DSA - November 14

1. Find Transition Point :

Solution :

```
class Solution {
    int transitionPoint(int arr[]) {
        // code here
        int n = arr.length;
        if(arr[0] == 1) return 0;

        int st = 0;
        int end = n-1;
        while(st <= end) {
            int mid = st + (end - st)/2;
            if(arr[mid] == 0) {
                st = mid+1;
            }
            else if(arr[mid] == 1) {
                if(arr[mid-1] == 0) {
                    return mid;
                }
                end = mid - 1;
            }
        }
        return -1;
    }
}
```

Time Complexity : O(log n)

Space Complexity : O(1)

2. Stock buy and sell :

Solution :

```
class Solution{
    //Function to find the days of buying and selling stock for max profit.
    ArrayList<ArrayList<Integer> > stockBuySell(int A[], int n) {
        // code here
        ArrayList<ArrayList<Integer>> list = new ArrayList<>();
        int i = 0;
        while(i< n-1) {
            while(i<n-1 && A[i] >= A[i+1]){
                i++;
            }
            if(i == n-1) break;
            int buy = i++;

            while(i<n && A[i] > A[i-1]){
                i++;
            }
            int sell = i - 1;

            ArrayList<Integer> res = new ArrayList<>();
            res.add(buy);
            res.add(sell);
            list.add(res);


        }
        return list;
    }
}
```

Time Complexity : O(n)

Space Complexity : O(n)

3. Maximum Index :

Solution :

```
class Solution {
    // Function to find the maximum index difference.
    int maxIndexDiff(int[] arr) {
        int n = arr.length;
        if (n == 1) {
            return 0;
        }
        int maxDiff = -1;
        int[] LMin = new int[n];
        int[] RMax = new int[n];

        LMin[0] = arr[0];
        for (int i = 1; i < n; ++i) LMin[i] = Math.min(arr[i], LMin[i - 1]);

        RMax[n - 1] = arr[n - 1];
        for (int j = n - 2; j >= 0; --j) RMax[j] = Math.max(arr[j], RMax[j + 1]);

        int i = 0, j = 0;
        while (i < n && j < n) {
            if (LMin[i] <= RMax[j]) {
                maxDiff = Math.max(maxDiff, j - i);
                j++;
            } else {
                i++;
            }
        }
        return maxDiff;
    }
}
```

Time Complexity : O(n)

Space Complexity : O(n)

4. First repeated Element :

Solution :

```
import java.util.HashSet;
```

```
public class Solution {
    public static int firstRepeated(int[] arr) {
        HashSet<Integer> set = new HashSet<>();
        int firstRepeatedIndex = -1;

        // Traverse from the end to find the first repeating element's 1-based position
        for (int i = arr.length - 1; i >= 0; i--) {
            if (set.contains(arr[i])) {
                firstRepeatedIndex = i + 1; // Convert 0-based index to 1-based position
            } else {
                set.add(arr[i]);
            }
        }

        return firstRepeatedIndex;
    }
}
```

Time Complexity : O(n)

Space Complexity : O(n)

5. Wave pattern :

Solution :

```
class Solution {
    public static void convertToWave(int[] arr) {
        // code here
        for(int i=0;i<=arr.length-2;i=i+2){
            swap(arr,i,i+1);
        }
    }

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

Time Complexity : O(n)

Space Complexity : O(1)

6. Find first and last occurrence :

Solution :

```
import java.util.ArrayList;

class GFG {
    ArrayList<Integer> find(int arr[], int x) {
        ArrayList<Integer> result = new ArrayList<>();
        int firstOccurrence = findFirst(arr, x);
        int lastOccurrence = findLast(arr, x);

        result.add(firstOccurrence);
        result.add(lastOccurrence);

        return result;
    }

    // Helper function to find the first occurrence of x
    int findFirst(int[] arr, int x) {
        int start = 0;
        int end = arr.length - 1;
        int firstOccurrence = -1;

        while (start <= end) {
            int mid = start + (end - start) / 2;

            if (arr[mid] == x) {
                firstOccurrence = mid;
                end = mid - 1;  // Continue searching in the left half
            } else if (arr[mid] < x) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }

        return firstOccurrence;
    }

    // Helper function to find the last occurrence of x
    int findLast(int[] arr, int x) {
        int start = 0;
        int end = arr.length - 1;
        int lastOccurrence = -1;

        while (start <= end) {
            int mid = start + (end - start) / 2;

            if (arr[mid] == x) {
                lastOccurrence = mid;
                start = mid + 1;  // Continue searching in the right half
            } else if (arr[mid] < x) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }

        return lastOccurrence;
    }
}
```

Time Complexity : O(log n)

Space Complexity : O(1)

7. Remove Sorted Array :

Solution :

```
class Solution {
    // Function to remove duplicates from the given array
    public int remove_duplicate(List<Integer> arr) {
        // Code Here
        int i = 0;
        int n = arr.size();

        for(int j=1;j<n;j++) {
            if(!arr.get(j).equals(arr.get(i))){
                i++;
                arr.set(i,arr.get(j));
            }
        }

        return i+1;
    }
}
```

Time Complexity : O(n)

Space Complexity : O(1)

8. Coin Change

Solution :

```
//Back-end complete function Template for Java

class Solution {
    // Function to count the number of ways to make a sum using given coins
    public int count(int coins[], int sum) {
        int N = coins.length;
        // Create a table to store the number of ways to make each sum from 0 to 'sum'
        int table[] = new int[sum + 1];

        // Initialize the table with 0
        for (int i = 0; i < sum + 1; i++) table[i] = 0;

        // There is always 1 way to make a sum of 0, so set table[0] to 1
        table[0] = 1;

        // Calculate the number of ways to make each sum from 0 to 'sum'
        for (int i = 0; i < N; i++)
            for (int j = coins[i]; j <= sum; j++) table[j] += table[j - coins[i]];

        // Return the number of ways to make the desired sum
        return table[sum];
    }
}
```

Time Complexity : O((sum+1)*N) ~ O((sum)*N)

Space Complexity : O((sum+1)) ~ O((sum))