**Problem solving :-**

**Sorting techniques :-**

**Selection sort**

class Solution {

    public int[] selectionSort(int[] nums) {

        int n = nums.length;

        for (int i = 0; i < n - 1; i++) {

            // Assume the current index is the minimum

            int minIndex = i;

            // Find the index of the actual minimum in the remaining array

            for (int j = i + 1; j < n; j++) {

                if (nums[j] < nums[minIndex]) {

                    minIndex = j;

                }

            }

            // Swap the found minimum with the current element

            int temp = nums[i];

            nums[i] = nums[minIndex];

            nums[minIndex] = temp;

        }

        return nums;

    }

}

**Bubble sort :-**

class Solution {

    public int[] bubbleSort(int[] nums) {

        int n = nums.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - 1 - i; j++) {

                if (nums[j] > nums[j + 1]) {

                    // Swap adjacent elements

                    int temp = nums[j];

                    nums[j] = nums[j + 1];

                    nums[j + 1] = temp;

                }

            }

        }

        return nums;

    }

}

…

Rotating array by one postion left

class Solution {

    public void rotateArrayByOne(int[] nums) {

        int n=nums.length;

        int[] temp=new int[nums.length];

        for(int i=1;i<n;i++){

            temp[(i-1)%n]=nums[i];

        }

        temp[n-1]=nums[0];

        for(int i=0;i<n;i++){

            nums[i]=temp[i];

        }

    }

}

Rotating an array by k postion to right

class Solution {

    public void rotate(int[] nums, int k) {

        int n=nums.length;

        int[] temp=new int[n];

       for(int i=0;i<n;i++){

        temp[(i+k)%n]=nums[i];

       }

       for(int i=0;i<n;i++){

         nums[i]=temp[i];

       }

    }

}

class Solution {

public int longestSubarray(int[] nums, int k) {

int maxLen = 0;

for (int i = 0; i < nums.length; i++) {

int sum = 0;

for (int j = i; j < nums.length; j++) {

sum += nums[j];

if (sum == k) {

maxLen = Math.max(maxLen, j - i + 1);

}

}

}

return maxLen;

}

}

class Solution {

public int singleNumber(int[] nums) {

int result = 0;

for (int num : nums) {

result ^= num; // XOR each number

}

return result;

}

}

import java.util.\*;

class Solution {

public int[] unionArray(int[] nums1, int[] nums2) {

int n = nums1.length;

int m = nums2.length;

Set<Integer> st = new TreeSet<>(); // TreeSet keeps elements sorted

for (int i = 0; i < n; i++) {

st.add(nums1[i]);

}

for (int j = 0; j < m; j++) {

st.add(nums2[j]);

}

int[] arr = new int[st.size()];

int i = 0;

for (int val : st) {

arr[i++] = val;

}

return arr;

}

}

class Solution {

    public String removeOuterParentheses(String s) {

        StringBuilder sb=new StringBuilder();

            char[] ca=s.toCharArray();

            int count=0;

           for(char c:ca) {

                  if(c=='('){

                    if(count>0){

                         sb.append("(");

                    }

                    count++;

                  }

                  else if(c==')'){

                      count--;

                    if(count>0){

                        sb.append(")");

                    }

                  }

           }

           return sb.toString();

    }

}

Reversing a number

class Solution {

    public int reverse(int x) {

        /\*

        int rev = 0;

        while (x != 0) {

            int dig = x % 10;

            // Check for overflow before actually multiplying

            if (rev > Integer.MAX\_VALUE / 10 || (rev == Integer.MAX\_VALUE / 10 && dig > 7)) {

                return 0; // Overflow case

            }

            if (rev < Integer.MIN\_VALUE / 10 || (rev == Integer.MIN\_VALUE / 10 && dig < -8)) {

                return 0; // Underflow case

            }

            rev = rev \* 10 + dig;

            x = x / 10;

        }

        return rev;

        \*/

       String str = Integer.toString(Math.abs(x));

        String sb = new StringBuilder(str).reverse().toString();

        int num=Integer.parseInt(sb);

         return num;

    }

}

Palindrome

class Solution {

    public boolean isPalindrome(int x) {

        int num=x;

        if(x<0){

            return false;

        }

         int rev=0;

         while(x!=0){

           int dig=x%10;

           rev=rev\*10+dig;

           x=x/10;

         }

    if(num==rev){

        return true;

    }

    return false;

        }}

class Solution {

    public boolean isPalindrome(int x) {

        int num=x;

        if(x<0){

            return false;

        }

         int rev=0;

         while(x!=0){

           int dig=x%10;

           rev=rev\*10+dig;

           x=x/10;

         }

    if(num==rev){

        return true;

    }

    return false;

    }

}

class Solution {

    public int reverse(int x) {

        /\*

        int rev = 0;

        while (x != 0) {

            int dig = x % 10;

            // Check for overflow before actually multiplying

            if (rev > Integer.MAX\_VALUE / 10 || (rev == Integer.MAX\_VALUE / 10 && dig > 7)) {

                return 0; // Overflow case

            }

            if (rev < Integer.MIN\_VALUE / 10 || (rev == Integer.MIN\_VALUE / 10 && dig < -8)) {

                return 0; // Underflow case

            }

            rev = rev \* 10 + dig;

            x = x / 10;

        }

        return rev;

        \*/

       String str = Integer.toString(Math.abs(x));

        String sb = new StringBuilder(str).reverse().toString();

        int num=Integer.parseInt(sb);

         return num;

    }

}

class Solution {

    public boolean isAnagram(String s, String t) {

       char[] ss=s.toCharArray();

       char[] tt=t.toCharArray();

      Arrays.sort(ss);

      Arrays.sort(tt);

       if(Arrays.equals(ss,tt)){

        return true;

       }

       return false;

    }

}