DSA Practice Question Set - 8

1)3 sum closest :

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.

Example 1:

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).

Example 2:

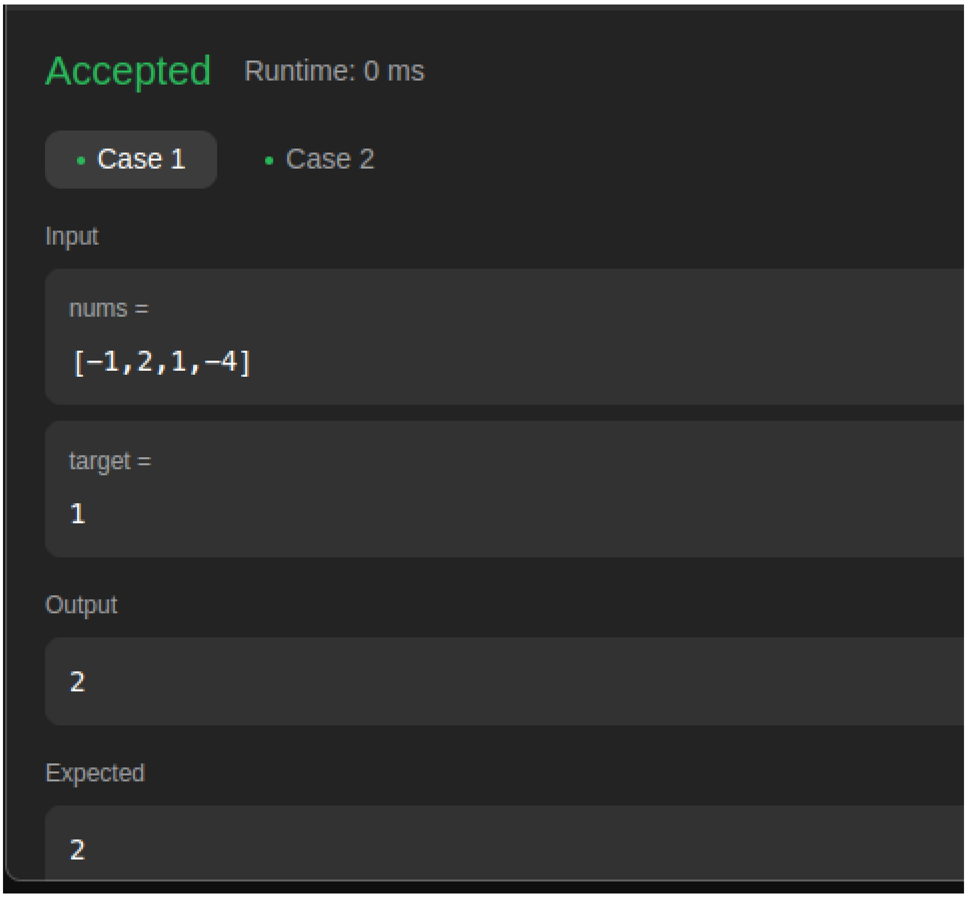
Input: nums = [0,0,0], target = 1
Output: 0
Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

Program :

```
import java.util.Arrays;

public class Solution {
  public int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int closestSum = Integer.MAX_VALUE;

    for (int i = 0; i < nums.length - 2; i++) {
      int left = i + 1;
      int right = nums.length - 1;

      while (left < right) {
```

```
                int currentSum = nums[i] + nums[left] + nums[right];

                if (Math.abs(currentSum - target) < Math.abs(closestSum -
target)) {
                    closestSum = currentSum;
                }

                if (currentSum < target) {
                    left++;
                } else if (currentSum > target) {
                    right--;
                } else {
                    return currentSum;
                }
            }
        }

        return closestSum;
    }
}
```

Output :



```
Input
nums =
[-1,2,1,-4]
target =
1
Output
2
Expected
2
```

Time Complexity : O(n^2)
Space Complexity : O(log n)

2) JUMP GAME II

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].

Example 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.

