

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. $(-1 + 2 + 1 = 2)$.

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;
                }
            }
        }
    }
}
```

```

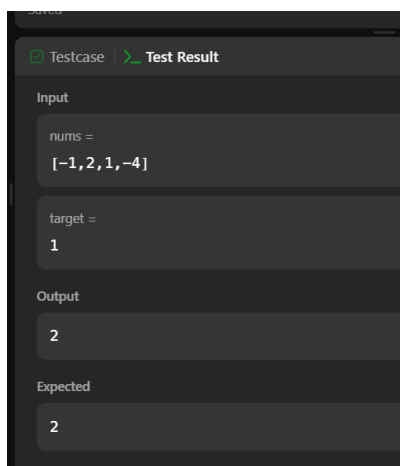
        if (Math.abs(currSum - target) < Math.abs(closest - target)) {
            closest = currSum;
        }

        if (currSum < target) {
            left++;
        } else {
            right--;
        }
    }
}

return closest;
}
}

```

Output:



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:

- $0 \leq j \leq \text{nums}[i]$ and
- $i + j < n$

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

Example 1:

Input: $\text{nums} = [2, 3, 1, 1, 4]$

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 ans = 0;
        int end = 0;
        int farthest = 0;

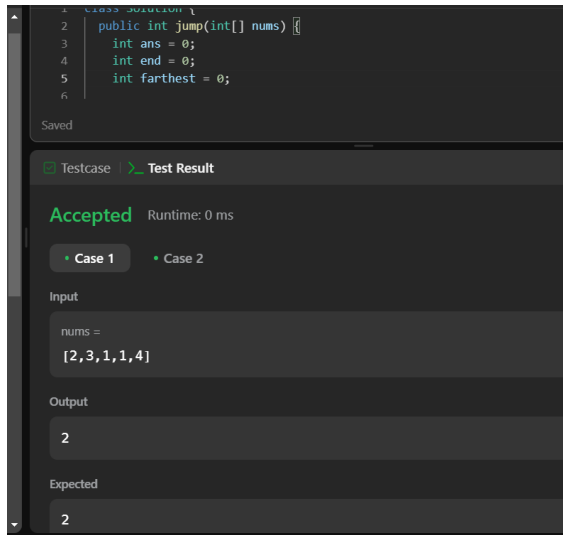
        for (int i = 0; i < nums.length - 1; ++i) {
            farthest = Math.max(farthest, i + nums[i]);
            if (farthest >= nums.length - 1) {
                ++ans;
                break;
            }
            if (i == end) {
                ++ans;
                end = farthest;
            }
        }
    }
}
```

```

        return ans;
    }
}

```

Output:



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.

Code:

```

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 sortedWord = new String(chars);

    if (!map.containsKey(sortedWord)) {
        map.put(sortedWord, new ArrayList<>());
    }

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

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

```

Output:

```

1 class Solution {
2     public List<List<String>> groupAnagrams(String[] strs)
3     {
4         Map<String, List<String>> map = new HashMap<>();
5     }
6 }

```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

strs =

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

Output

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

Expected

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

4.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.

Example 1:

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) {  
        int strLen = s.length();  
  
        int[] dp = new int[strLen + 1];
```

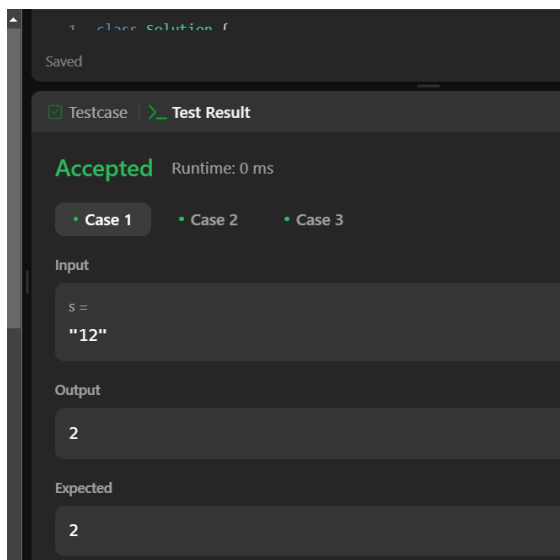
```

    dp[0] = 1;
    if (s.charAt(0) != '0') {
        dp[1] = 1;
    } else {
        return 0;
    }
    for (int i = 2; i <= strLen; ++i) {
        if (s.charAt(i - 1) != '0') {
            dp[i] += dp[i - 1];
        }
        if (s.charAt(i - 2) == '1' ||
            (s.charAt(i - 2) == '2' && s.charAt(i - 1) <= '6')) {
            dp[i] += dp[i - 2];
        }
    }

    return dp[strLen];
}
}

```

Output:



5.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 i^{th} 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 $4 + 3 = 7$.

Code:

```
class Solution {
    public int maxProfit(int[] prices) {
        int max = 0;
        int start = prices[0];
        int len = prices.length;
        for(int i = 1; i < len; i++){
            if(start < prices[i]) max += prices[i] - start;
            start = prices[i];
        }
        return max;
    }
}
```

Output:


```
1 class Solution {
2     public int maxProfit(int[] prices) {
3         int max = 0;
4         int start = prices[0];
5         for (int i = 1; i < prices.length; i++) {
6             if (prices[i] > start) {
7                 max = Math.max(max, prices[i] - start);
8             }
9             start = prices[i];
10        }
11        return max;
12    }
13 }
```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

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

Output

7

Expected

7

6. Number of islands

Given an $m \times n$ 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

Code:

