1. Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Input: nums = [-1,2,1,-4], target = 1

Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

CODE:

```
import java.util.Arrays;

class Solution {

  public int threeSumClosest(int[] nums, int target) {

    Arrays.sort(nums);

    int closest = nums[0] + nums[1] + nums[2];

    for (int i = 0; i < nums.length - 2; i++) {

      int left = i + 1, right = nums.length - 1;

      while (left < right) {

        int currSum = nums[i] + nums[left] + nums[right];

        if (currSum == target) {

          return target;

        }

        if (Math.abs(currSum - target) < Math.abs(closest - target)) {

          closest = currSum;

        }

        if (currSum < target) {

          left++;

        } else {

          right--;

        }

      }

    }

    return closest;
    }
}
```

```
1
Output
2
Expected
2
```

Time Complexity: O(n)

2. You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach* nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

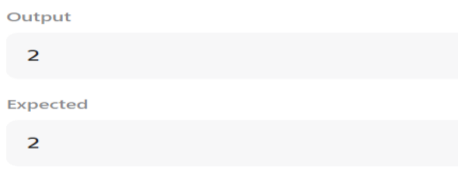
Input: nums = [2,3,1,1,4]

Output: 2

Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

CODE:

```
class Solution {
    public int jump(int[] nums) {
        int n = nums.length;
        if (n == 1) return 0;
        int jumps = 0;
        int maxReach = 0;
        int currentEnd = 0;
        for (int i = 0; i < n - 1; i++) {
            maxReach = Math.max(maxReach, i + nums[i]);
            if (i == currentEnd) {
                jumps++;
                currentEnd = maxReach;
            }
        }
        return jumps;
    }
}
```



```
Output
2
Expected
2
```

Time Complexity:O(n)

3. Given an array of strings strs, group the

anagrams

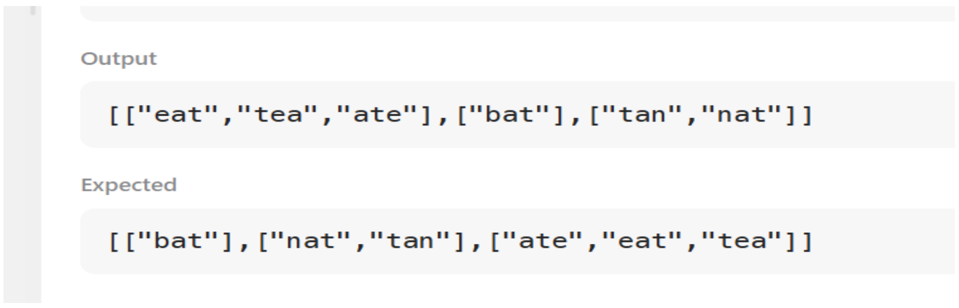
together. You can return the answer in **any order**.

Input: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

CODE:

```
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> map = new HashMap<>();
        for (String str : strs) {
            char[] charArray = str.toCharArray();
            Arrays.sort(charArray);
            String sortedStr = new String(charArray);
            if (!map.containsKey(sortedStr)) {
                map.put(sortedStr, new ArrayList<>());
            }
            map.get(sortedStr).add(str);
        }
        return new ArrayList<>(map.values());
     }

}
```


Output

```
[["eat","tea","ate"],["bat"],["tan","nat"]]
```

Expected

```
[["bat"],["nat","tan"],["ate","eat","tea"]]
```

Time Complexity:O(n)

4. You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:

"1" -> 'A'
"2" -> 'B'
...
"25" -> 'Y'
"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

- "AAJF" with the grouping (1, 1, 10, 6)
- "KJF" with the grouping (11, 10, 6)
- The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the **number of ways** to **decode** it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a **32-bit** integer.

Input: s = "12"

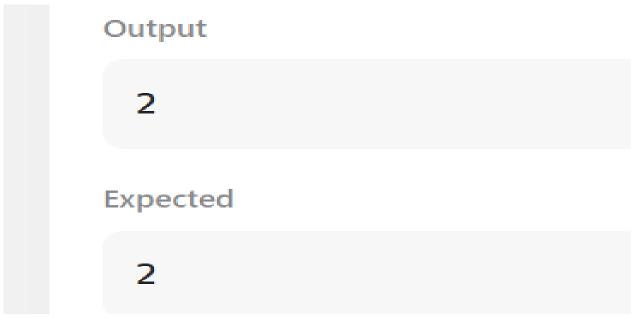
Output: 2

Explanation:

"12" could be decoded as "AB" (1 2) or "L" (12).

