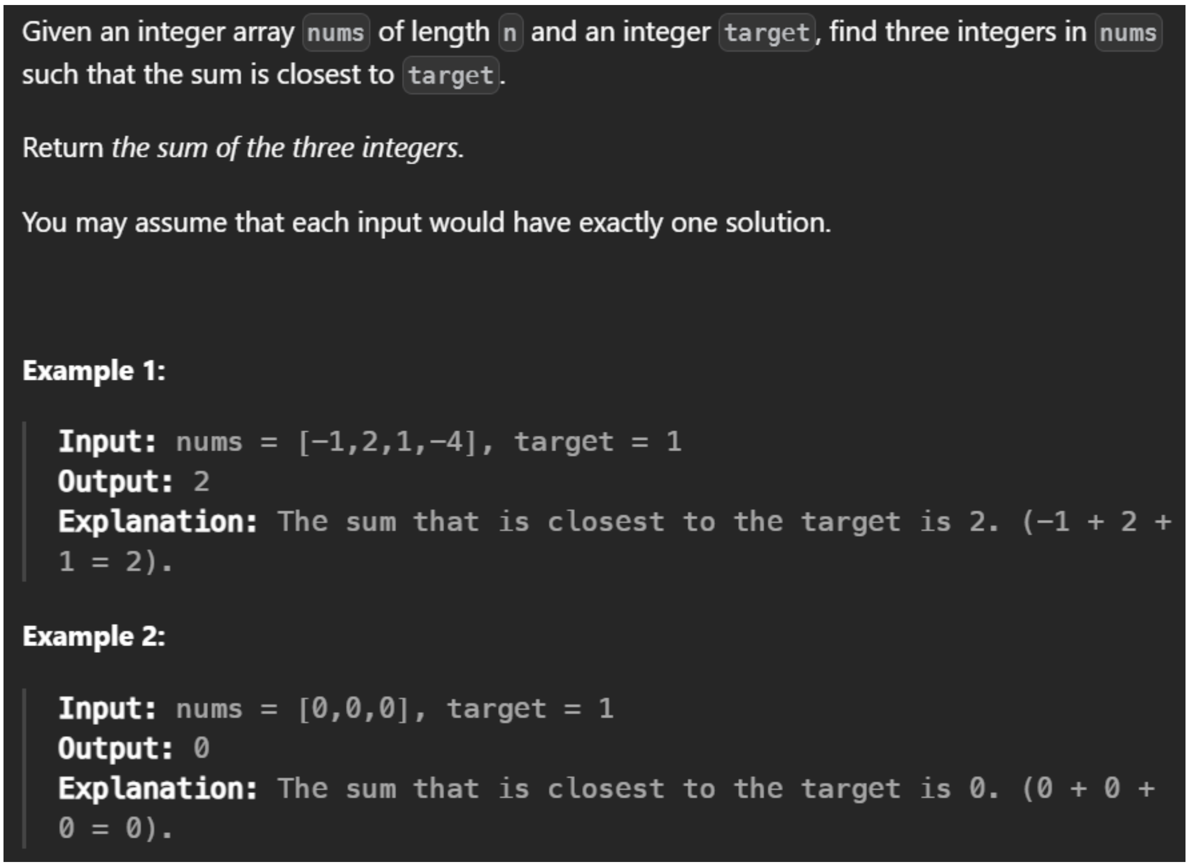
DSA Practice – 8

1. 3Sum 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).
```

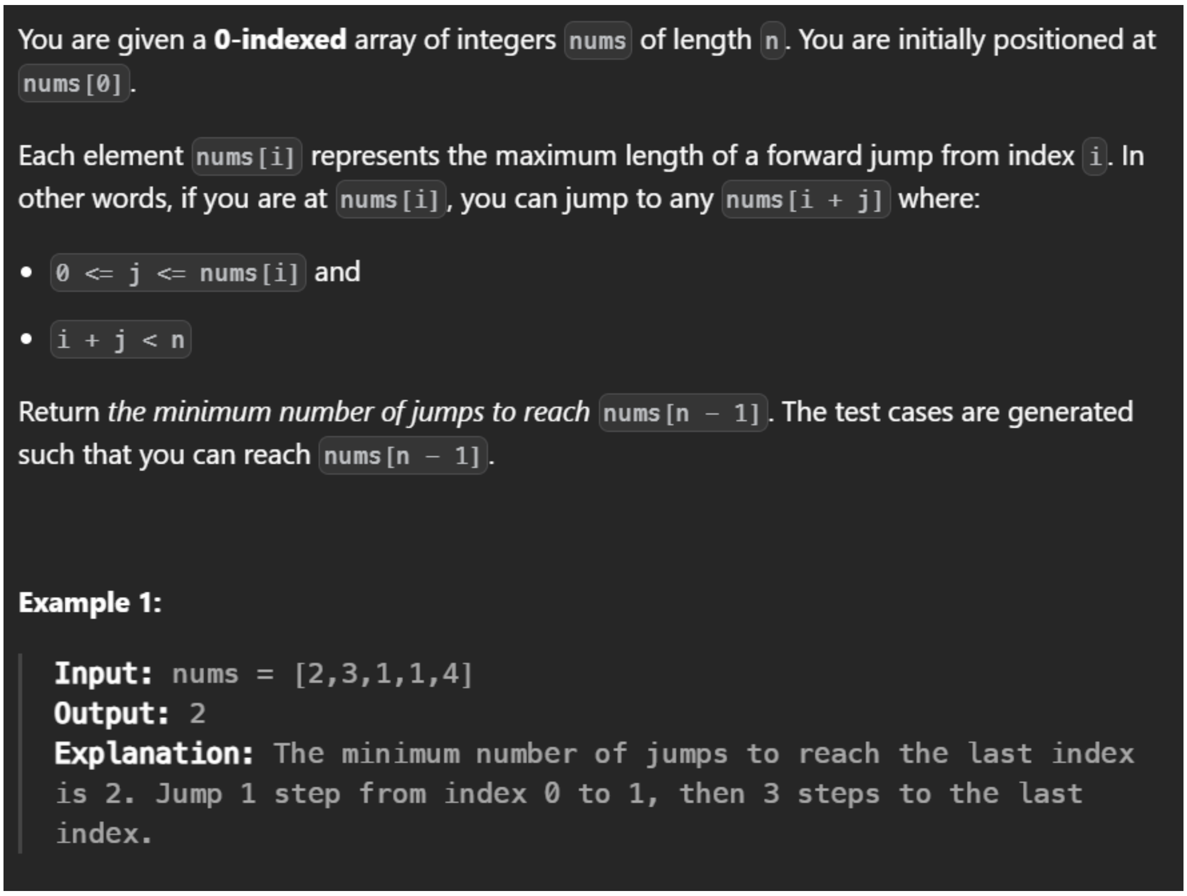
Code:

```
import java.util.Arrays;
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        int closest = Integer.MAX_VALUE;
        Arrays.sort(nums);
        for (int i = 0; i < nums.length - 2; i++) {
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int sum3 = nums[i] + nums[left] + nums[right];
                if (sum3 < target) {
                    left++;
                } else {
                    right--;
                }
                if (Math.abs(sum3 - target) < Math.abs(closest - target)) {
                    closest = sum3;
                }
            }
        }
        return closest;
    }
}
public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums = {-1, 2, 1, -4};
        int target = 1;
        int result = solution.threeSumClosest(nums, target);
        System.out.println("Closest Sum: " + result);
    }
}
```

Output:2

Time Complexity: O (n^2)

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.
```

