<u>Assignment 10</u>

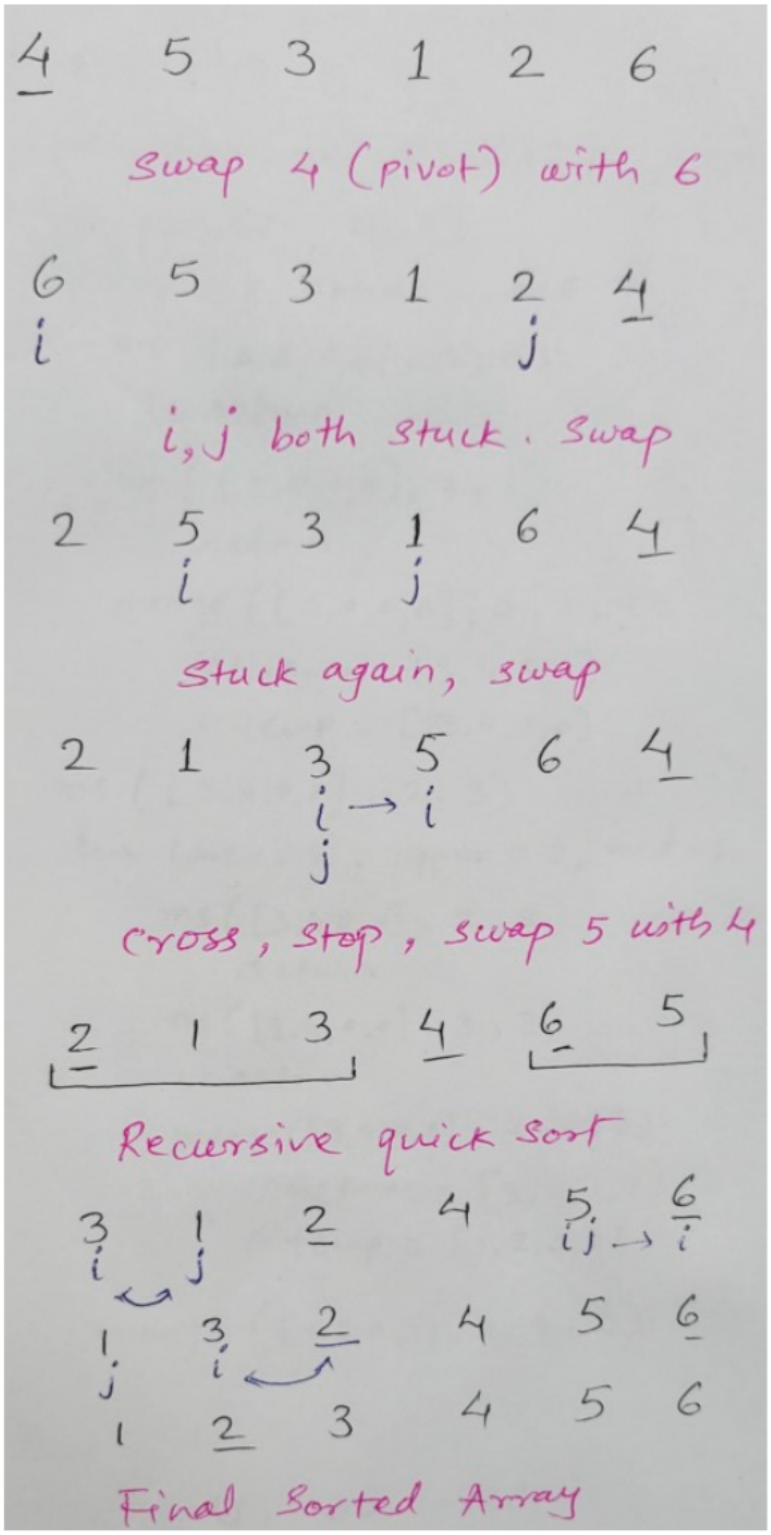
1. Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an *n*-element sequence as the pivot, we choose the element at rank (index) *n*/2, that is, an element in the middle of the sequence. What is the running time of this version of quick-sort on a sequence that is already sorted?

Answer: In the case of a sorted list, the pivot will always be the median. Therefore, the running time of quick-sort on a sorted list is O(n log n).

Example: Input List 1 2 3 4 5 6 7 8 9 10

Median of first, middle and last are 1, 5, 10. The pivot is 5.

2. Show all the steps of In-Place QuickSort in sorting the following array:
   [4, 5, 3, 1, 2, 6]
   Use leftmost values as pivots.



$\underline{4}\quad 5\quad 3\quad 1\quad 2\quad 6$

swap 4 (pivot) with 6

$\underset{i}{6}\quad 5\quad 3\quad 1\quad \underset{j}{2}\quad \underline{4}$

i, j both stuck. Swap

$2\quad \underset{i}{5}\quad 3\quad \underset{j}{1}\quad 6\quad \underline{4}$

Stuck again, swap

$2\quad 1\quad \underset{\substack{i \\ j}}{3} \xrightarrow{} \underset{i}{5}\quad 6\quad \underline{4}$

cross, stop, swap 5 with 4

$\underbrace{\underline{2}\quad 1\quad 3}\quad \underline{4}\quad \underbrace{\underline{6}\quad 5}$

Recursive quick sort

$\underset{i}{3}\quad \underset{j}{1}\quad \underline{2}\quad 4\quad \underset{i\,j}{5} \to \underset{i}{\underline{6}}$

$\underset{j}{1}\quad \underset{i}{3}\quad \underline{2}\quad 4\quad 5\quad \underline{6}$

$1\quad \underline{2}\quad 3\quad 4\quad 5\quad 6$

Final Sorted Array

3. Devise an algorithm to find the second smallest element for a given array.
   How many different ways can you think of to solve the problem?

   Write algorithm for at least one such ways and determine the running time of your algorithm.

   Strategy 1: find the second smallest using one for loop.

```
function secondSmallest(int[] arr) {
        int smallest = arr[0];
        int secondSmallest = arr[0];
        for (int i = 0; i < arr.length; i++) {
                if (arr[i] < smallest) {
                        secondSmallest = smallest;
                        smallest = arr[i];
                } else if (arr[i] < secondSmallest) {
                        secondSmallest = arr[i];
                }
        }
        return secondSmallest;
}
```

   Strategy 2: Sort the array, then return first element in sorted array.

   Strategy 3: Find the smallest, remove it from the array, again find and return the smallest.

   Strategy 4: Use partition algorithm introduced in QuickSort. (Use QuickSelect algorithm)