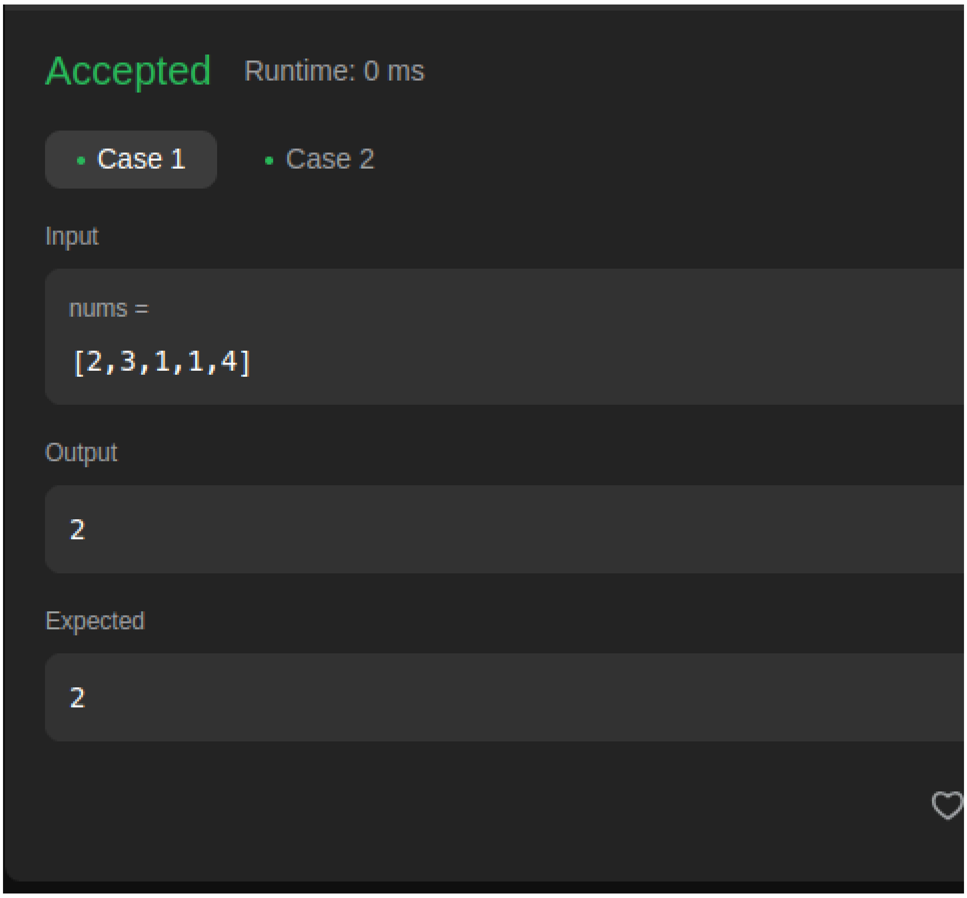
Example 2:

Input: nums = [2,3,0,1,4]
Output: 2

Program :

```
class Solution {
    public int jump(int[] nums) {
        int left = 0,right = 0,jump = 0;

        while (right<nums.length - 1){
            int far = 0;
            for(int i = left;i<=right;i++){
                far = Math.max(far,i+nums[i]);
            }
            left = right+1;
            right = far;
            jump++;
        }
        return jump;


    }
}
```

Output :



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =

```
[2,3,1,1,4]
```

Output

```
2
```

Expected

```
2
```

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

3)Group Anagrams:

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

Example 1:

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

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

Explanation:

- **There is no string in strs that can be rearranged to form "bat".**
- **The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.**

- The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

Example 2:

Input: strs = [""]

Output: [[""]]

Example 3:

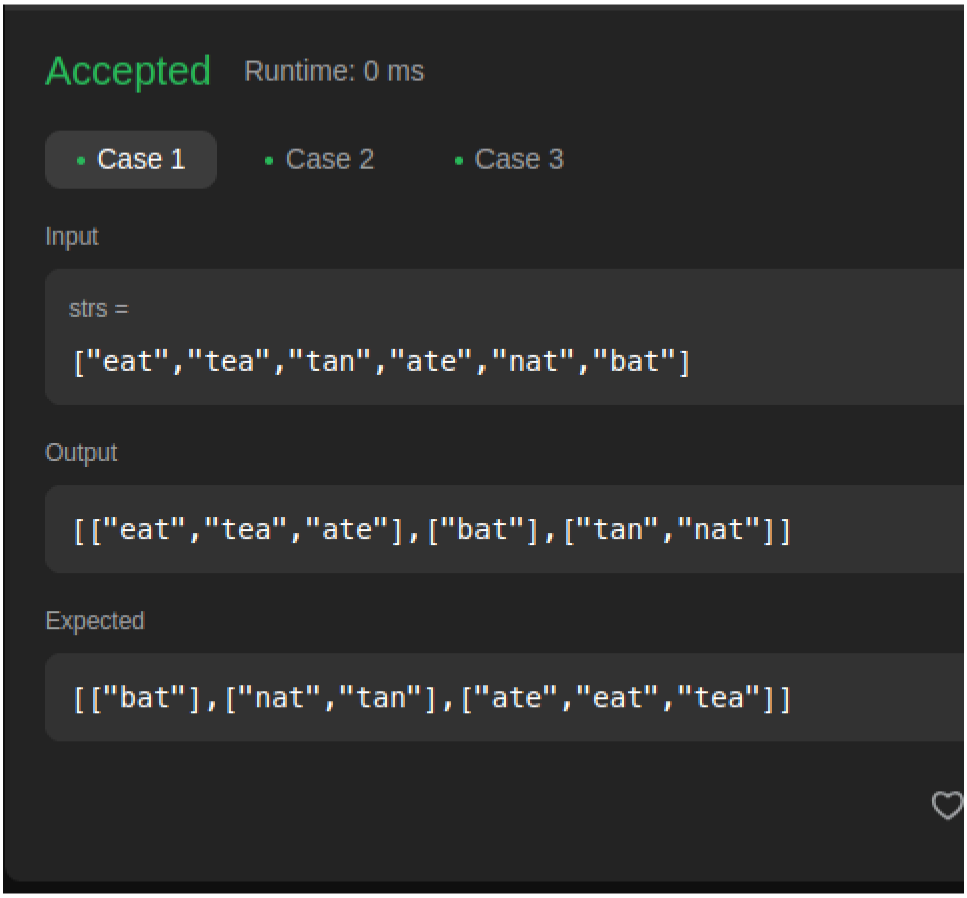
Input: strs = ["a"]

Output: [["a"]]

Program :

```
class Solution {

  public List<List<String>> groupAnagrams(String[] strs) {

    Map<String,List<String>> map = new HashMap<>();


    for(String word : strs){

     char[] chars = word.toCharArray();

     Arrays.sort(chars);

     String sortedchar = new String(chars);


     if(!map.containsKey(sortedchar)){

        map.put(sortedchar,new ArrayList<>());

     }

     map.get(sortedchar).add(word);
    }

    return new ArrayList<>(map.values());

  }

}
```

Output :



Accepted Runtime: 0 ms

Input

strs =

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

Output

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

Expected

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

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

4)Best Time to Buy and Sell Stock II

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.

Find and return *the maximum profit you can achieve*.

Example 1:

Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
Total profit is 4.