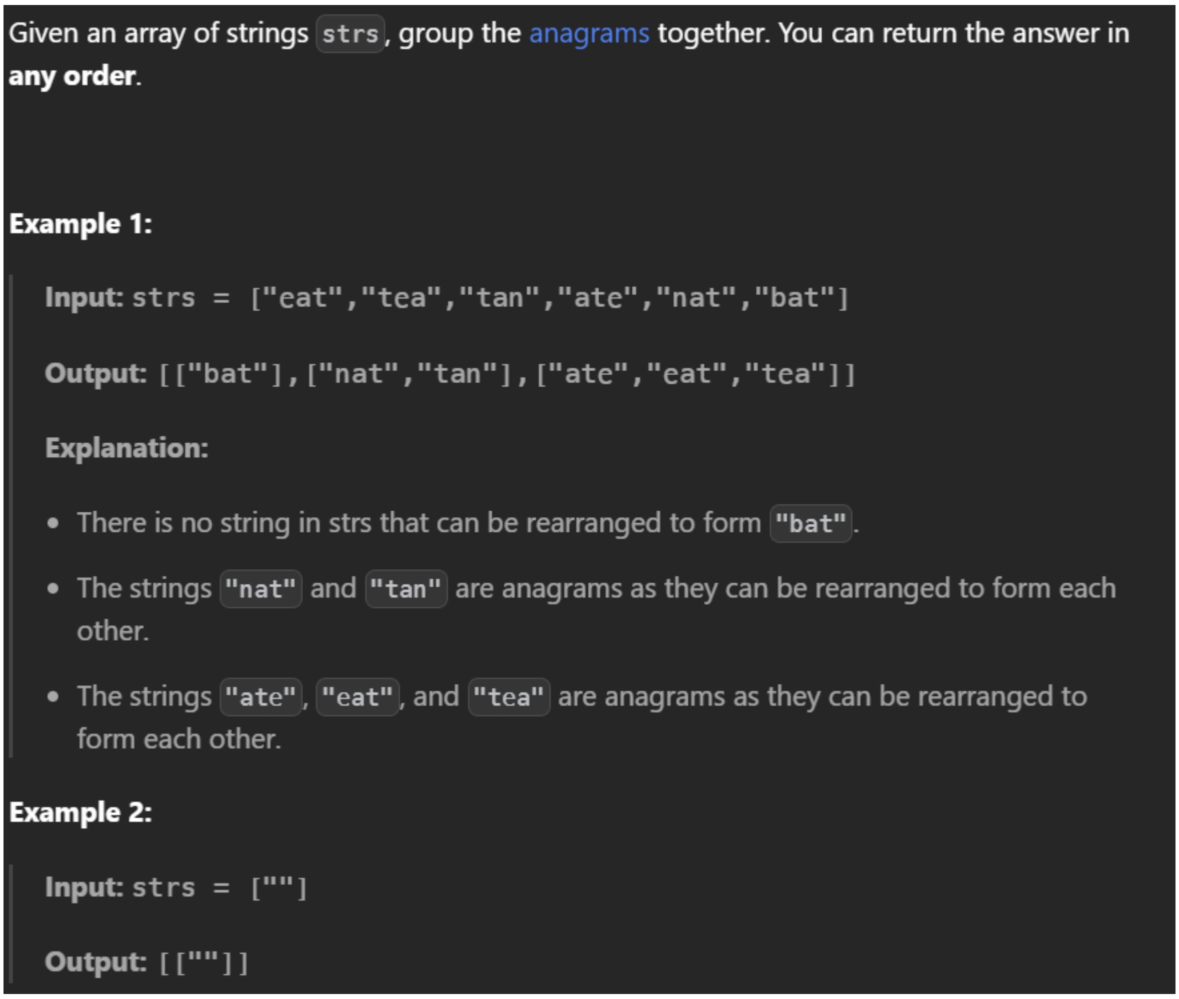
Code:

```
class Solution {
    public int jump(int[] nums) {
        int n = 0, j = 0, jumps = 0;
        while (j < nums.length - 1) {
            int f = 0;
            for (int i = n; i <= j; i++) {
                f = Math.max(f, i + nums[i]);
            }
            n = j + 1;
            j = f;
            jumps++;
        }
        return jumps;
    }
}
public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums = {2, 3, 1, 1, 4};
        int result = solution.jump(nums);
        System.out.println("Minimum Jumps: " + result);
    }
}
```

