1)Remove duplicates from Sorted Array

```
import java.io.*;
class Removedup{
                static int removedupsort(int[] arr){
                        int n=arr.length;
                        int j=1;
                        if(n<=1) return n;
                        for(int i=1; i<n;i++){
                                if(arr[i] !=arr[i-1]){
                                        arr[j]=arr[i];
                                        j++;
                                }
                        }
                        return j;
                }
                public static void main(String[] args) {
                        int[] arr = {1, 2, 2, 3, 4, 4, 4, 5, 5};
                        int result =removedupsort (arr);
                        for (int i = 0; i <result; i++) {
                                System.out.print(arr[i] + " ");
        }
    }
}
```

Time complexity:O(n)

Space complexity:O(1)

```
Output
1 2 3 4 5
=== Code Execution Successful ===
```

2)Stock Buy and Sell – Max one Transaction Allowed

```
import java.util.ArrayList;
import java.util.List;
class MaxProfit1{
        static int maxProfit(int[] prices){
                int n=prices.length;
                int minsofar=prices[0];
                int result=0;
                for(int i=0;i<n;i++){
                        minsofar=Math.min(minsofar,prices[i]);
                        result=Math.max(result,prices[i]-minsofar);
                }
                return result;
        }
        public static void main(String[] args){
                int[] prices = {7, 10, 1, 3, 6, 9, 2};
                System.out.println(maxProfit(prices));
  }
}
```

Time complexity:O(n)

Space complexity:O(1)

```
Output
8

=== Code Execution Successful ===
```

3) Sort an array in wave form

```
import java.io.*;
class SortWave{
        void  swap(int[] arr, int a , int b){
                int temp=arr[a];
                arr[a]=arr[b];
                arr[b]=temp;
        }
        void  sortwave(int[] arr, int n){
                for(int i=0;i<n;i+=2){
                        if(i>0 && arr[i-1]>arr[i]){
                                swap(arr, i, i-1);
                        }
                        if( i<n-1 && arr[i+1] > arr[i]){
                                swap(arr , i, i+1);
                        }
                }
        }
        public static void main(String[]  args){
```

```
                Main ob=new Main();
                int arr[] = {10, 90, 49, 2, 1, 5, 23};
                int n = arr.length;
                ob.sortwave(arr, n);
                for(int i:arr){
                        System.out.print(i+" ");
                }
        }
}
```

Output

```
90 10 49 1 5 2 23
=== Code Execution Successful ===
```

Time complexity:O(n)

Space complexity:O(1)

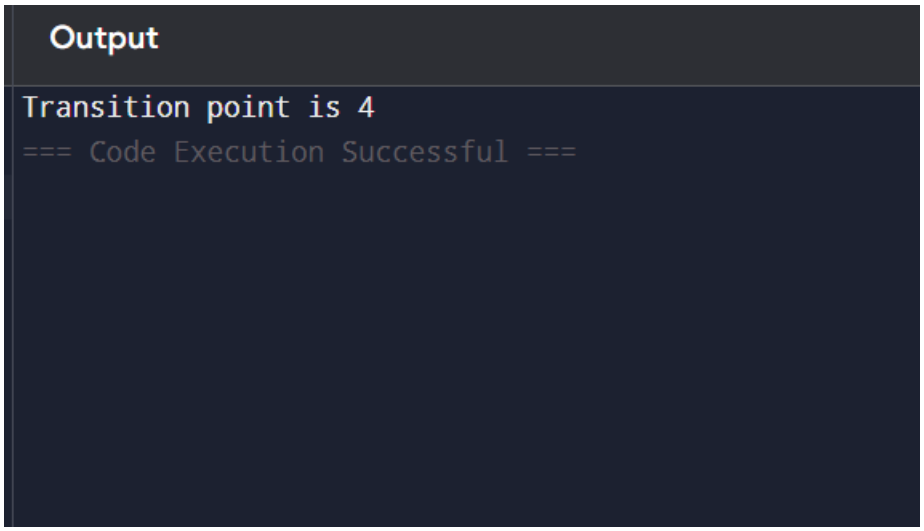
4) Find the transition point in a binary array

```
import java.util.*;
class Transition{
        static int transbinary(int[] arr, int n){
                for(int i=0; i<n; i++){
                        if(arr[i]==1){
                                return i;
                        }
                return -1;
        }
```

```
public static void main(String[] args){
        int arr[] = {0, 0, 0, 0, 1, 1};
        int n = arr.length;
        int point = findTransitionPoint(arr, n);
        if (point >= 0)
            System.out.print("Transition point is " + point);
        else
            System.out.print("There is no transition point");
}
}
```