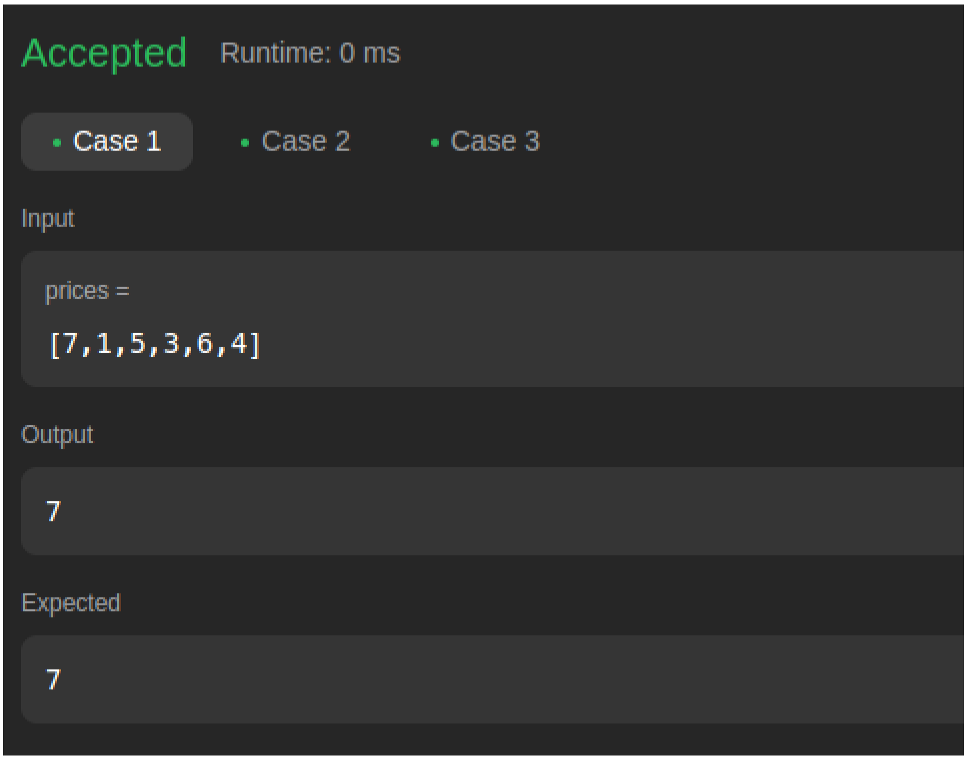
Example 3:

Input: prices = [7,6,4,3,1]
Output: 0
Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Program :

```
class Solution {
  public int maxProfit(int[] prices) {
     int prof = 0;

     for(int i=1;i<prices.length;i++){
        if(prices[i]>prices[i-1]){
                prof+=prices[i] - prices[i-1];
            }
        }
        return prof;
    }
}
```

Output :



Accepted Runtime: 0 ms

Input

```
prices =
[7,1,5,3,6,4]
```

Output

```
7
```

Expected

```
7
```

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

5)Decode Ways :

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.