Output: Minimum Jumps: 2

Time Complexity: O (n^2)

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: `[[""]]`

Code:

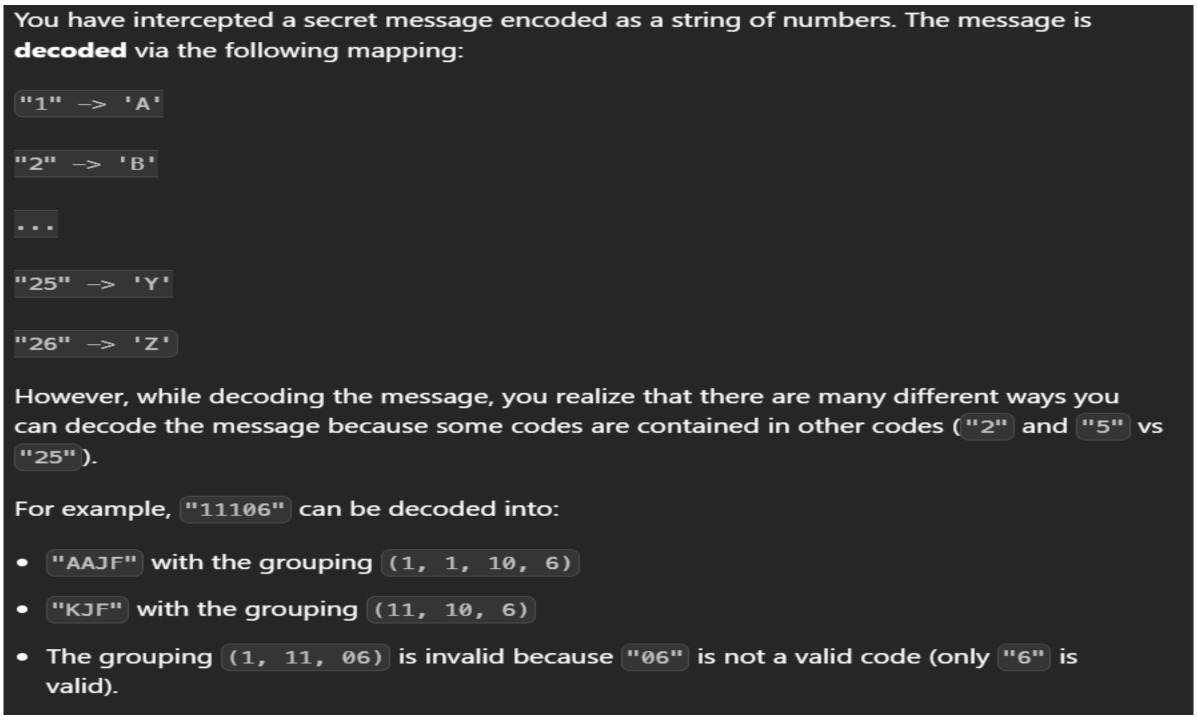
```
import java.util.*;
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> res = new HashMap<>();

        for (String s : strs) {
            char[] chars = s.toCharArray();
            Arrays.sort(chars);
            String key = new String(chars);
            if (!res.containsKey(key)) {
                res.put(key, new ArrayList<>());
            }
            res.get(key).add(s);
        }
        return new ArrayList<>(res.values());
    }
}
public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();
        String[] strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
        List<List<String>> result = solution.groupAnagrams(strs);
        for (List<String> group : result) {
            System.out.println(group);
        }
    }
}
```

Output:

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

Time Complexity: O(n*m)

