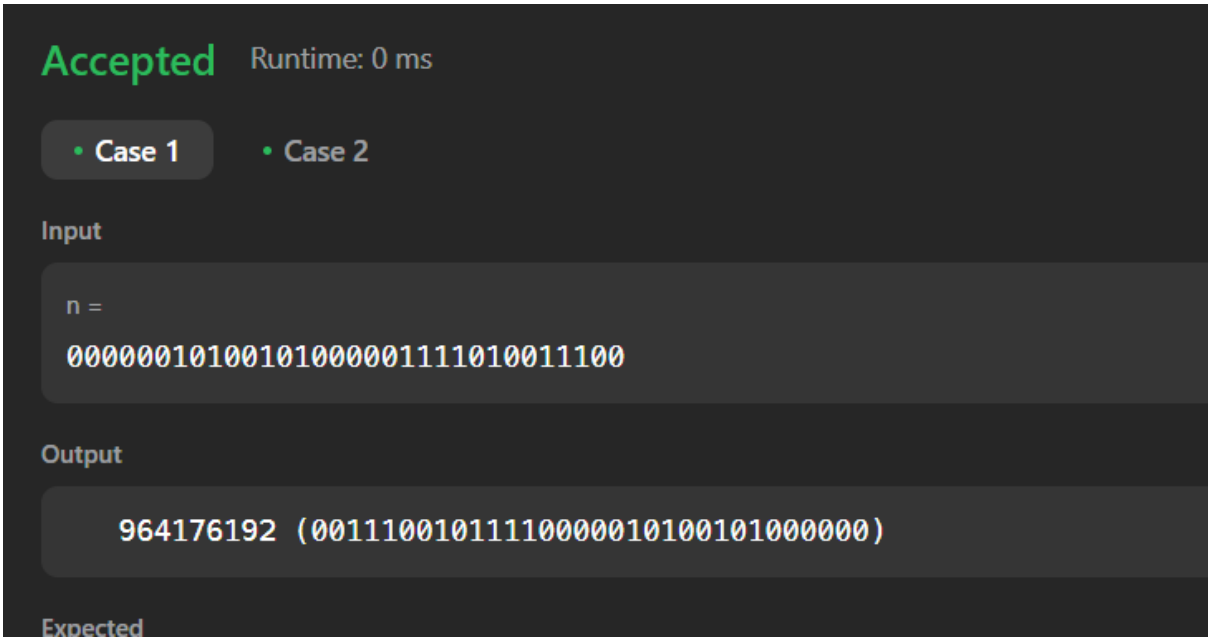
1.

```
class Solution {
    public String longestNiceSubstring(String s) {
        int n = s.length();
        if (n < 2) return "";

        for (int i = 0; i < n; i++) {
            char ch = s.charAt(i);
            if (s.contains(Character.toString(Character.toUpperCase(ch))) &&
                s.contains(Character.toString(Character.toLowerCase(ch)))) {
                continue;
            }

            String left = longestNiceSubstring(s.substring(0, i));
            String right = longestNiceSubstring(s.substring(i + 1));

            return left.length() >= right.length() ? left : right;
        }

        return s;
    }
}
```

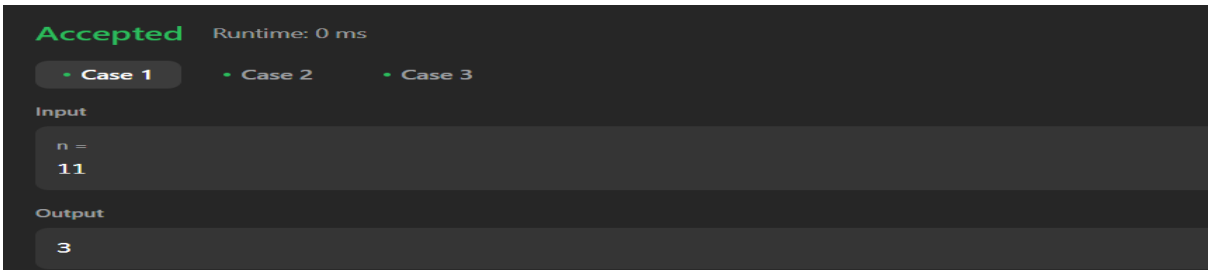
2.

```
public class Solution {

    public int reverseBits(int n) {
        int result = 0;
        for (int i = 0; i < 32; i++) {
            result <<= 1;
            result |= (n & 1);
            n >>= 1;
        }
        return result;
    }
}
```