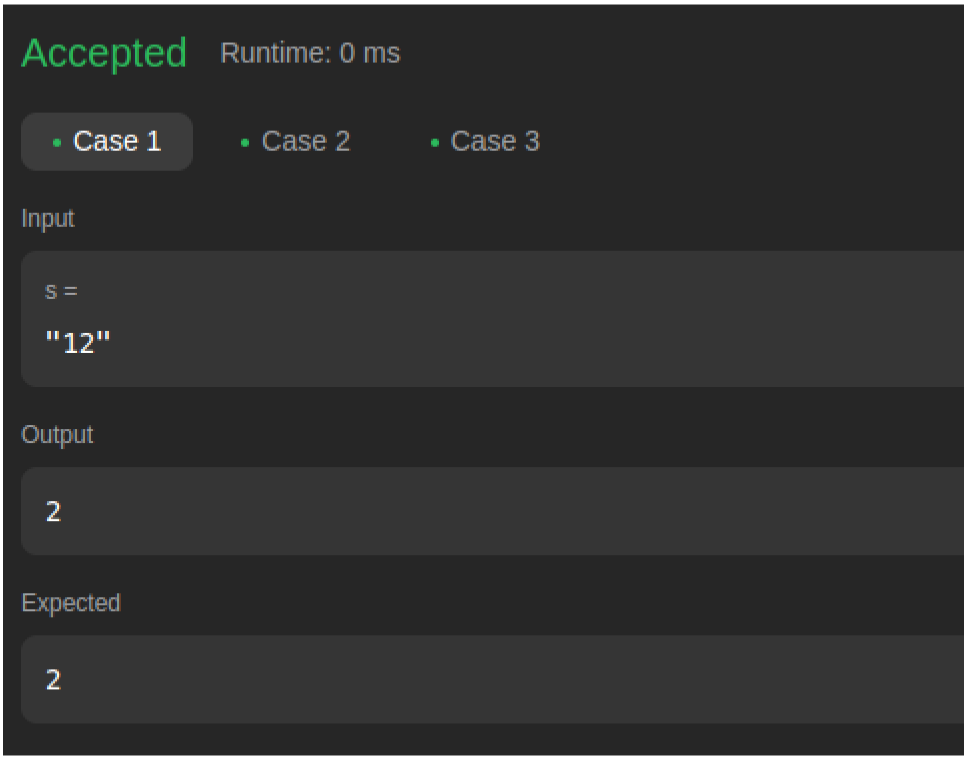
Program :

```
class Solution {

  public int numDecodings(String s) {

    // Convert the input string to a character array for easier access to individual characters

    char[] a = s.toCharArray();

    int n = a.length;



    // Initialize a DP array with -1 to keep track of already computed results

    int[] dp = new int[n + 1];
```

```
        for (int i = 0; i <= n; i++) dp[i] = -1;


        // Return 0 if the string starts with '0' or if there are two consecutive '0's at the beginning

        if (a[0] == '0' || (a[0] == '0' && n > 1 && a[1] == '0')) return 0;


        // Call helper function to start decoding from index 0

        return get(0, a, n, dp);

    }


    static int get(int i, char[] a, int n, int[] dp) {

        // Base cases: if at the end of string with valid character, or beyond the last character, return 1

        if ((i == n - 1 && a[n - 1] != '0') || i >= n) return 1;


        // If current character is '0', there's no valid decoding for this path

        if (a[i] == '0') {

            return dp[i] = 0;

        }


        // If result for this index is already calculated, return it
        if (dp[i] != -1) return dp[i];

        // Recursive call to decode one character
        int pickone = get(i + 1, a, n, dp);

        // Initialize two-character decoding path as 0
        int picktwo = 0;

        // Get the numeric value of the two-digit combination
        int k1 = (int) (a[i] - '0');
        int k2 = (int) (a[i + 1] - '0');
        k1 *= 10;

        // If two-digit number is within 1-26, try decoding it
        if (k1 + k2 <= 26) {
            picktwo = get(i + 2, a, n, dp);
        }

        // Store the result for this index in dp array and return it
        return dp[i] = pickone + picktwo;
    }
```

```
}
```

Output :



Accepted Runtime: 0 ms

Input

s =

"12"

Output

2

Expected

2

Time Complexity : O(n)

Space Complexity : O(n)

6)Number of Islands :

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.

Example 1:

Input: grid = [

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

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

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

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

]

Output: 1

Example 2:

Input: grid = [

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

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

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

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

]

