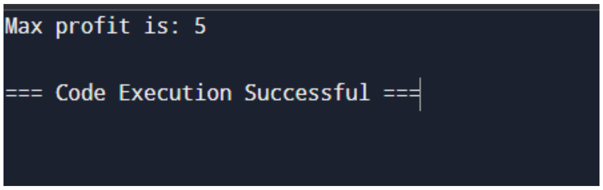
1.Stock Buy and Sell

Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Example 2:
Input: prices = [7,6,4,3,1]
Output: 0

Code:

```
import java.util.*;

public class Main {

    public static void main(String[] args) {

        int arr[] = {7, 1, 5, 3, 6, 4};

        int maxPro = maxProfit(arr);

        System.out.println("Max profit is: " + maxPro);

}

    static int maxProfit(int[] arr) {

        int maxPro = 0;

        int minPrice = Integer.MAX_VALUE;

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

            minPrice = Math.min(minPrice, arr[i]);

            maxPro = Math.max(maxPro, arr[i] - minPrice);

        }

        return maxPro;

       }

 }
```

OUTPUT:



```
Max profit is: 5

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

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(1)

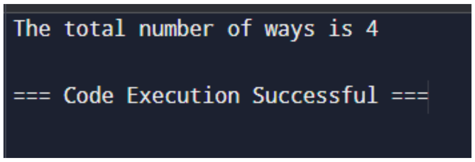
2.Coin Change(Count Ways):

Input: coins[] = [1, 2, 3], sum = 4
Output: 4
Input: coins[] = [5, 10], sum = 3
Output: 0

Code:

```
import java.util.*;
class Main {
    static long countWaysToMakeChange(int[] arr, int n, int T) {
        long[] prev = new long[T + 1];
        for (int i = 0; i <= T; i++) {
            if (i % arr[0] == 0)
                prev[i] = 1;
        }
        for (int ind = 1; ind < n; ind++) {
            long[] cur = new long[T + 1];
            for (int target = 0; target <= T; target++) {
                long notTaken = prev[target];
                long taken = 0;
                if (arr[ind] <= target)
                    taken = prev[target - arr[ind]]; // Fixed reference to prev instead of cur
                cur[target] = notTaken + taken;
            }
            prev = cur;
        }
        return prev[T];
    }
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3 };
        int target = 4;
        int n = arr.length;
        System.out.println("The total number of ways is " + countWaysToMakeChange(arr, n, target));
    }
}
```

OUTPUT:



```
The total number of ways is 4

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

TIME COMPLEXITY: O(N*T)

SPACE COMPLEXITY: O(T)

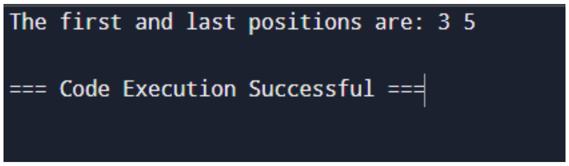
3.First and Last Occurrences:

Example 1:
Input Format: n = 8, arr[] = {2, 4, 6, 8, 8, 8, 11, 13}, k = 8
Result: 3 5
Example 2:
Input Format: n = 8, arr[] = {2, 4, 6, 8, 8, 8, 11, 13}, k = 10
Result: -1 -1

Code:

```
import java.util.*;
public class Main {
    public static int firstOccurrence(ArrayList<Integer> arr, int n, int k) {
        int low = 0, high = n - 1;
        int first = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr.get(mid) == k) {
                first = mid;
                high = mid - 1; // Move left to find earlier occurrences
            } else if (arr.get(mid) < k) {
                low = mid + 1; // Search right half
            } else {
                high = mid - 1; // Search left half
            }
        }
        return first;
    }
    public static int lastOccurrence(ArrayList<Integer> arr, int n, int k) {
        int low = 0, high = n - 1;
        int last = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr.get(mid) == k) {
                last = mid;
                low = mid + 1; // Move right to find later occurrences
            } else if (arr.get(mid) < k) {
                low = mid + 1; // Search right half
            } else {
                high = mid - 1; // Search left half
            }
        }
        return last;
    }

    public static int[] firstAndLastPosition(ArrayList<Integer> arr, int n, int k) {
        int first = firstOccurrence(arr, n, k);
        if (first == -1) return new int[] {-1, -1}; // If not found
        int last = lastOccurrence(arr, n, k);
        return new int[] {first, last};
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr = new ArrayList<>(Arrays.asList(2, 4, 6, 8, 8, 8, 11, 13));
        int n = arr.size(), k = 8;
        int[] ans = firstAndLastPosition(arr, n, k);
        System.out.println("The first and last positions are: " + ans[0] + " " + ans[1]);
    }
}
```

OUTPUT:



```
The first and last positions are: 3 5

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

TIME COMPLEXITY: O(2*logN)

SPACE COMPLEXITY:O(1)

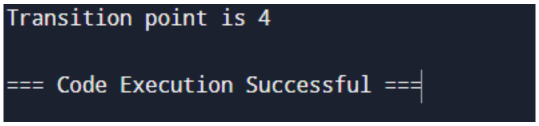
4. Find Transition Point:

Input: 0 0 0 1 1
Output: 3
Explanation: Index of first 1 is 3
Input: 0 0 0 0 1 1 1 1
Output: 4
Explanation: Index of first 1 is 4

CODE:

```
class Main {
    static int findTransitionPoint(int arr[], int n) {
        int lb = 0, ub = n - 1;
        while (lb <= ub) {
            int mid = (lb + ub) / 2;
            if (arr[mid] == 0)
                lb = mid + 1;
            else if (arr[mid] == 1) {
                if (mid == 0 || arr[mid - 1] == 0)
                    return mid;
                ub = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String args[]) {
        int arr[] = {0, 0, 0, 0, 1, 1};
        int point = findTransitionPoint(arr, arr.length);
        System.out.println(point >= 0 ? "Transition point is " + point : "There is no transition point");
    }
}
```

OUTPUT:



```
Transition point is 4

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

TIME COMPLEXITY: O(log n)

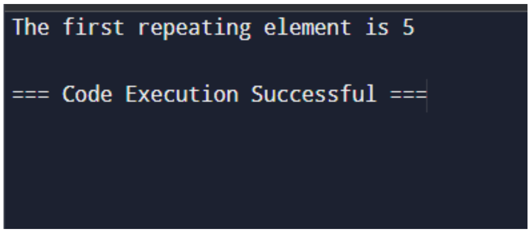
SPACE COMPLEXITY: O(1)

5. First Repeating Element:

Input: arr[] = {10, 5, 3, 4, 3, 5, 6}
Output: 5
Explanation: 5 is the first element that repeats
Input: arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10}
Output: 6
Explanation: 6 is the first element that repeats
CODE:

```
import java.util.*;
class Main {
    static void printFirstRepeating(int arr[]) {
        int min = -1;
        HashSet<Integer> set = new HashSet<>();
        for (int i = arr.length - 1; i >= 0; i--) {
            if (set.contains(arr[i]))
                min = i;
            else
                set.add(arr[i]);
        }
        if (min != -1)
            System.out.println("The first repeating element is " + arr[min]);
        else
            System.out.println("There are no repeating elements");
    }
    public static void main(String[] args) {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printFirstRepeating(arr);
    }
}
```

OUTPUT:



```
The first repeating element is 5

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

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(n)

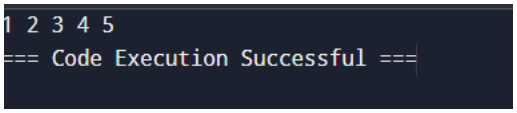
6. Remove duplicates from Sorted Array

Input: arr[] = {2, 2, 2, 2, 2}
Output: arr[] = {2}
Explanation: All the elements are 2, So only keep one instance of 2.
Input: arr[] = {1, 2, 2, 3, 4, 4, 4, 5, 5}
Output: arr[] = {1, 2, 3, 4, 5}
Input: arr[] = {1, 2, 3}
Output : arr[] = {1, 2, 3}
Explanation : No change as all elements are distinct

CODE:

```
class Main {
    static int removeDuplicates(int[] arr) {
        int n = arr.length;
        if (n <= 1)
            return n;
        int idx = 1;
        for (int i = 1; i < n; i++) {
            if (arr[i] != arr[i - 1]) {
                arr[idx++] = arr[i];
            }
        }
        return idx;
    }
    public static void main(String[] args) {
        int[] arr = {1, 2, 2, 3, 4, 4, 4, 5, 5};
        int newSize = removeDuplicates(arr);

        for (int i = 0; i < newSize; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

OUTPUT:



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

TIME COMPLEXITY:O(N)

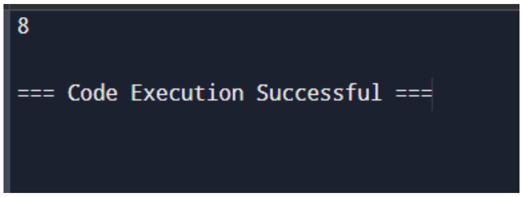
SPACE COMPLEXITY:O(1)

7.Maximum Index:

CODE:

```
public class Main {
    int maxIndexDiff(int arr[], int n) {
        int maxDiff = -1;
        for (int i = 0; i < n; ++i) {
            for (int j = n - 1; j > i; --j) {
                if (arr[j] > arr[i] && maxDiff < (j - i)) {
                    maxDiff = j - i;
                }
            }
        }
        return maxDiff;
    }
    public static void main(String[] args) {
        Main max = new Main();
        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);
    }
}
```

OUTPUT:



```
8

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

TIME COMPLEXITY:O(n)

SPACE COMPLEXITY: O(1)

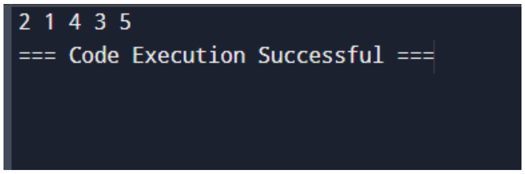
8.Wave Array

Input: arr[] = [1, 2, 3, 4, 5]
Output: [2, 1, 4, 3, 5]
Input: arr[] = [2, 4, 7, 8, 9, 10]
Output: [4, 2, 8, 7, 10, 9]

CODE:

```
public class Main {
  public static void waveSort(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i += 2) {
      if (i > 0 && arr[i] < arr[i - 1]) {
        int temp = arr[i];
        arr[i] = arr[i - 1];
        arr[i - 1] = temp;
      }
      if (arr[i] < arr[i + 1]) {
        int temp = arr[i];
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }
  }
  public static void main(String[] args) {
    int arr[] = {1, 2, 3, 4, 5};
    waveSort(arr);
    for (int num : arr) {
      System.out.print(num + " ");
    }
  }
}
```

OUTPUT:



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

TIME COMPLEXITY:O(n)

SPACE COMPLEXITY: O(1)