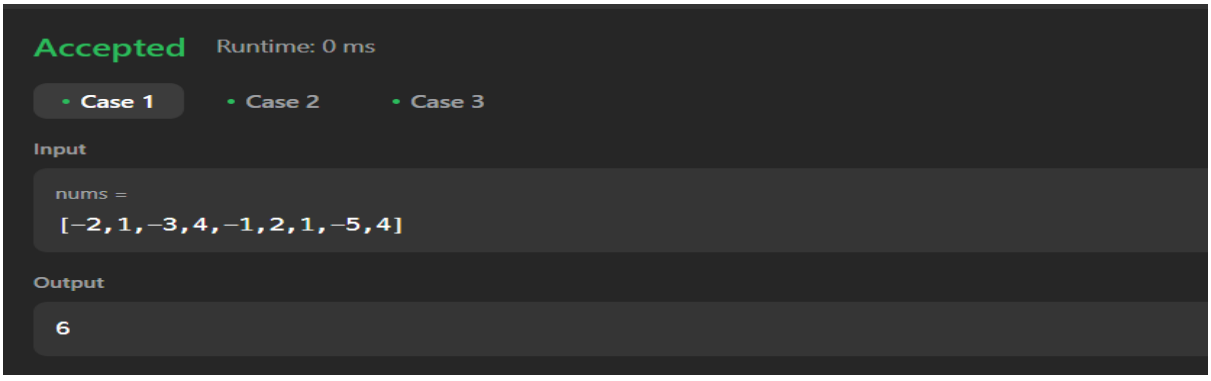
3.

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```



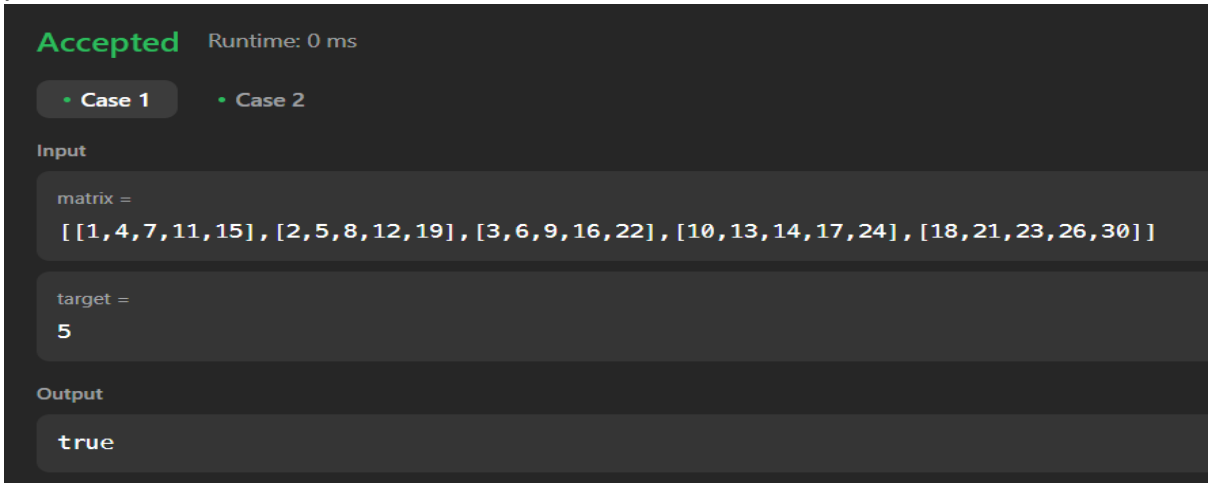
4.

```
public class Solution {

    public int maxSubArray(int[] nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```



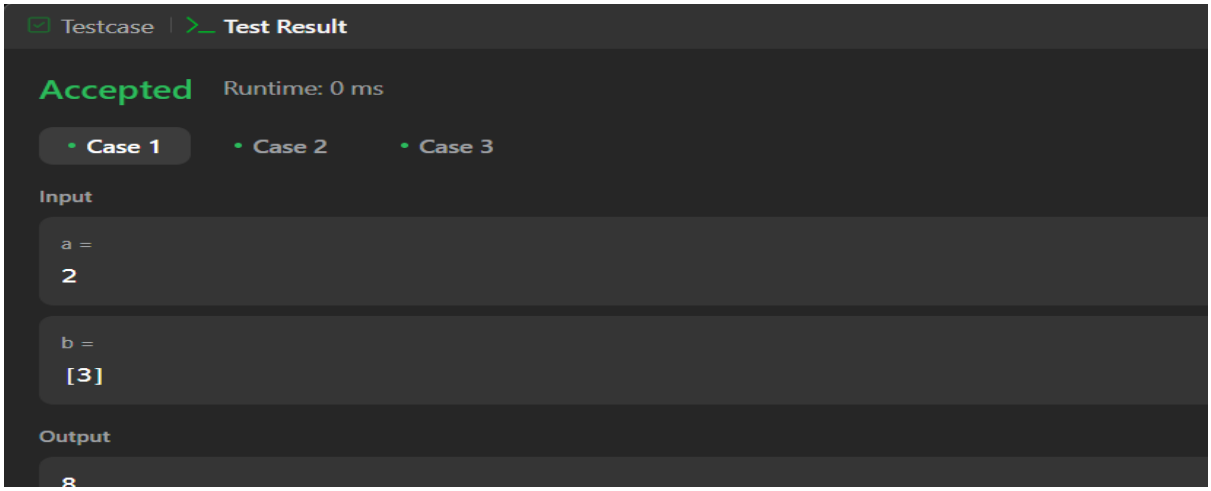
5.

