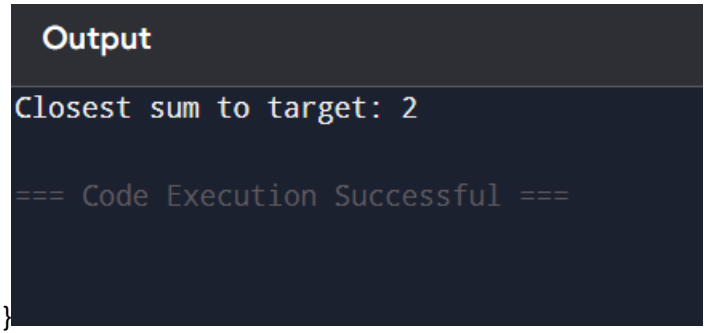
1) 3 SUM CLOSEST

```
import java.util.Arrays;
class Main {
  public int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int closestSum = nums[0] + nums[1] + nums[2];
    for (int i = 0; i < nums.length - 2; i++) {
      int j = i + 1; // Left pointer
      int k = nums.length - 1; // Right pointer
      while (j < k) {
        int sum = nums[i] + nums[j] + nums[k];
        if (Math.abs(target - sum) < Math.abs(target - closestSum)) {
          closestSum = sum;
        }
        if (sum < target) {
          j++;
        } else {
          k--;
        }
      }
    }
    return closestSum;
  }

  public static void main(String[] args) {
    Main solution = new Main();
    int[] nums = {-1, 2, 1, -4};
    int target = 1;
    System.out.println("Closest sum to target: " + solution.threeSumClosest(nums, target));
  }
```



}

Time Complexity: O(n^2)

Space Complexity: O(1)

2)JUMP GAME 2

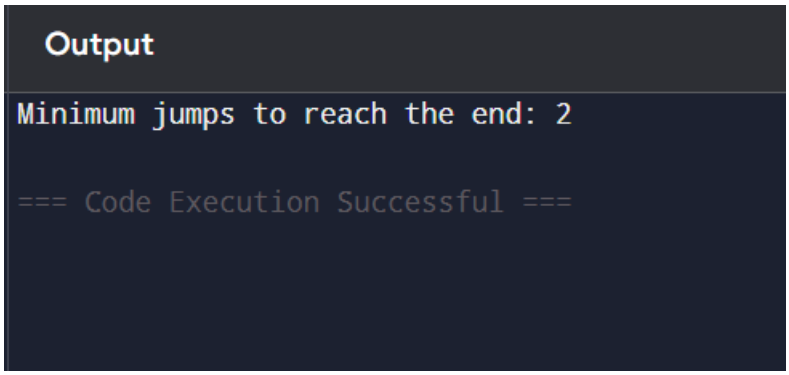
```
public class Solution {
    public int jump(int[] A) {
        int jumps = 0;
        int currFar = 0;
        int currEnd = 0;

        for (int i = 0; i < A.length - 1; i++) {
            currFar = Math.max(currFar, i + A[i]);
            if (i == currEnd) {
                jumps++; // Increment the jump count
                currEnd = currFar; // Update the end of the range
                if (currEnd >= A.length - 1) {
                    break;
                }
            }
        }
        return jumps;
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] arr = {2, 3, 1, 1, 4};
```

```
        System.out.println("Minimum jumps to reach the end: " + solution.jump(arr));
    }
}
```

TIME COMPLEXITY:O(n)

SPACE COMPLEXITY:O(1)



3)MERGE SORT

```
import java.io.*;
class Main {
    static void merge(int arr[], int l, int m, int r)
    {
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[] = new int[n1];
        int R[] = new int[n2];
        for (int i = 0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[m + 1 + j];
        int i = 0, j = 0;
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
```

```
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
static void sort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;
        sort(arr, l, m);
        sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
static void printArray(int arr[])
{
    int n = arr.length;
```

```
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        System.out.println("Given array is");
        printArray(arr);
        sort(arr, 0, arr.length - 1);
        System.out.println("\nSorted array is");
        printArray(arr);
    }
}
```