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'`

`...`

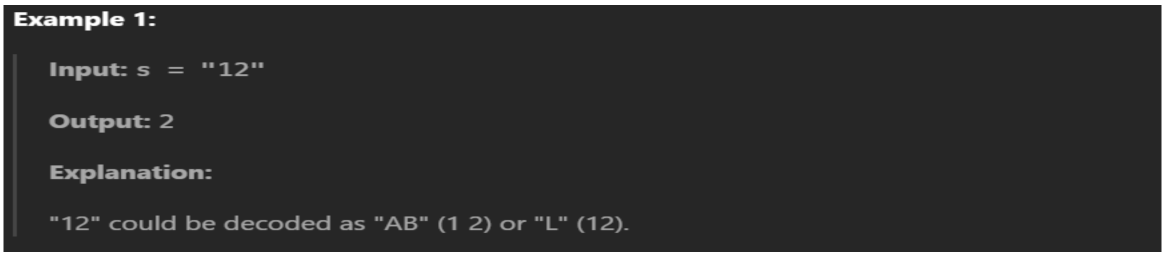
`"25" -> 'Y'`

`"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).



Example 1:

> **Input:** s = "12"
>
> **Output:** 2
>
> **Explanation:**
>
> "12" could be decoded as "AB" (1 2) or "L" (12).

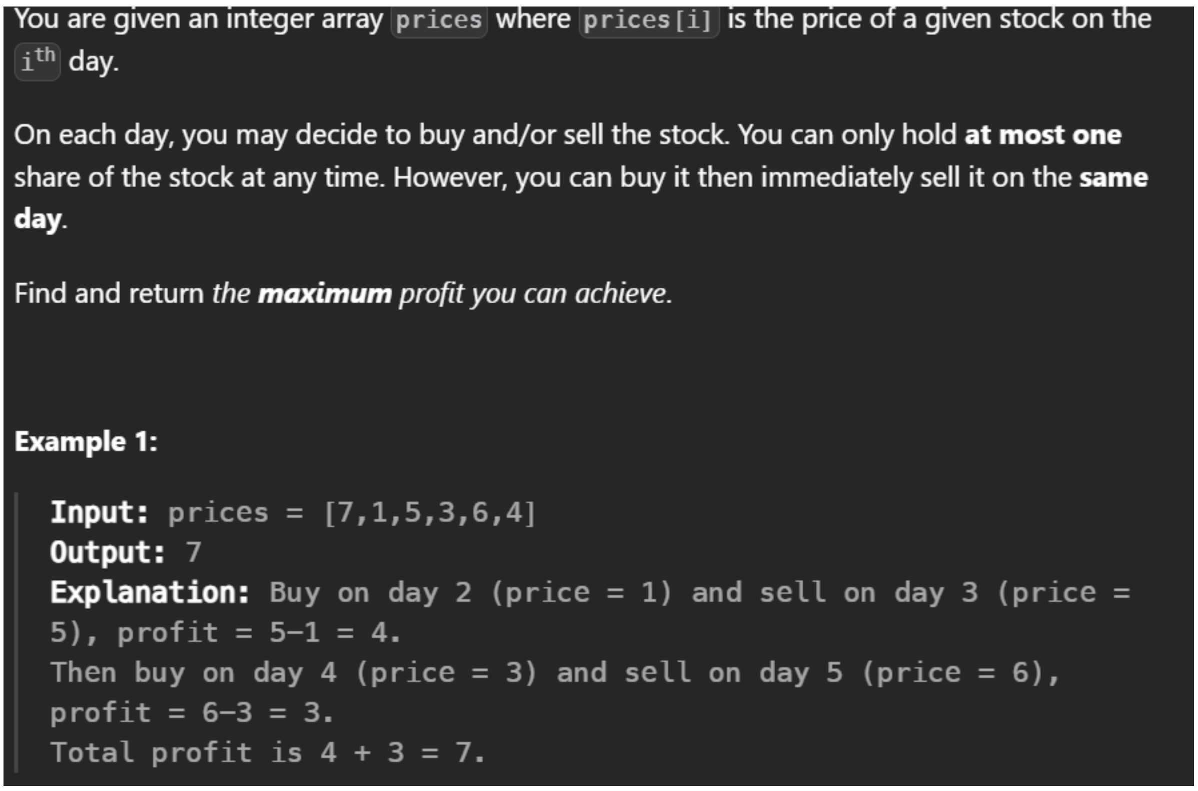
Code:

```
class Solution {
   public int numDecodings(String s) {
      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++) {
         if (s.charAt(i - 1) != '0') {
            dp[i] += dp[i - 1];
         }
         int twoDigit = Integer.parseInt(s.substring(i - 2, i));
         if (twoDigit >= 10 && twoDigit <= 26) {
            dp[i] += dp[i - 2];
         }
      return dp[n];
   }
   public static void main(String[] args) {
      Solution solution = new Solution();
      String s = "12";
      System.out.println(solution.numDecodings(s));
   }
}
```