```
public class Solution {


    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return false;
        }
        int row = 0;
        int col = matrix[0].length - 1;
        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }
        return false;
    }
}
```



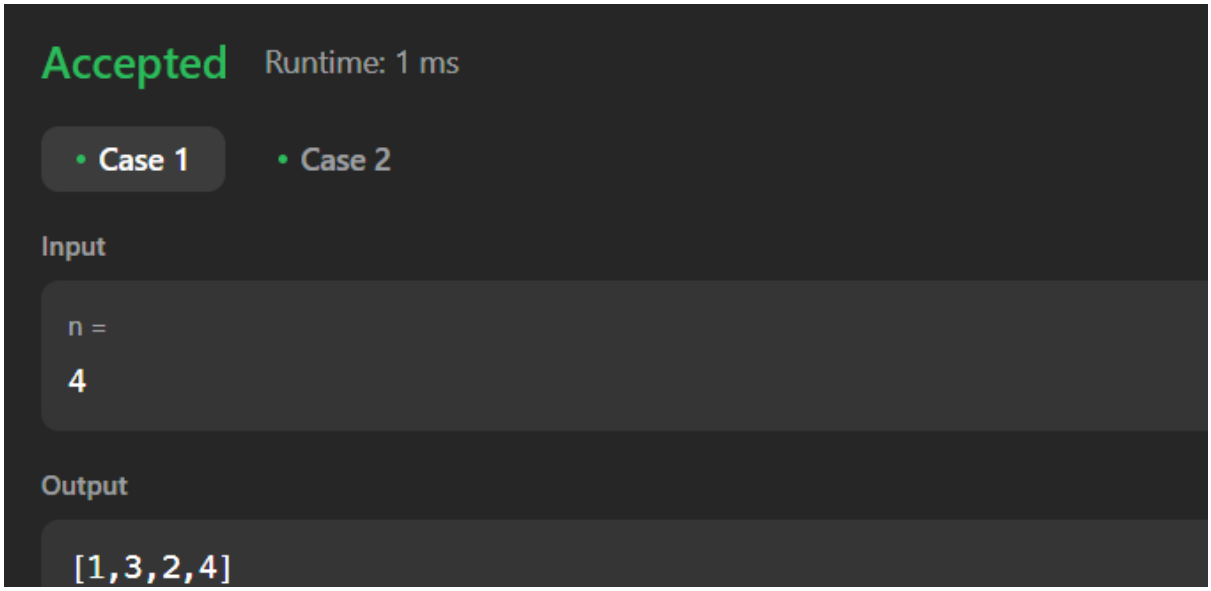
6.

```
public class Solution {
    public int superPow(int a, int[] b) {
        int mod = 1337;
        a %= mod;
        int result = 1;
        for (int i = 0; i < b.length; i++) {
            result = (powMod(result, 10, mod) * powMod(a, b[i], mod)) % mod;
        }
        return result;
    }

    private int powMod(int x, int y, int mod) {
        int res = 1;
        while (y > 0) {
            if (y % 2 == 1) {
                res = (res * x) % mod;
            }
            x = (x * x) % mod;
            y /= 2;
        }
        return res;
    }
}
```



7.