Output

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

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

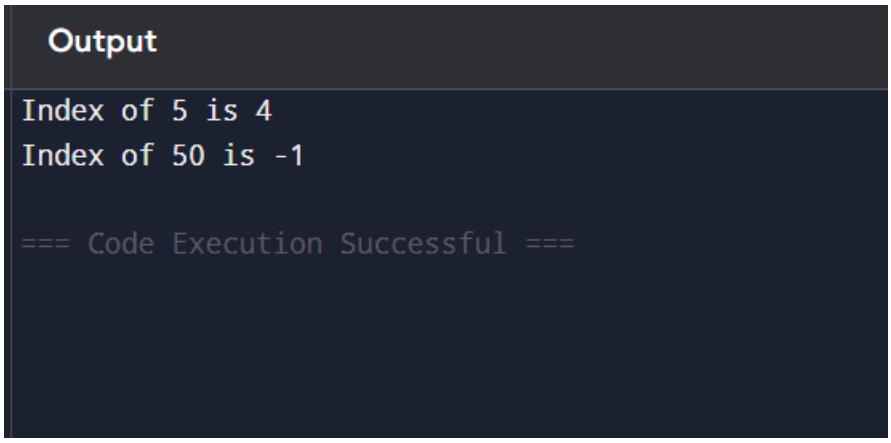
4)TERNARY SEARCH

```
class Main {
    static int ternarySearch(int l, int r, int key, int ar[])
    {
        if (r >= l) {
            int mid1 = l + (r - l) / 3;
            int mid2 = r - (r - l) / 3;
            if (ar[mid1] == key) {
```

```
            return mid1;
        }
        if (ar[mid2] == key) {
            return mid2;
        }
        if (key < ar[mid1]) {
            return ternarySearch(l, mid1 - 1, key, ar);
        }
        else if (key > ar[mid2]) {
            return ternarySearch(mid2 + 1, r, key, ar);
        }
        else {

            return ternarySearch(mid1 + 1, mid2 - 1, key, ar);
        }
    }
    return -1;
}
public static void main(String args[])
{
    int l, r, p, key;
    int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    l = 0;
    r = 9;
    key = 5;
    p = ternarySearch(l, r, key, ar);
    System.out.println("Index of " + key + " is " + p);
    key = 50;
    p = ternarySearch(l, r, key, ar);
    System.out.println("Index of " + key + " is " + p);
}
```

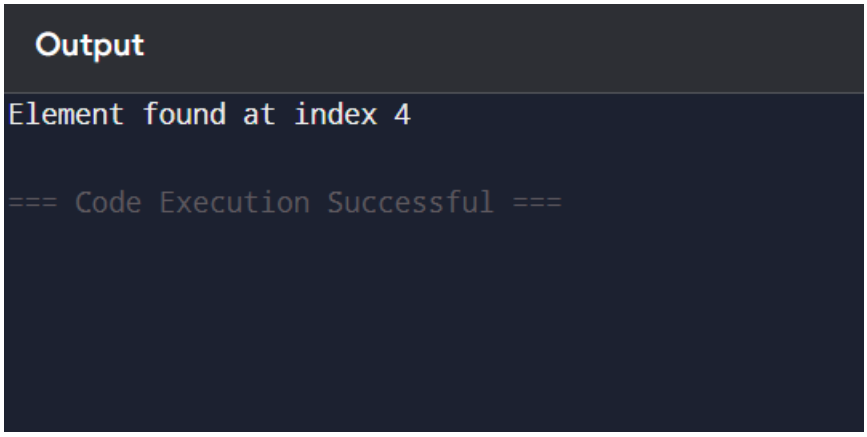
}



5)INTERPOLATION SEARCH:

```
import java.util.*;
class Main {
	public static int interpolationSearch(int arr[], int lo,int hi, int x)
	{
		int pos;
		if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {
			pos = lo+ (((hi - lo) / (arr[hi] - arr[lo]))* (x - arr[lo]));
			if (arr[pos] == x)
				return pos;
			if (arr[pos] < x)
				return interpolationSearch(arr, pos + 1, hi,x);
			if (arr[pos] > x)
				return interpolationSearch(arr, lo, pos - 1,x);
		}
		return -1;
	}
	public static void main(String[] args)
	{
		int arr[] = { 10, 12, 13, 16, 18, 19, 20, 21,
					22, 23, 24, 33, 35, 42, 47 };
```

