

Name – Chirag Yadav

Uid –22BCS16377

Sec – 608-B

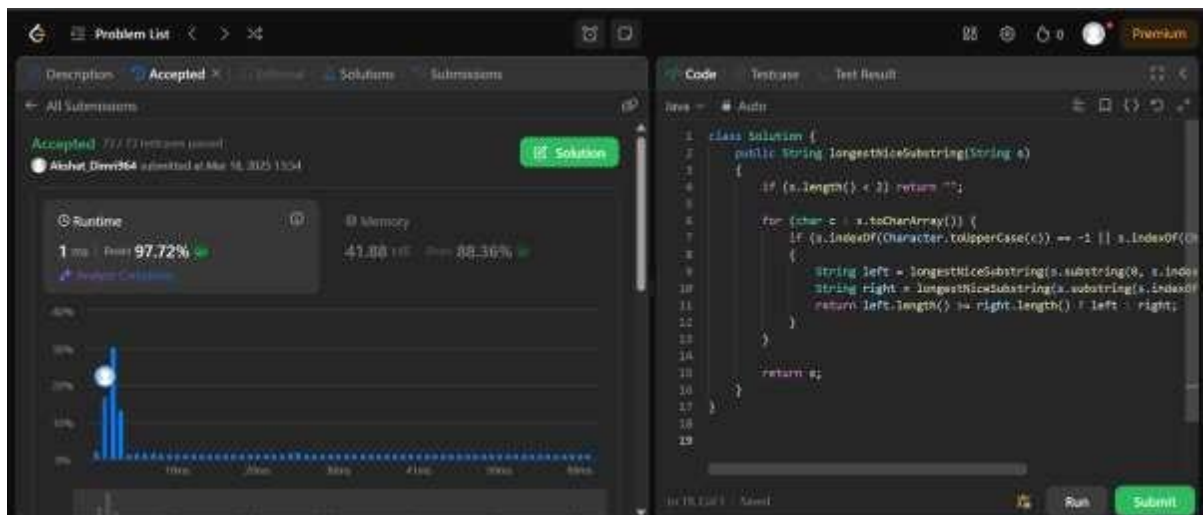
AP Assignment – 04

Q1- Longest Nice Substring

Code –

```
class Solution {
    public String longestNiceSubstring(String s)
    {
        if (s.length() < 2) return "";
        for (char c : s.toCharArray()) {
            if (s.indexOf(Character.toUpperCase(c)) == -1 || s.indexOf(Character.toLowerCase(c))
== -1)
            {
                String left = longestNiceSubstring(s.substring(0, s.indexOf(c)));
                String right = longestNiceSubstring(s.substring(s.indexOf(c) + 1));
                return left.length() >= right.length() ? left : right;
            }
        }

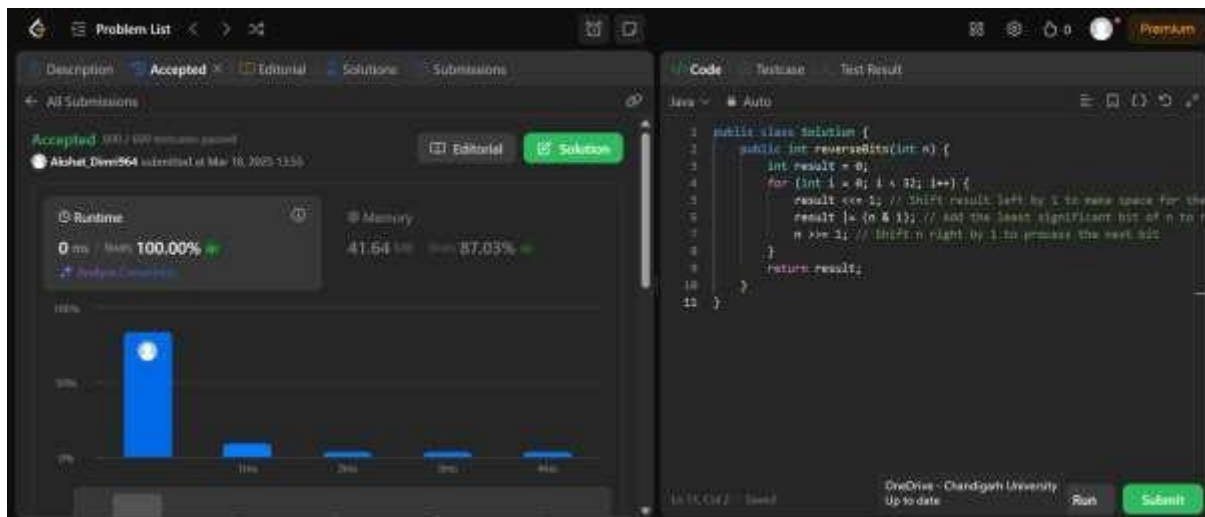
        return s;
    }
}
```



Q2-Reverse bits of a given 32 bits unsigned integer.

