1.3sum_closest
Solution:

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int closest_sum = Integer.MAX_VALUE / 2;  // A large value but not overflow

        for (int i = 0; i < nums.length - 2; ++i) {
            int left = i + 1, right = nums.length - 1;
            while (left < right) {
                int current_sum = nums[i] + nums[left] + nums[right];
                if (Math.abs(current_sum - target) < Math.abs(closest_sum - target)) {
                    closest_sum = current_sum;
                }
                if (current_sum < target) {
                    ++left;
                } else if (current_sum > target) {
                    --right;
                } else {
                    return current_sum;
                }
            }
        }

        return closest_sum;
    }
}
```

output:
2

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

2.Jump game -2

Solution:

```
class Solution {
    public int jump(int[] nums) {
        int near = 0, far = 0, jumps = 0;

        while (far < nums.length - 1) {
            int farthest = 0;
            for (int i = near; i <= far; i++) {
```

```
                farthest = Math.max(farthest, i + nums[i]);
            }
            near = far + 1;
            far = farthest;
            jumps++;
        }

        return jumps;
    }
}
```

output:
2

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

3.Group Anagrams

Solution:

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

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

Time Complexity:O(m*nlogn)
Space Complexity:O(mn)

4.Decoding ways
Solution:

```
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:
3

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

5.Best Time to Buy and sell -II

Solution:

```
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;
    }
}
```

Output:
7

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

6.Number of islands

Soution:

```
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:
Number

Time Complexity:O(r*c)
Space Complexity:O(r*c)

7.Quick sort

Solution:

```
import java.util.Arrays;

class GfG {
    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(n^2)
Space Complexity:O(n)

8.Merge sort

Solution:

```
import java.io.*;

class GfG {
    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

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

9.Ternary Search

Solution:

```
class GFG {

    static int ternarySearch(int l, int r, int key, int ar[])

    {
        while (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]) {

            r = mid1 - 1;
        }
        else if (key > ar[mid2]) {

            l = mid2 + 1;
        }
        else {

            l = mid1 + 1;
            r = mid2 - 1;
        }
    }

    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);

        // 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(1)

10.Interpolation Search

Solution:

```
import java.util.*;

class GFG {

        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 x is smaller, x is in left sub array
                        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))
Space Complexity:O(1)