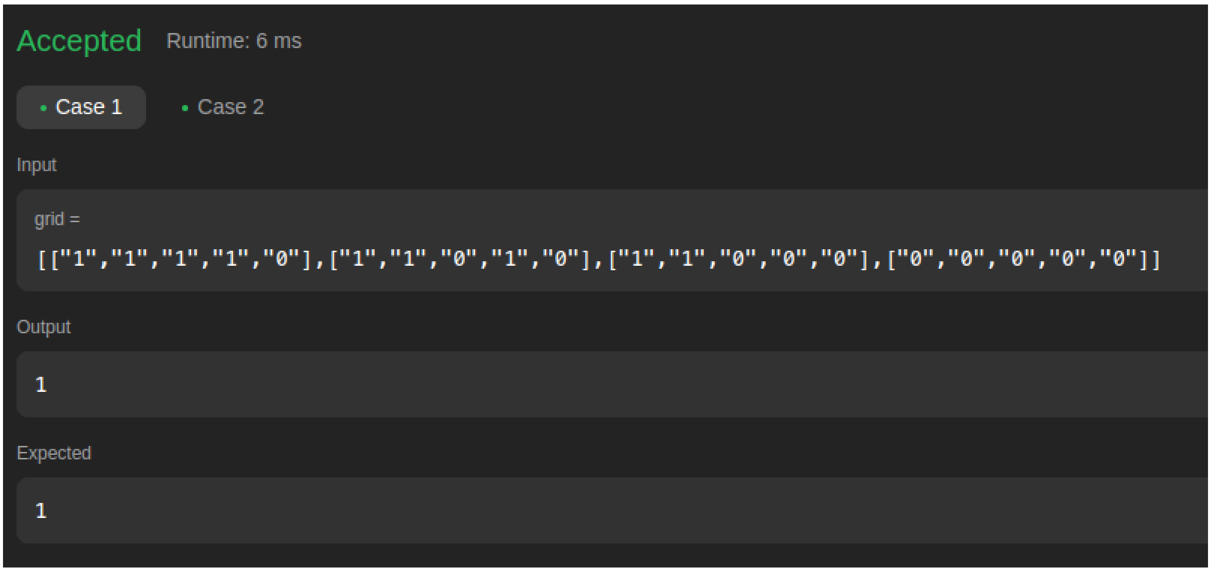
Output: 3

```
class Solution {
  public int numIslands(char[][] grid) {
     int islands = 0;
     int rows = grid.length;
     int cols = grid[0].length;
     Set<String> visited = new HashSet<>();
```

```
    int[][] directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};


    for (int r = 0; r < rows; r++) {
      for (int c = 0; c < cols; c++) {
        if (grid[r][c] == '1' && !visited.contains(r + "," + c)) {
          islands++;
          bfs(grid, r, c, visited, directions, rows, cols);
        }
      }
    }


    return islands;
  }


  private void bfs(char[][] grid, int r, int c, Set<String> visited, int[][] directions, int rows, int cols) {
    Queue<int[]> q = new LinkedList<>();
    visited.add(r + "," + c);
    q.add(new int[]{r, c});

```

```
        while (!q.isEmpty()) {
            int[] point = q.poll();
            int row = point[0], col = point[1];


            for (int[] direction : directions) {
                int nr = row + direction[0], nc = col + direction[1];
                if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1' && !visited.contains(nr + "," + nc)) {
                    q.add(new int[]{nr, nc});
                    visited.add(nr + "," + nc);
                }
            }
        }
    }
}
```

Output :



Input

```
grid =
[["1","1","1","1","0"],["1","1","0","1","0"],["1","1","0","0","0"],["0","0","0","0","0"]]
```

Output

```
1
```

Expected

```
1
```

Time Complexity : O(m*n)

Space Complexity : O(m*n)

7)Quick Sort :

```
class QuickSort {


  // Partition function
  static int partition(int[] arr, int low, int high) {

     // Choose the pivot
     int pivot = arr[high];

     // Index of smaller element and indicates
     // the right position of pivot found so far
     int i = low - 1;


     // Traverse arr[low..high] and move all smaller
     // elements to the left side. Elements from low to
```

```
        // i are smaller after every iteration
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }

        // Move pivot after smaller elements and
        // return its position
        swap(arr, i + 1, high);
        return i + 1;
    }


    // Swap function
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }


    // The QuickSort function implementation
    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {

            // pi is the partition return index of pivot
            int pi = partition(arr, low, high);


            // Recursion calls for smaller elements
            // and greater or equals elements
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }


    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        quickSort(arr, 0, n - 1);

        for (int val : arr) {
            System.out.print(val + " ");
        }
    }
}
```

Output :

```
Sorted Array
1 5 7 8 9 10
```

Time Complexity : O(n^2)
Space Complexity : O(n)