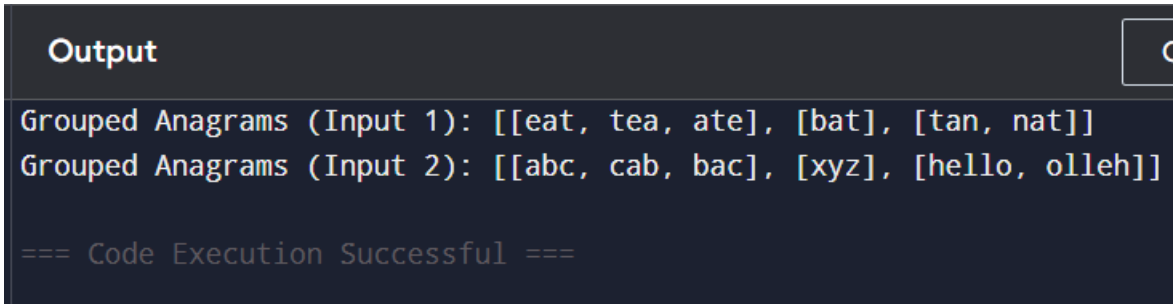
```
        int n = arr.length;
        int x = 18;
        int index = interpolationSearch(arr, 0, n - 1, x);
        if (index != -1)
            System.out.println("Element found at index "
                                                + index);
        else
            System.out.println("Element not found.");
    }
}
```



6)GROUP ANAGRAMS

```
import java.util.*;
public class Main {
   public List<List<String>> groupAnagrams(String[] strs) {
      List<List<String>> result = new ArrayList<>();
      Map<String, List<String>> map = new HashMap<>();
      for (String str : strs) {
         char[] chars = str.toCharArray();
         Arrays.sort(chars);
         String sortedString = new String(chars);
         map.putIfAbsent(sortedString, new ArrayList<>());
```

```
            map.get(sortedString).add(str);
        }
        result.addAll(map.values());
        return result;
    }
    public static void main(String[] args) {
        Main solution = new Main();
        String[] input1 = {"eat", "tea", "tan", "ate", "nat", "bat"};
        String[] input2 = {"hello", "olleh", "abc", "cab", "bac", "xyz"};
        System.out.println("Grouped Anagrams (Input 1): " + solution.groupAnagrams(input1));
        System.out.println("Grouped Anagrams (Input 2): " + solution.groupAnagrams(input2));
    }
}
```



Time Complexity: O(n·klogk)

Space Complexity: O(n·k)

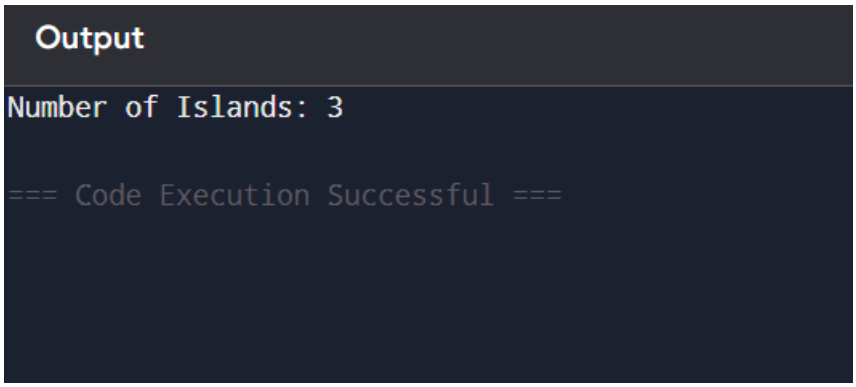
7)NUMBER OF ISLANDS:

```
public class Main {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int numIslands = 0;
        int rows = grid.length;
        int cols = grid[0].length;
```

```
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == '1') {
                numIslands++;
                dfs(grid, i, j);
            }
        }
    }
    return numIslands;
}
private void dfs(char[][] grid, int i, int j) {
    if (i < 0 || i >= grid.length || j < 0 || j >= grid[0].length || grid[i][j] == '0') {
        return;
    }
    grid[i][j] = '0';
    dfs(grid, i + 1, j);
    dfs(grid, i - 1, j);
    dfs(grid, i, j + 1);
    dfs(grid, i, j - 1);
}

public static void main(String[] args) {
    Main solution = new Main();
    char[][] grid = {
        {'1', '1', '0', '0', '0'},
        {'1', '1', '0', '0', '0'},
        {'0', '0', '1', '0', '0'},
        {'0', '0', '0', '1', '1'}
    };
    System.out.println("Number of Islands: " + solution.numIslands(grid));
}
```