Code –

```
public class Solution {
    public int reverseBits(int n) {
        int result = 0;
        for (int i = 0; i < 32; i++) {
            result <<= 1; // Shift result left by 1 to make space for the next bit
            result |= (n & 1); // Add the least significant bit of n to result
            n >>= 1; // Shift n right by 1 to process the next bit
        }
        return result;
    }
}
```



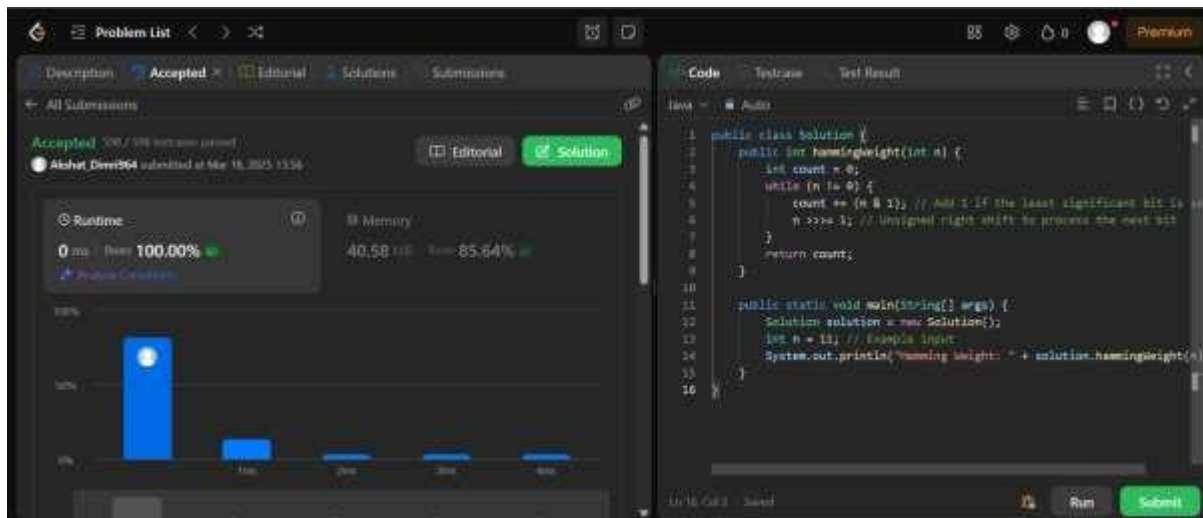
Q3 - Given a positive integer  $n$ , write a function that returns the number of set bits in its binary representation (also known as the [Hamming weight](#)).

Code –

```
public class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1); // Add 1 if the least significant bit is set
            n >>= 1; // Unsigned right shift to process the next bit
        }
        return count;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int n = 11; // Example input
        System.out.println("Hamming Weight: " + solution.hammingWeight(n));
    }
}
```

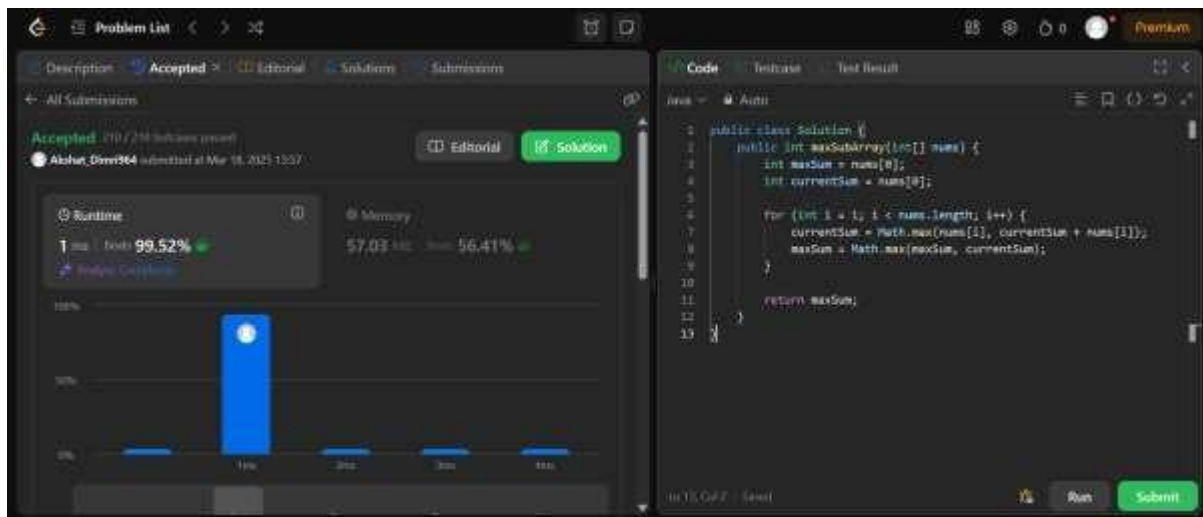
}



Q4 – Given an integer array *nums*, find the subarray with the largest sum, and return *its sum*.

