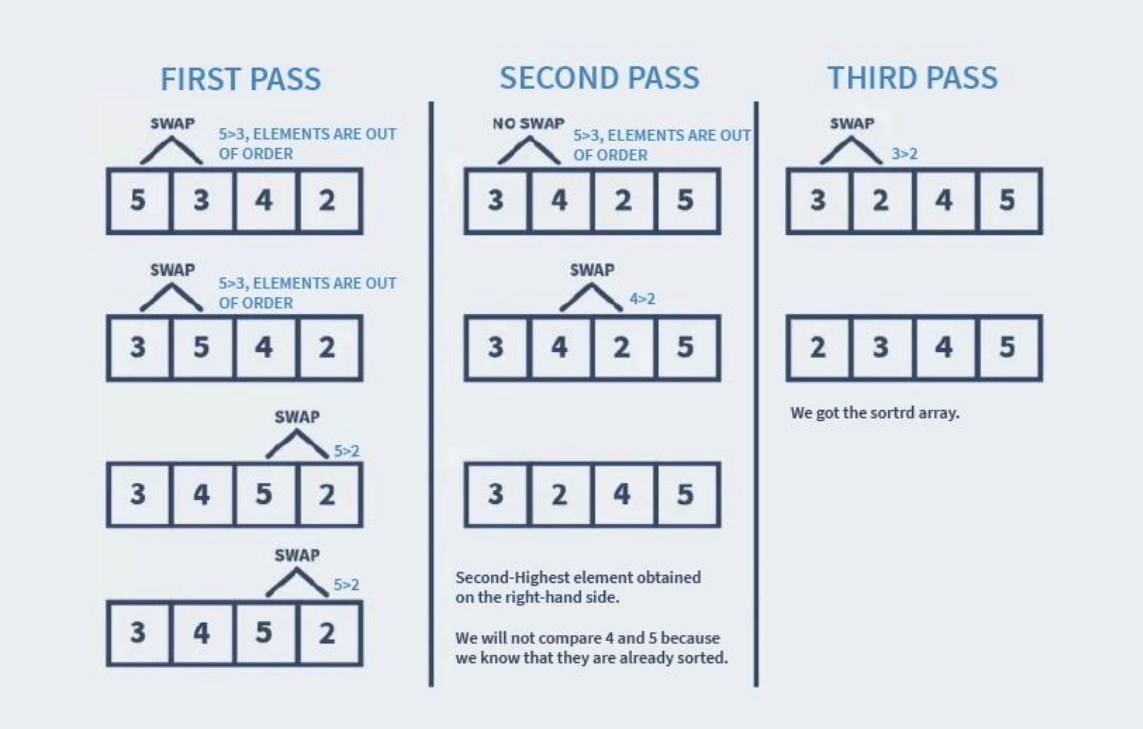
# Lab#04-Implementation of Sorting Algo (Array)

## Objective

Implement sorting algorithms (Array)