}



Time Complexity: O(m×n)

Space Complexity: O(m×n)

8)DECODE WAYS

```
public class Solution {
    public int numDecodings(String s) {
        int dp1 = 1, dp2 = 0, n = s.length();
        for (int i = n - 1; i >= 0; i--) {
            int dp = s.charAt(i) == '0' ? 0 : dp1;
            if (i < n - 1 && (s.charAt(i) == '1' || s.charAt(i) == '2' && s.charAt(i + 1) < '7'))
                dp += dp2;
            dp2 = dp1;
            dp1 = dp;
        }
        return dp1;
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        String test1 = "12";
        String test2 = "226";
        String test3 = "06";
        System.out.println("Number of decodings for '12': " + solution.numDecodings(test1));
        System.out.println("Number of decodings for '226': " + solution.numDecodings(test2));
```

```
        System.out.println("Number of decodings for '06': " + solution.numDecodings(test3));
    }
}
```

```
Output
Number of decodings for '12': 2
Number of decodings for '226': 3
Number of decodings for '06': 0

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

TIME COMPLEXITY:O(n)

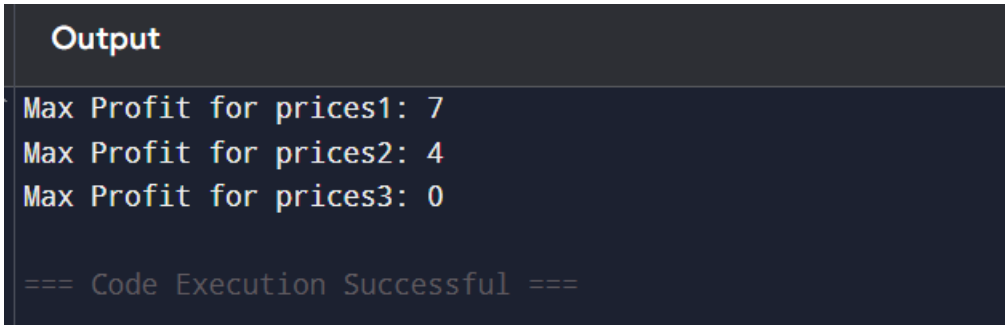
SPACE COMPLEXITY:O(1)

9)BEST TIME TO BUY AND SELL STOCK 2:

```
public class Solution {
    public int maxProfit(int[] prices) {
        int i = 0, buy, sell, profit = 0, N = prices.length - 1;
        while (i < N) {
            while (i < N && prices[i + 1] <= prices[i]) i++;
            buy = prices[i];
            while (i < N && prices[i + 1] > prices[i]) i++;
            sell = prices[i];
            profit += sell - buy;
        }
        return profit;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] prices1 = {7, 1, 5, 3, 6, 4};
        int[] prices2 = {1, 2, 3, 4, 5};
```

```
        int[] prices3 = {7, 6, 4, 3, 1};
        System.out.println("Max Profit for prices1: " + solution.maxProfit(prices1));
        System.out.println("Max Profit for prices2: " + solution.maxProfit(prices2));
        System.out.println("Max Profit for prices3: " + solution.maxProfit(prices3));
    }
}
```



TIME COMPLEXITY:O(n)

SPACE COMPLEXITY:O(1)