1.Given an array, arr[]. Sort the array using bubble sort algorithm.
Examples :
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: An array that is already sorted should remain unchanged after applying bubble sort.

Solution:

```
import java.io.*;
class Bubble{
    static void bubbleSort(int arr[], int n){
        int i, j, temp;
        boolean swapped;
        for (i = 0; i < n - 1; i++) {
            swapped = false;
            for (j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }
    static void printArray(int arr[], int size){
        int i;
        for (i = 0; i < size; i++)
            System.out.print(arr[i] + " ");
        System.out.println();}
    public static void main(String args[]){
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 };
        int n = arr.length;
        bubbleSort(arr, n);
        System.out.println("Sorted array: ");
        printArray(arr, n);}}
```

Output:
Sorted array:
11 12 22 25 34 64 90

Time Complexity:O(n^2)
Space Complexity:O(1)

2.Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, arr[] in ascending order. Given an array, arr[], with starting index low and ending index high, complete the functions partition() and quickSort(). Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.
Note: The low and high are inclusive.
Examples:
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

Solution:

```
import java.util.Arrays;
Class  Ouick {
   static int partition(int[] arr, int low, int high) {

       int pivot = arr[high];

       int i = low - 1;

       for (int j = low; j <= high - 1; j++) {
           if (arr[j] < pivot) {
               i++;
               swap(arr, i, j);
           }
       }

       swap(arr, i + 1, high);
       return i + 1;
   }

   static void swap(int[] arr, int i, int j) {
       int temp = arr[i];
       arr[i] = arr[j];
       arr[j] = temp;
   }

   static void quickSort(int[] arr, int low, int high) {
       if (low < high) {

           int pi = partition(arr, low, high);
           quickSort(arr, low, pi - 1);
           quickSort(arr, pi + 1, high);
       }
```

```
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        quickSort(arr, 0, n - 1);

        for (int val : arr) {
            System.out.print(val + " ");
        }
    }
}
```

Output:
Sorted Array
1 5 7 8 9 10

Time Complexity:O(n^2)
Space Complexity:O(n)

3.Given a string s of lowercase English letters, the task is to find the first non-repeating character. If there is no such character, return '$'.
Examples:
Input: s = "geeksforgeeks"
Output: 'f'
Explanation: 'f' is the first character in the string which does not repeat.

Solution:

```
import java.util.Arrays;
class Non_rep {

    static final int MAX_CHAR = 26;

    static char nonRepeatingChar(String s) {

        // Initialize frequency array
        int[] freq = new int[MAX_CHAR];
        for (char c : s.toCharArray())
            freq[c - 'a']++;

        for (int i = 0; i < s.length(); ++i) {
            if (freq[s.charAt(i) - 'a'] == 1)
                return s.charAt(i);}  return '$';}
```

```
public static void main(String[] args) {
        String s = "racecar";

        System.out.println(nonRepeatingChar(s));
    }
}
```

Output:
e
Time Complexity:O(n)
Space Complexity:O(max_char)

4.Given two strings s1 and s2 of lengths m and n respectively and below operations that can be performed on s1. Find the minimum number of edits (operations) to convert ‘s1‘ into ‘s2‘.
Insert: Insert any character before or after any index of s1
Remove: Remove a character of s1
Replace: Replace a character at any index of s1 with some other character.
Note: All of the above operations are of equal cost.
Examples:
Input: s1 = “geek”, s2 = “gesek”
Output: 1
Explanation: We can convert s1 into s2 by inserting a ‘s’ between two consecutive ‘e’ in s2.

Solution:

```
public class Distance {
    public static int editDist(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        int prev; // Stores dp[i-1][j-1]
        int[] curr = new int[n + 1]; // Stores dp[i][j-1] and dp[i][j]

        for (int j = 0; j <= n; j++)
            curr[j] = j;

        for (int i = 1; i <= m; i++) {
            prev = curr[0];
            curr[0] = i;
            for (int j = 1; j <= n; j++) {
                int temp = curr[j];
                if (s1.charAt(i - 1) == s2.charAt(j - 1))
                    curr[j] = prev;
                else
                    curr[j] = 1 + Math.min(curr[j - 1], Math.min(prev, curr[j]));
                prev = temp;}} return curr[n];}
```

```
    public static void main(String[] args) {
        String s1 = "GEEXSFRGEEKKS", s2 = "GEEKSFORGEEKS";
        System.out.println(editDist(s1, s2));
    }
}
```

Output:
3
Time Complexity:O(m*n)
Space Complexity:O(n)

5.Given an array arr[] and an integer k, the task is to find k largest elements in the given array. Elements in the output array should be in decreasing order.
Examples:
Input: [1, 23, 12, 9, 30, 2, 50], K = 3
Output: [50, 30, 23]

Solution:

```
import java.util.*;
public class KLargest {
    public static List<Integer> kLargest(int[] arr, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>(k);
        for (int i = 0; i < k; i++) {
            minHeap.add(arr[i]);
        }
        for (int i = k; i < arr.length; i++) {
            if (arr[i] > minHeap.peek()) {
                minHeap.poll();
                minHeap.add(arr[i]);}}
        List<Integer> res = new ArrayList<>();
        while (!minHeap.isEmpty()) {
            res.add(minHeap.poll());
}
        Collections.reverse(res);
        return res;}
    public static void main(String[] args) {
        int[] arr = {1, 23, 12, 9, 30, 2, 50};
        int k = 3;

        List<Integer> res = kLargest(arr, k);
        for (int ele : res) {
            System.out.print(ele + " ");
        }
    }}
```

Output:
50 30 23
Time Complexity:O(n*log(k))
Space Complexity:O(k)

6.Given an array of integers arr[] representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the largest possible number. Since the result may be very large, return it as a string.
Examples:
Input: arr[] = [3, 30, 34, 5, 9]
Output: "9534330"
Explanation: Given numbers are {3, 30, 34, 5, 9}, the arrangement "9534330" gives the largest value.

Solution:

```
class Solution {
    public String largestNumber(int[] nums) {
         String[] numStrs = new String[nums.length];
        for (int i = 0; i < nums.length; i++) {
            numStrs[i] = String.valueOf(nums[i]);
        }
        Arrays.sort(numStrs, (a, b) -> (b + a).compareTo(a + b));
        if (numStrs[0].equals("0")) {
            return "0";
        }

        StringBuilder result = new StringBuilder();
        for (String numStr : numStrs) {
            result.append(numStr);
        }

        return result.toString();

      public static void main(String args[]){
        int[] a={3,30,34,5,9};
        System.out.println(largestNumber(a));
    }
    }
}
```

Output:
9534330

Time Complexity:O(N*k logn)
Space Complexity:O(n*k)