**Bubble Sort**



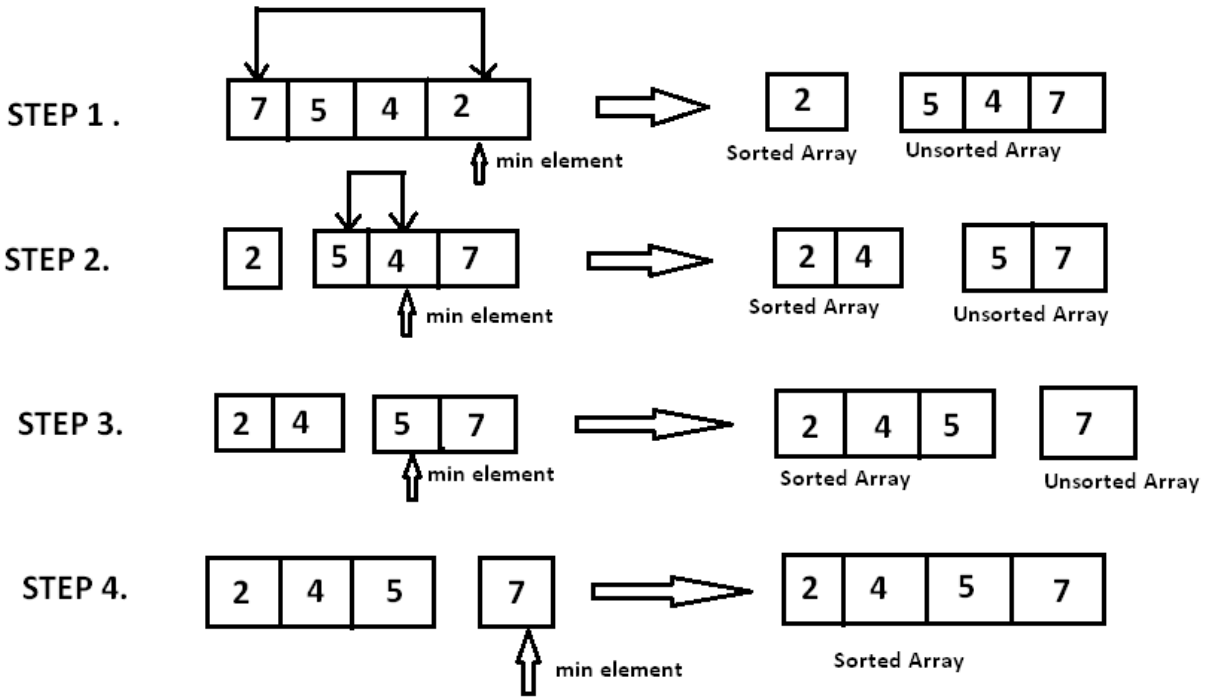

**Pseudo Code**

```
for i=0 to A.length-2

    for j=0 to A.length-2-i
                if(A[j]>A[j+1])
                    temp=A[j+1]
                    A[j+1]=A[j]
                    A[j]=temp
```

**Selection Sort**

**Happy coding** ☺




**Pseudo Code**

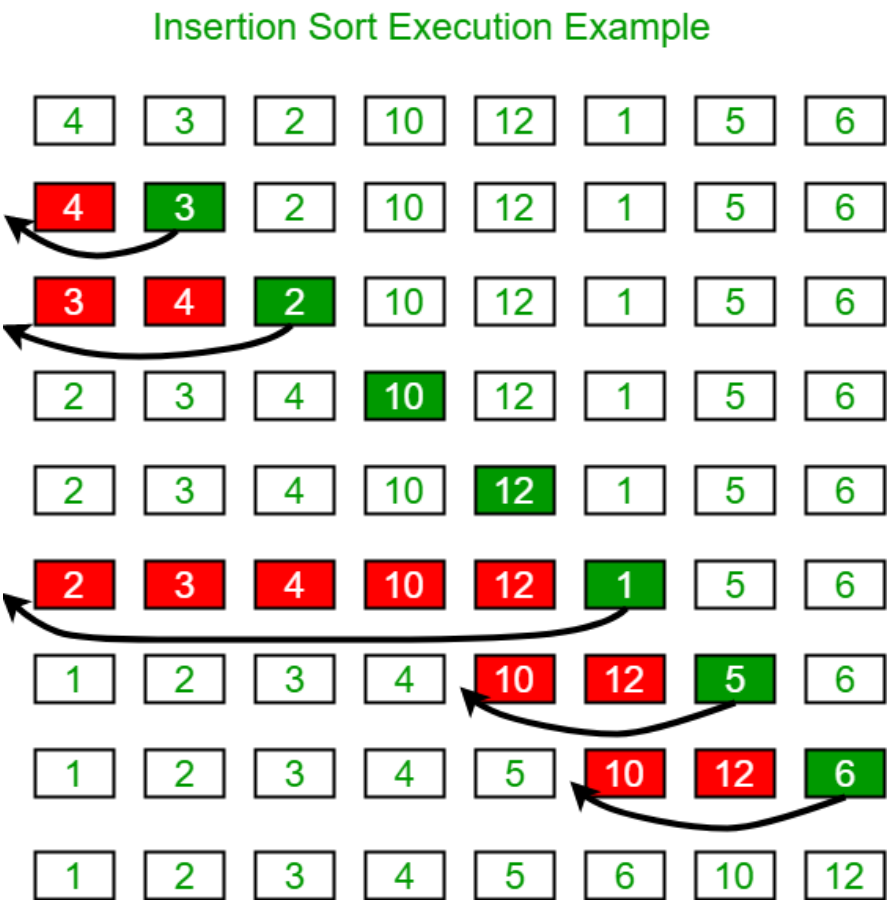
```
for i=0 to A.length-2
        mindIndex=i

        for j=i+1 to A.length-1
                if(A[j]<A[minIndex])
                        minIndex=j

        temp=A[i]
        A[i]=A[minIndex]
        A[minIndex]=temp
```