CODE:

```
class Solution {
    public int numDecodings(String s) {
        if (s == null || s.length() == 0 || s.charAt(0) == '0') {
            return 0;
        }
        int n = s.length();
        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = 1;
        for (int i = 2; i <= n; i++) {
            int oneDigit = Integer.parseInt(s.substring(i - 1, i));
            int twoDigits = Integer.parseInt(s.substring(i - 2, i));

            if (oneDigit >= 1 && oneDigit <= 9) {
                dp[i] += dp[i - 1];
            }
            if (twoDigits >= 10 && twoDigits <= 26) {
                dp[i] += dp[i - 2];
            }
        }
        return dp[n];
    }
}
```



```
Output
2
Expected
2
```

Time Complexity:O(n)

5. Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Input: grid = [

  ["1","1","1","1","0"],

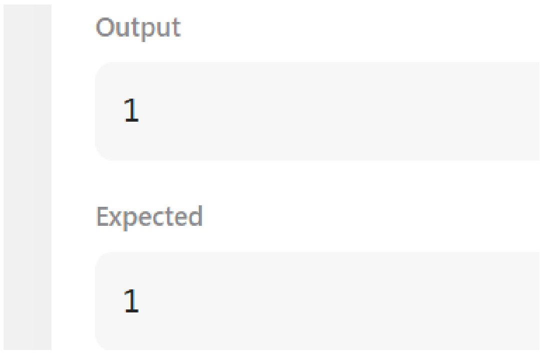
  ["1","1","0","1","0"],

```
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
```

Output: 1

CODE:

```
class Solution {
    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 row, int col) {
        if (row < 0 || col < 0 || row >= grid.length || col >= grid[0].length || grid[row][col] == '0') {
            return;
        }
        grid[row][col] = '0';
        dfs(grid, row - 1, col);
        dfs(grid, row + 1, col);
        dfs(grid, row, col - 1);
        dfs(grid, row, col + 1);
    }
}
```



```
Output

1

Expected

1
```

Time Complexity:O(n*m)

6. You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

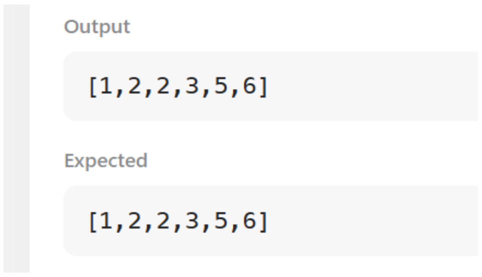
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

CODE:

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int p1 = m - 1;
        int p2 = n - 1;
        for (int p = n + m - 1; p >= 0; p--) {
            if (p2 < 0) {
                break;
            }

            if (p1 >= 0 && nums1[p1] > nums2[p2]) {
```

```
            nums1[p] = nums1[p1];
            p1--;
          } else {
            nums1[p] = nums2[p2];
            p2--;
          }
        }
    }
}
```



Output

```
[1,2,2,3,5,6]
```

Expected

```
[1,2,2,3,5,6]
```

Time Complexity:O(n+m)

7. Given a sorted array **arr[]** of size **N** and an integer **K**. The task is to check if K is present in the array or not using ternary search.- It is a divide and conquer algorithm that can be used to find an element in an array. In this algorithm, we divide the given array into three parts and determine which has the key (searched element).

Input:

N = 5, K = 6

arr[] = {1,2,3,4,6}

Output: 1

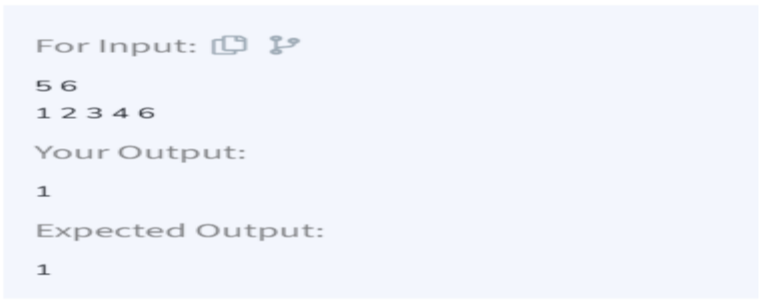
Exlpanation: Since, 6 is present in

the array at index 4 (0-based indexing),

output is 1.

CODE:

```
class Solution{
  static int ternarySearch(int arr[], int N, int K)
  {
      int left = 0, right = N - 1;
      while (left <= right) {
            int mid1 = left + (right - left) / 3;

            int mid2 = right - (right - left) / 3;

            if (arr[mid1] == K) {

                return 1;

            }

            if (arr[mid2] == K) {

                return 1;

            }

            if (K < arr[mid1]) {

                right = mid1 - 1;

            } else if (K > arr[mid2]) {

                left = mid2 + 1;

            } else {

                left = mid1 + 1;

                right = mid2 - 1;

            }

        }

        return -1;

    }

}
```



```
For Input:
5 6
1 2 3 4 6
Your Output:
1
Expected Output:
1
```

Time Complexity: O(log3N)