```
class Solution{
public void dfs(char[][]grid,int i,int j){
int m=grid.length,n=grid[0].length;
if(i<0||j<0||i>=m||j>=n||grid[i][j]=='0')return;
```

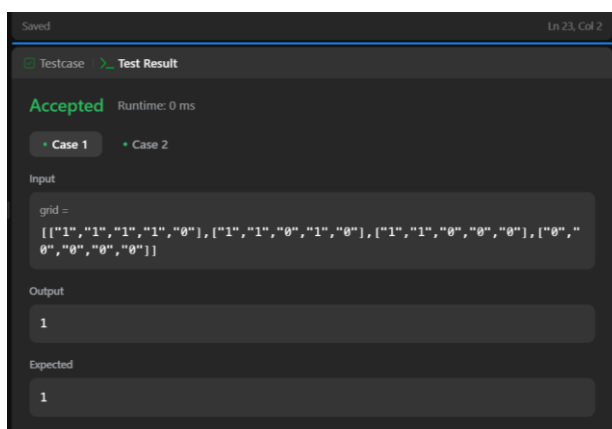
```

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 int numIslands(char[][]grid){
int m=grid.length,n=grid[0].length,count=0;
for(int i=0;i<m;i++){
for(int j=0;j<n;j++){
if(grid[i][j]=='1'){
count++;
dfs(grid,i,j);
}
}
}
return count;
}
}

```

Output:



7. Quick Sort

Code: import java.util.Arrays;

```

class Quicksort {

    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return i + 1;
    }

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

    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

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

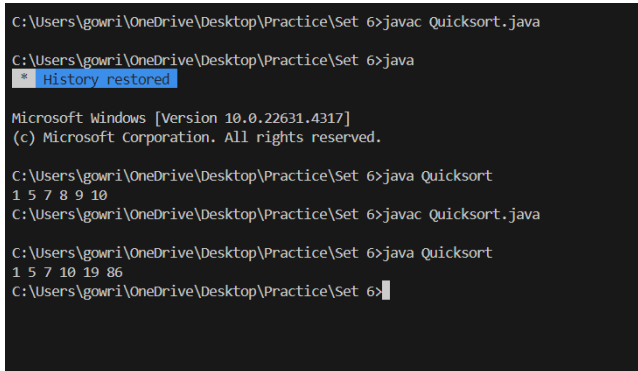
```

```

        int n = arr.length;
        quickSort(arr, 0, n - 1);
    for (int val : arr) {
        System.out.print(val + " ");
    }
}
}
}

```

Output:



```

C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>javac Quicksort.java
C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>java
* History restored
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>java Quicksort
1 5 7 8 9 10
C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>javac Quicksort.java

C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>java Quicksort
1 5 7 10 19 86
C:\Users\gowri\OneDrive\Desktop\Practice\Set 6>

```

Time Complexity: $O(n \log n)$

8.Merge sort

Code:

```

import java.util.Scanner;

class MergeSort {
    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, k = 1;
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[]) {
        for (int i = 0; i < arr.length; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int arr[] = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        System.out.println("Given array is:");
        printArray(arr);
        sort(arr, 0, arr.length - 1);
        System.out.println("Sorted array is:");
        printArray(arr);
        scanner.close();
    }
}

```

Output:

```

C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>javac MergeSort.java
C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>java MergeSort
Enter the number of elements in the array:
8
Enter the elements of the array:
38 27 16 45 12 8 98 102
Given array is:
38 27 16 45 12 8 98 102
Sorted array is:
8 12 16 27 38 45 98 102
C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>

```

9.Ternary Search

Code:

```
import java.util.Scanner;
import java.util.Arrays;
class TernarySearch {
    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[]) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
```

```

int[] ar = new int[n];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {
    ar[i] = scanner.nextInt();
}

Arrays.sort(ar);

System.out.println("Sorted array:");

for (int num : ar) {
    System.out.print(num + " ");
}

System.out.println();

System.out.println("Enter the key to search:");

int key = scanner.nextInt();

int result = ternarySearch(0, n - 1, key, ar);

if (result != -1) {
    System.out.println("Index of " + key + " is " + result);
} else {
    System.out.println(key + " is not present in the array.");
}

scanner.close();
}
}

```

Output:

```

C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>javac TernarySearch.java

C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>java TernarySearch
Enter the number of elements in the array:
10
Enter the elements of the array:
12 23 78 81 99 45 38 56 1 66
Sorted array:
1 12 23 38 45 56 66 78 81 99
Enter the key to search:
81
Index of 81 is 8

C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>

```


10.Interpolation Search

Code:

```
import java.util.Scanner;

class InterpolationSearch {

    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) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array:");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array in sorted order:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        System.out.println("Enter the element to search:");
        int x = scanner.nextInt();
        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.");

        scanner.close();

    }

}
```

Output:

```
C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>java InterpolationSearch
Enter the number of elements in the array:
15
Enter the elements of the array in sorted order:
10 12 13 16 18 19 21 22 23 24 33 35 42 47
67
Enter the element to search:
24
Element found at index 9

C:\Users\gowri\OneDrive\Desktop\Practice\Set 8>
```