**Insertion Sort**

**Happy coding** ☺

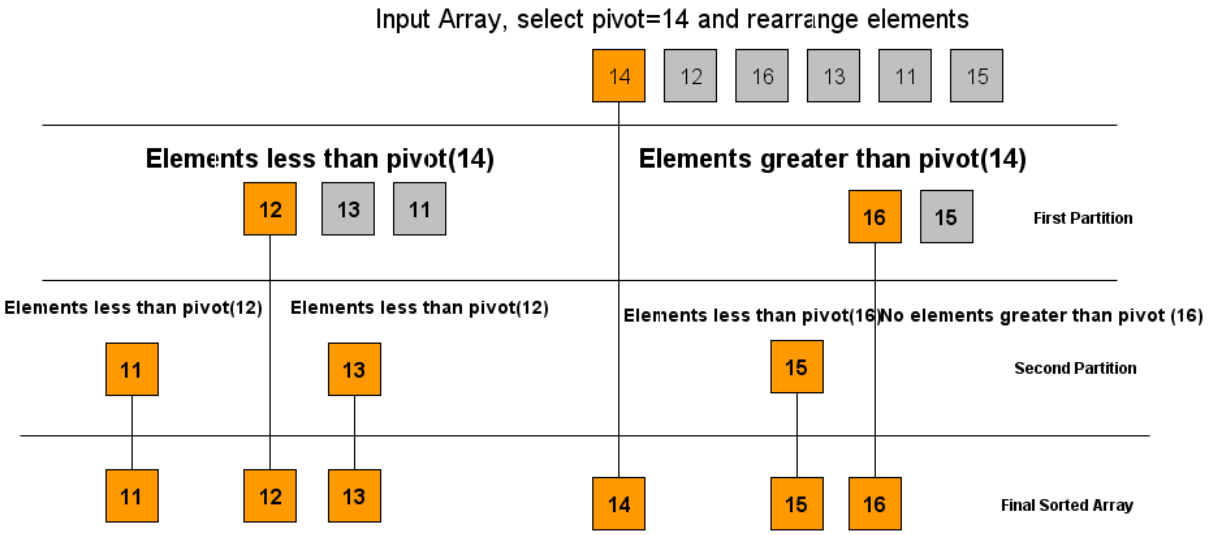



**Pseudo Code**

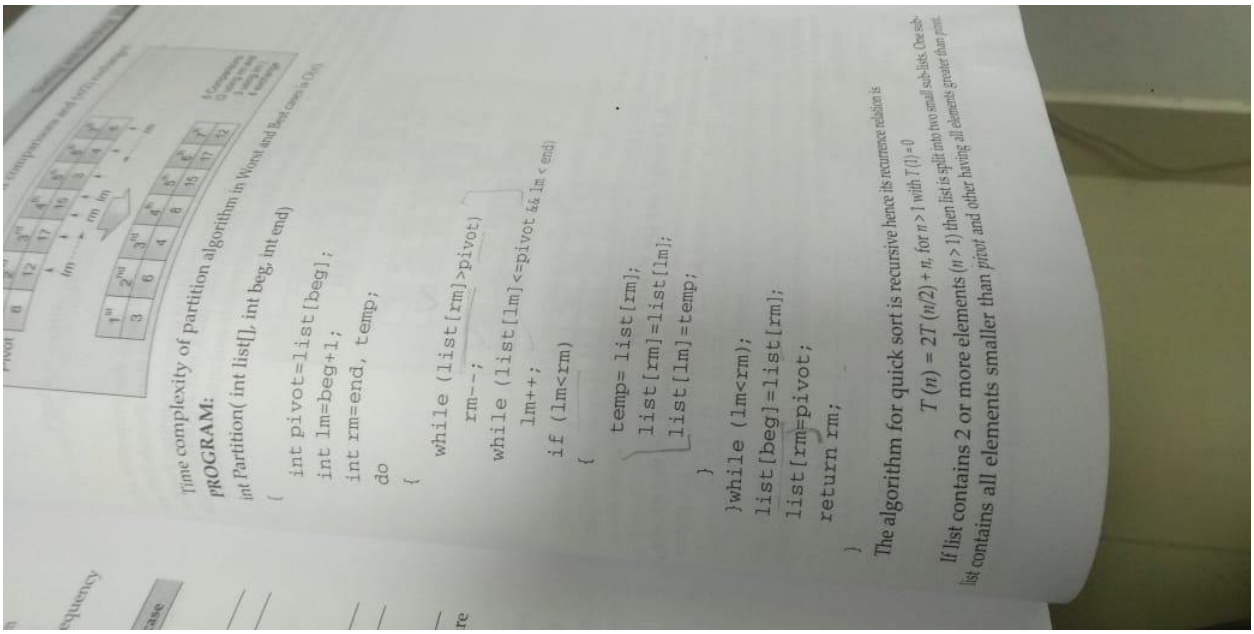
```
INSERTION-SORT(A)
  for i = 1 to n
          current ← A [i]
          j ← i – 1
           while j > = 0 and A[j] > current
                    A[j+1] ← A[j]
                    j ← j – 1
          End while
          A[j+1] ← current
  End for
```