Output:2

Time Complexity:O(n)

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.
```

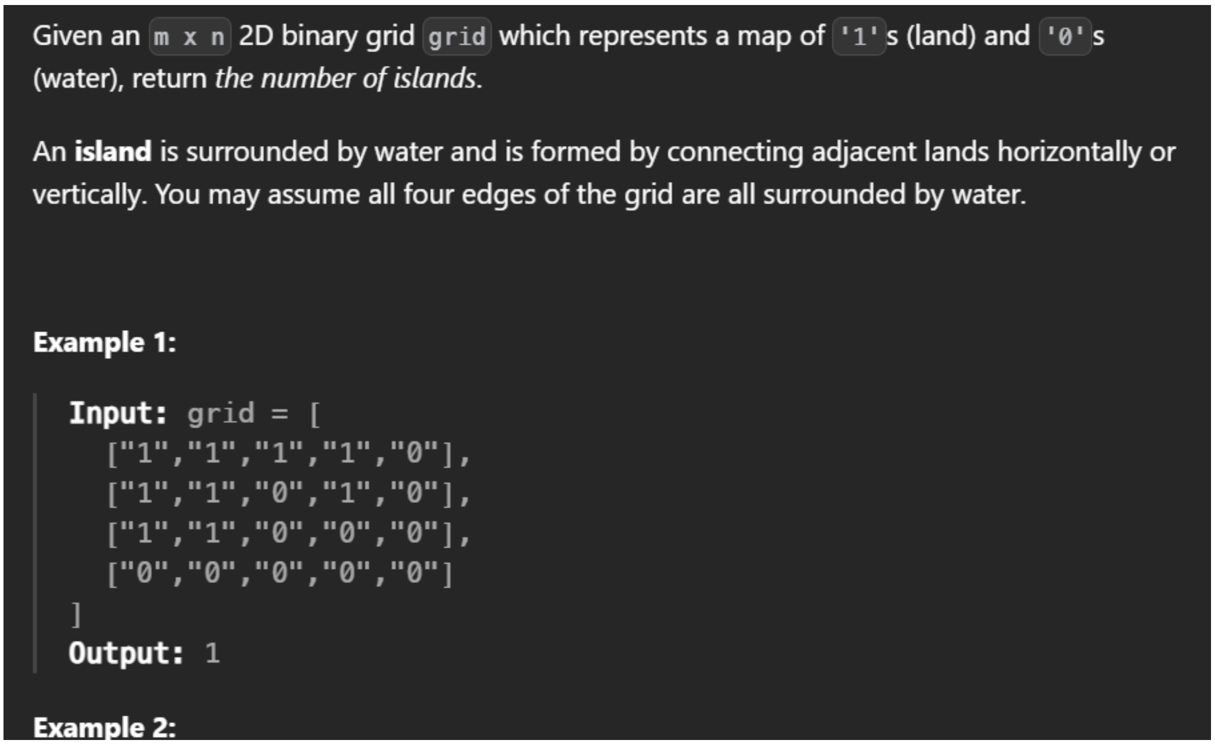
Code:

```
class Solution {
   public int maxProfit(int[] prices) {
      int profit = 0;
      for (int i = 1; i < prices.length; i++) {
         if (prices[i] > prices[i - 1]) {
            profit += prices[i] - prices[i - 1];
         }
      }
      return profit;
   }
}
public class Main {
   public static void main(String[] args) {
      Solution solution = new Solution();
      int[] prices = {7, 1, 5, 3, 6, 4};
      System.out.println(solution.maxProfit(prices));
   }
}
```

Output:7

Time 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:

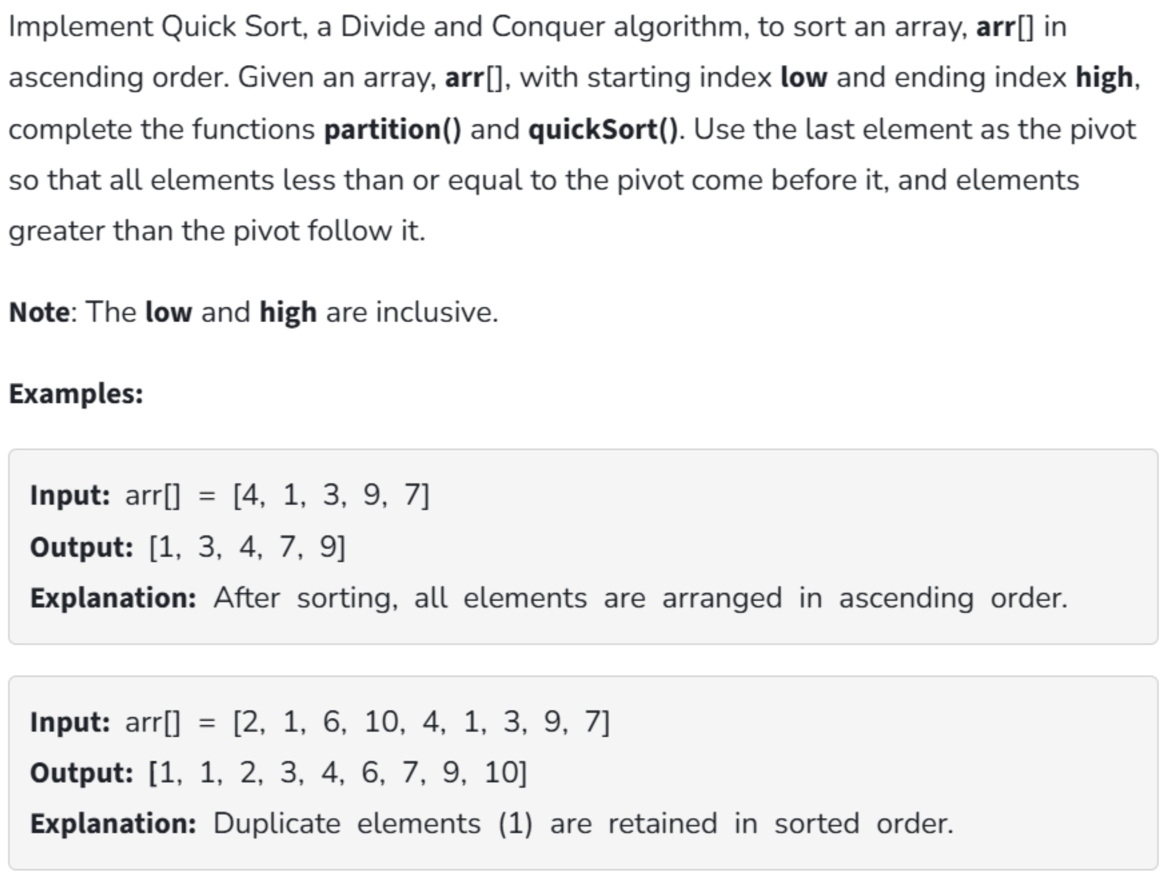
Code:

```
class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int m = grid.length;
        int n = grid[0].length;
        int numIslands = 0;
        int[] dirs = {-1, 0, 1, 0, -1};
        void dfs(int i, int j) {
            if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == '0') {
                return;
            }
            grid[i][j] = '0';
            for (int d = 0; d < 4; d++) {
                int x = i + dirs[d];
                int y = j + dirs[d + 1];
                dfs(x, y);
            }
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == '1') {
                    numIslands++;
                    dfs(i, j);
                }
            }
        }
        return numIslands;
    }
    public static void main(String[] args) {
        Solution sol = new Solution();
        char[][] grid1 = {
            {'1', '1', '1', '1', '0'},
            {'1', '1', '0', '1', '0'},
            {'1', '1', '0', '0', '0'},
            {'0', '0', '0', '0', '0'}
        };
        System.out.println(sol.numIslands(grid1));  // Output: 1
        char[][] grid2 = {
            {'1', '1', '0', '0', '0'},
            {'1', '1', '0', '0', '0'},
            {'0', '0', '1', '0', '0'},
            {'0', '0', '0', '1', '1'}
        };
        System.out.println(sol.numIslands(grid2));
    }
}
```

Output:3

Time Complexity: O (m*n)

7.Quick Sort



Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr**[] in ascending order. Given an array, **arr**[], with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

Note: The **low** and **high** are inclusive.

Examples:

Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
Explanation: After sorting, all elements are arranged in ascending order.

Input: arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]
Output: [1, 1, 2, 3, 4, 6, 7, 9, 10]
Explanation: Duplicate elements (1) are retained in sorted order.

Code:

```
import java.util.Arrays;
class Quick{
    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 + " ");
        }
    }
}
```

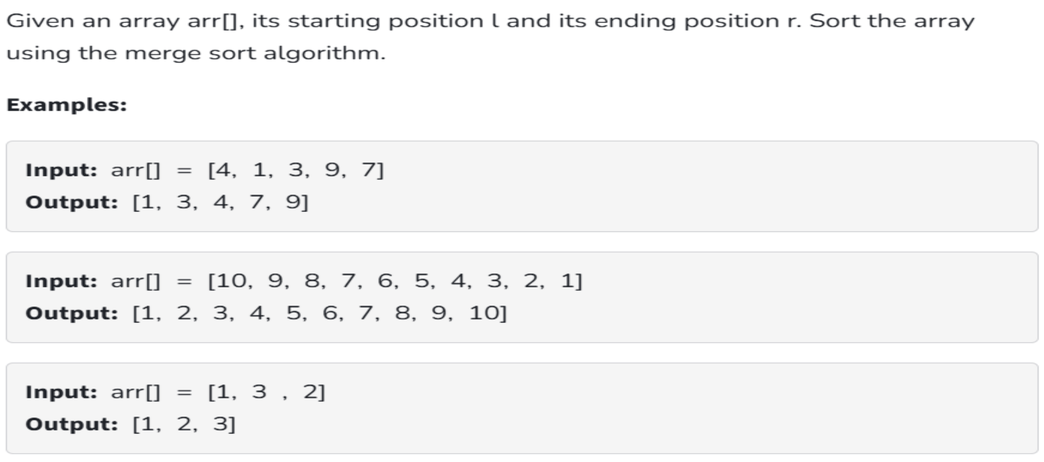
Output:

Sorted Array

1 5 7 8 9 10

Time Complexity:O(nlogn)

8.Merge Sort



Given an array arr[], its starting position l and its ending position r. Sort the array using the merge sort algorithm.

Examples:

```
Input: arr[] = [4, 1, 3, 9, 7]
Output: [1, 3, 4, 7, 9]
```

```
Input: arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
Input: arr[] = [1, 3 , 2]
Output: [1, 2, 3]
```

Code:

```
import java.io.*;
class Merge{
    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 };
        sort(arr, 0, arr.length - 1);
        System.out.println("\nSorted array is");
        printArray(arr);
    }
}
```

Output:

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

Time Complexity:O(nlogn)

9. Ternary Search

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.

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).

Example 1:

```
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 GFG {
  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);
    }
}
```

Output:

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

Time Complexity: O(log$_3$ N)

10. Interpolation Search

Code:

```
import java.util.*;
class Inter {
    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.");
    }
}
```

Output: Element found at index 4

Time Complexity: O(log2(log2 n))