```
public class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> res = new ArrayList<>();
        res.add(1);
        while (res.size() < n) {
            List<Integer> next = new ArrayList<>();
            for (int num : res) {
                if (num * 2 - 1 <= n) {
                    next.add(num * 2 - 1);
                }
            }
            for (int num : res) {
                if (num * 2 <= n) {
                    next.add(num * 2);
                }
            }
            res = next;
        }
        return res.stream().mapToInt(i -> i).toArray();
    }
}
```



8.

```java
import java.util.*;

public class Solution {


    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> points = new ArrayList<>();
        for (int[] b : buildings) {
            points.add(new int[]{b[0], -b[2]}); // Start of a building
            points.add(new int[]{b[1], b[2]}); // End of a building
        }
        points.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

        List<List<Integer>> result = new ArrayList<>();
        PriorityQueue<Integer> heights = new PriorityQueue<>(Collections.reverseOrder());
        heights.add(0);
        int prevMax = 0;

        for (int[] p : points) {
            if (p[1] < 0) {
                heights.add(-p[1]);
            } else {
                heights.remove(p[1]);
            }

            int currentMax = heights.peek();
            if (currentMax != prevMax) {
                result.add(Arrays.asList(p[0], currentMax));
                prevMax = currentMax;
            }
        }
        return result;
    }
}
```

Testcase | Test Result

Accepted Runtime: 1 ms

• Case 1 • Case 2

Input

buildings =

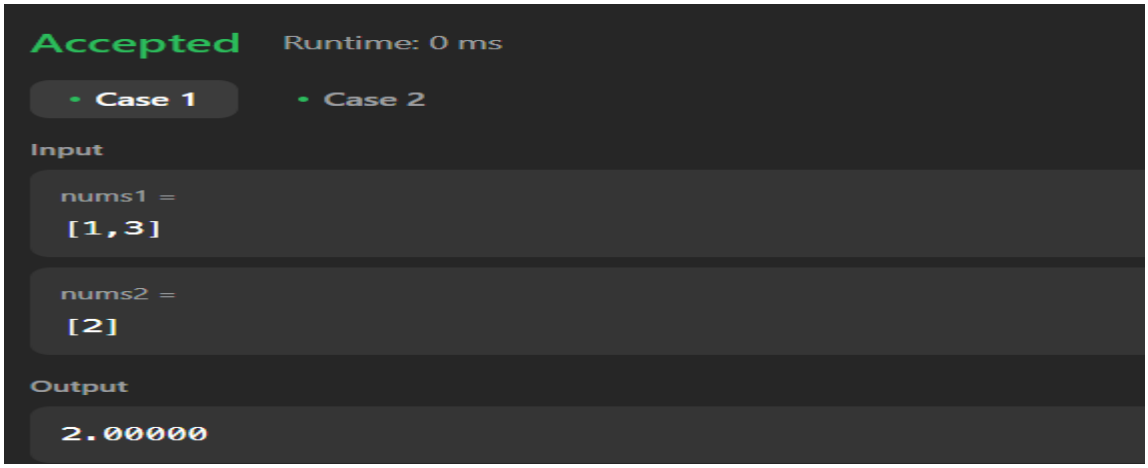
```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Output

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

9.

```
import java.util.*;

public class Solution {


    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }
        int x = nums1.length, y = nums2.length;
        int low = 0, high = x;

        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minRightX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minRightY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (Math.max(maxLeftX, maxLeftY) + Math.min(minRightX, minRightY)) / 2.0;
                } else {
                    return Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
            }
        }
        throw new IllegalArgumentException();
    }
}
```



10.

```
import java.util.*;

public class Solution {

    public int search(int[] nums, int target) {
    int left = 0, right = nums.length - 1;

    while (left <= right) {
      int mid = left + (right - left) / 2;

      if (nums[mid] == target) {
        return mid;
      }

      if (nums[left] <= nums[mid]) {
        if (nums[left] <= target && target < nums[mid]) {
          right = mid - 1;
        } else {
          left = mid + 1;
        }
      } else {
        if (nums[mid] < target && target <= nums[right]) {
          left = mid + 1;
        } else {
          right = mid - 1;
        }
      }
    }
    return -1;
  }
}
```