**Quick Sort**

**Happy coding** ☺

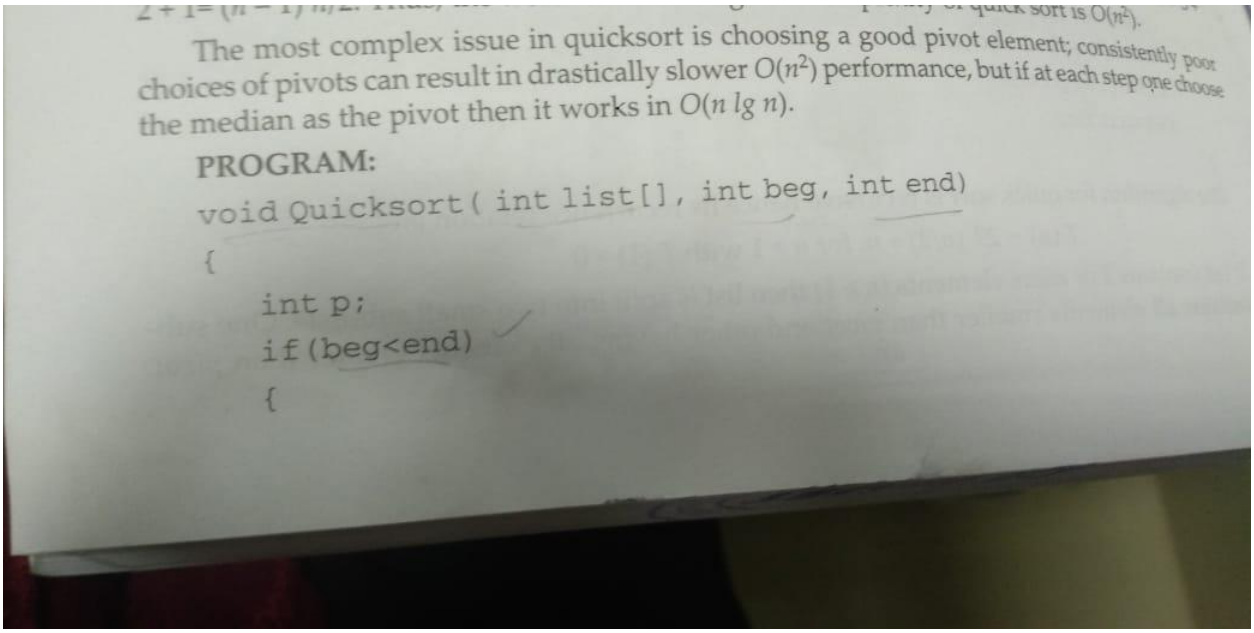



**Partition Alog**




**Quick Sort Algo**

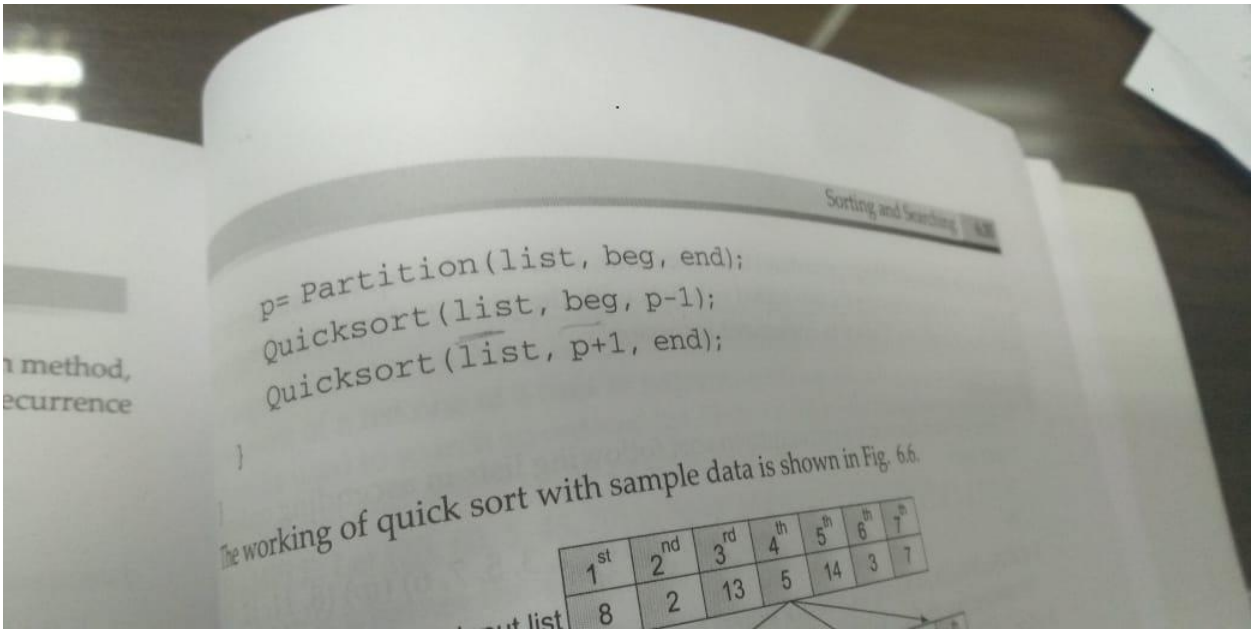
**Happy coding** ☺



The most complex issue in quicksort is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower $O(n^2)$ performance, but if at each step one choose the median as the pivot then it works in $O(n \lg n)$.

**PROGRAM:**

```
void Quicksort( int list[], int beg, int end)
{
    int p;
    if(beg<end)
    {
```



```
p= Partition(list, beg, end);
Quicksort(list, beg, p-1);
Quicksort(list, p+1, end);
}
```

The working of quick sort with sample data is shown in Fig. 6.6.

| 1st | 2nd | 3rd | 4th | 5th | 6th | 7th |
|---|---|---|---|---|---|---|
| 8 | 2 | 13 | 5 | 14 | 3 | 7 |

**How to calculate the execution time(milliseconds) of a method in java:**

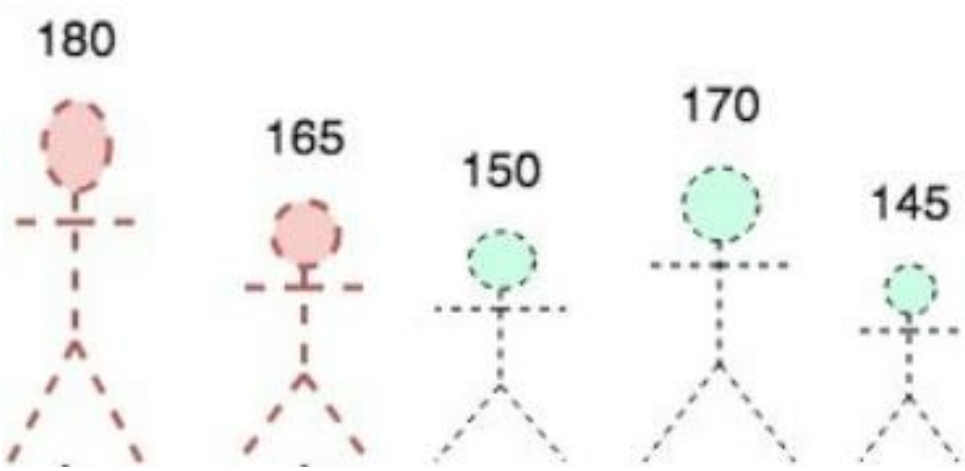
```
  long before1=System.currentTimeMillis();
  Searching_lab.greeting1();
  long after1 = System.currentTimeMillis();
  System.out.println("Execution time of greeting1 method is : "+(after1-
before1));
```

**Happy coding** ☺

```
  long before1=System.nanoTime();
  Searching_lab.greeting1();
  long after1 = System. System.nanoTime();
  System.out.println("Execution time of greeting1 method is : "+(after1-
before1));
```

**Exercises**

**Task#01**




Sort the given 05 students in ascending order of their heights

i. **Bubble sort**

   //Create method BubbleSort1D(int[] A)

ii. **Selection sort**

   //Create method SelectionSort1D(int[] A)

iii. **Insertion sort**

   //Create method InsertionSort1D(int[] A)

iv. **Quick sort**

   //Create method QuickSort1D(int[] A)

v. **Display the execution time of sorting algos and examine which one is the fastest and explain why?**

Sample Output:

**Happy coding** ☺

Time took by bubble sort : ____ (time in milliseconds/nanoseconds)

Time took by selection sort : ____ (time in milliseconds/nanoseconds)

Time took by insertion sort : ____ (time in milliseconds/nanoseconds)

Time took by quick sort : ____ (time in milliseconds/nanoseconds)

**Task#02**

Sort the following 2D array using (bubble, selection, insertion, and quick sort)

| | | |
|---|---|---|
| **2** | **10** | **15** |
| **5** | **1** | **3** |
| **6** | **9** | **4** |