8)Merge Sort :

```
// Java program for Merge Sort
import java.io.*;

class GfG {
```

```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
static void merge(int arr[], int l, int m, int r)
{
    // Find sizes of two subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temp arrays
    int L[] = new int[n1];
    int R[] = new int[n2];

    // Copy data to temp arrays
    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];

    // Merge the temp arrays

    // Initial indices of first and second subarrays
    int i = 0, j = 0;

    // Initial index of merged subarray array
    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++;
    }

    // Copy remaining elements of L[] if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy remaining elements of R[] if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Main function that sorts arr[l..r] using
// merge()
static void sort(int arr[], int l, int r)
{
    if (l < r) {

        // Find the middle point
        int m = l + (r - l) / 2;

        // Sort first and second halves
```

```
            sort(arr, l, m);
            sort(arr, m + 1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }

    // A utility function to print array of size n
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver code
    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
```

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

9)Ternary Search :

```
// Java program to illustrate
// recursive approach to ternary search

class TernarySearch {

   // Function to perform Ternary Search
   static int ternarySearch(int l, int r, int key, int ar[])
   {
       if (r >= l) {

           // Find the mid1 and mid2
           int mid1 = l + (r - l) / 3;
           int mid2 = r - (r - l) / 3;

           // Check if key is present at any mid
           if (ar[mid1] == key) {
               return mid1;
           }
           if (ar[mid2] == key) {
               return mid2;
           }
```

```
            // Since key is not present at mid,
            // check in which region it is present
            // then repeat the Search operation
            // in that region

            if (key < ar[mid1]) {

                // The key lies in between l and mid1
                return ternarySearch(l, mid1 - 1, key, ar);
            }
            else if (key > ar[mid2]) {

                // The key lies in between mid2 and r
                return ternarySearch(mid2 + 1, r, key, ar);
            }
            else {

                // The key lies in between mid1 and mid2
                return ternarySearch(mid1 + 1, mid2 - 1, key, ar);
            }
        }

        // Key not found
        return -1;
    }

    // Driver code
    public static void main(String args[])
    {
        int l, r, p, key;
        // Get the array
        // Sort the array if not sorted
        int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        // Starting index
        l = 0;

        // end element index
        r = 9;

        // Checking for 5

        // Key to be searched in the array
        key = 5;

        // Search the key using ternarySearch
        p = ternarySearch(l, r, key, ar);

        // Print the result
        System.out.println("Index of " + key + " is " + p);

        // Checking for 50

        // Key to be searched in the array
        key = 50;

        // Search the key using ternarySearch
        p = ternarySearch(l, r, key, ar);

        // Print the result
        System.out.println("Index of " + key + " is " + p);
    }
```

```
}
```

Output :

```

Index of 5 is 4
Index of 50 is -1
```

Time Complexity : O(2*log3n)
Space Complexity : O(log3n)

10)Interpolation Search :

```
// Java program to implement interpolation
// search with recursion
import java.util.*;

class GFG {

  // If x is present in arr[0..n-1], then returns
  // index of it, else returns -1.
  public static int interpolationSearch(int arr[], int lo,
                                        int hi, int x)
  {
     int pos;

     // Since array is sorted, an element
     // present in array must be in range
     // defined by corner
     if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {

         // Probing the position with keeping
         // uniform distribution in mind.
         pos = lo
             + (((hi - lo) / (arr[hi] - arr[lo]))
```

```java
                * (x - arr[lo]));

        // Condition of target found
        if (arr[pos] == x)
            return pos;

        // If x is larger, x is in right sub array
        if (arr[pos] < x)
            return interpolationSearch(arr, pos + 1, hi,
                                    x);

        // If x is smaller, x is in left sub array
        if (arr[pos] > x)
            return interpolationSearch(arr, lo, pos - 1,
                                    x);
    }
    return -1;
}

// Driver Code
public static void main(String[] args)
{

    // Array of items on which search will
    // be conducted.
    int arr[] = { 10, 12, 13, 16, 18, 19, 20, 21,
                22, 23, 24, 33, 35, 42, 47 };

    int n = arr.length;

    // Element to be searched
    int x = 18;
    int index = interpolationSearch(arr, 0, n - 1, x);

    // If element was found
    if (index != -1)
      System.out.println("Element found at index "
                + index);
    else
      System.out.println("Element not found.");
  }
}
```

Output :

```
Element found at index 4
```

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