Time complexity:O(n)

Space complexity:O(1)



5)Coin Change – Count Ways to Make Sum

```
import java.util.Arrays;
class CoinChange {
   static long count(int coins[], int n, int sum)
{
   int dp[] = new int[sum + 1];
   dp[0] = 1;
   for (int i = 0; i < n; i++)
```

```
        for (int j = coins[i]; j <= sum; j++)
            dp[j] += dp[j - coins[i]];
    return dp[sum];
}
    public static void main(String args[])
    {
        int coins[] = { 1, 2, 3 };
        int n = coins.length;
        int sum = 5;
        System.out.println(count(coins, n, sum));
    }
}
```

Output

```
5

=== Code Execution Successful ===
```

Time complexity : O(N*sum)
Auxiliary Space : O(N*sum)

6) Find first and last positions of an element in a sorted array

```
import java.io.*;
class GFG {
        public static void findFirstAndLast(int arr[], int x)
        {
                int n = arr.length;
                int first = -1, last = -1;
                for (int i = 0; i < n; i++) {
                        if (x != arr[i])
```

```
                continue;
            if (first == -1)
                first = i;
            last = i;
        }
        if (first != -1) {
            System.out.println("First Occurrence = "+ first);
            System.out.println("Last Occurrence = " + last);
        }
        else
            System.out.println("Not Found");
    }
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };
        int x = 8;
        findFirstAndLast(arr, x);
    }
}
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Output

```
First Occurrence = 8
Last Occurrence = 9

=== Code Execution Successful ===
```

7) MAXIMUM INDEX:

```
public class FindMaximum {
```

```
int max(int x, int y)
{
	return x > y ? x : y;
}

int min(int x, int y)
{
	return x < y ? x : y;
}
int maxIndexDiff(int arr[], int n)
{
	int maxDiff;
	int i, j;
	int RMax[] = new int[n];
	int LMin[] = new int[n];

	LMin[0] = arr[0];
	for (i = 1; i < n; ++i)
		LMin[i] = min(arr[i], LMin[i - 1]);
	RMax[n - 1] = arr[n - 1];
	for (j = n - 2; j >= 0; --j)
		RMax[j] = max(arr[j], RMax[j + 1]);
	i = 0;
	j = 0;
	maxDiff = -1;
	while (j < n && i < n) {
		if (LMin[i] <= RMax[j]) {
			maxDiff = max(maxDiff, j - i);
			j = j + 1;
		}
		else
```

```
                    i = i + 1;
            }
            return maxDiff;
        }
        public static void main(String[] args)
        {
            FindMaximum max = new FindMaximum();
            int arr[] = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };
            int n = arr.length;
            int maxDiff = max.maxIndexDiff(arr, n);
            System.out.println(maxDiff);
        }
}
```

Time Complexity: O(n)
Auxiliary Space: O(n)

```
Output
8

=== Code Execution Successful ===
```

8) Find first and last positions of an element in a sorted array

```
import java.io.*;
class firstandlast{
        public static void findFirstAndLast(int arr[], int x)
        {
            int n = arr.length;
            int first = -1, last = -1;
            for (int i = 0; i < n; i++) {
```

```
                if (x != arr[i])
                        continue;
                if (first == -1)
                        first = i;
                last = i;
        }
        if (first != -1) {
                System.out.println("First Occurrence = "
                                                + first);
                System.out.println("Last Occurrence = " + last);
        }
        else
                System.out.println("Not Found");
    }
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };
        int x = 8;
        findFirstAndLast(arr, x);
    }
}
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Output

```
First Occurrence = 8
Last Occurrence = 9

=== Code Execution Successful ===
```