Code –

```
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;  
    }  
}
```



Q5 - Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Code –

```

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;
    }
}

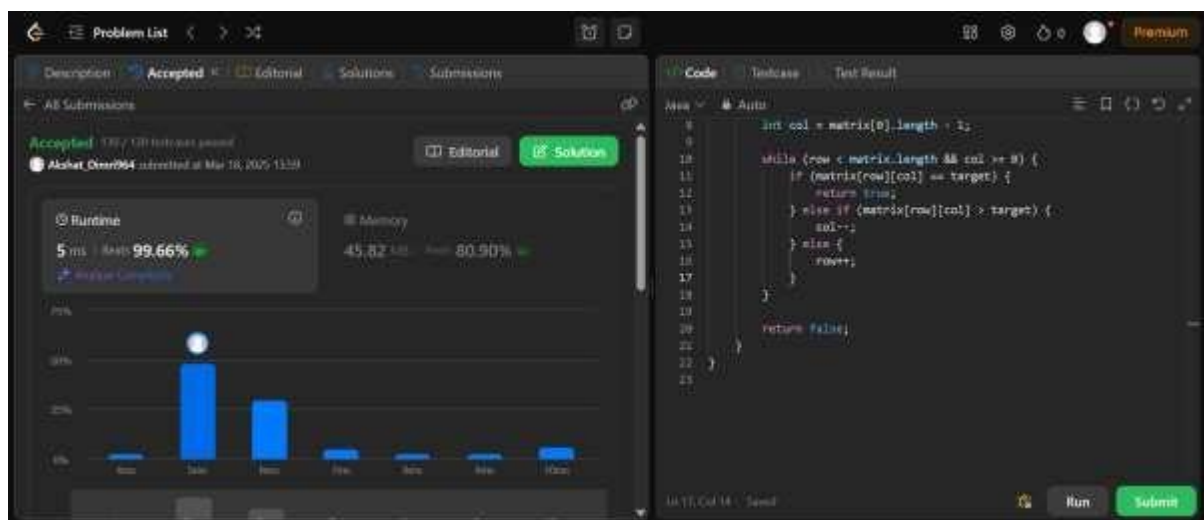
```

```

        row++;
    }
}

return false;
}
}

```



Q6 – Your task is to calculate  $a^b \bmod 1337$  where  $a$  is a positive integer and  $b$  is an extremely large positive integer given in the form of an array.

Code –

```

public class Solution {
    private static final int MOD = 1337;

    public int superPow(int a, int[] b) {
        a %= MOD;
        return powerMod(a, b, b.length);
    }
}

```

```

private int powerMod(int a, int[] b, int length) {
    if (length == 0) return 1;

    int lastDigit = b[length - 1];
    int[] newB = new int[length - 1];
    System.arraycopy(b, 0, newB, 0, length - 1);

    int part1 = modExp(a, lastDigit);
    int part2 = modExp(powerMod(a, newB, length - 1), 10);

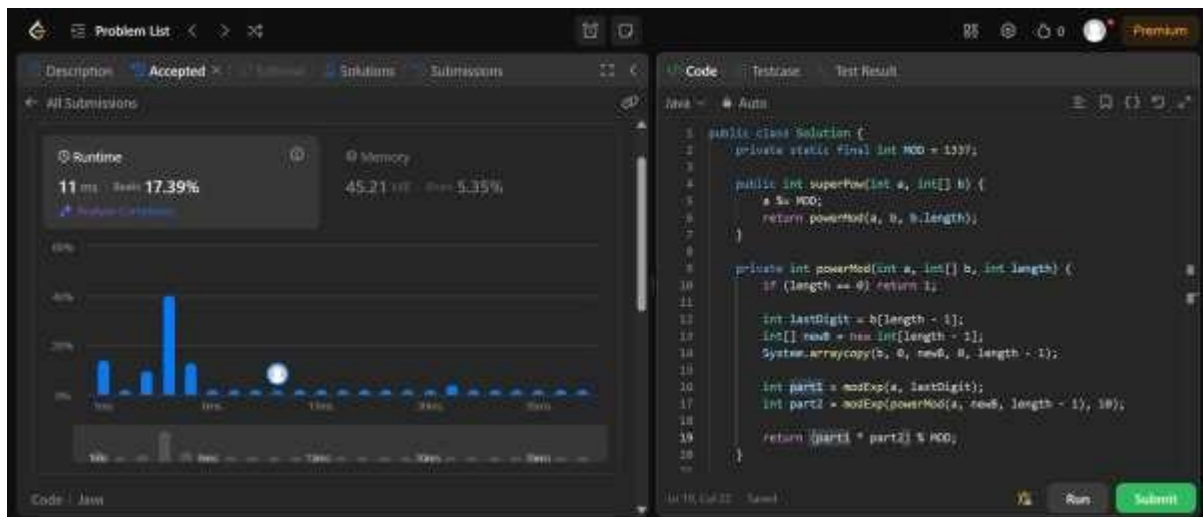
    return (part1 * part2) % MOD;
}

private int modExp(int base, int exp) {
    int result = 1;
    base %= MOD;

    for (int i = 0; i < exp; i++) {
        result = (result * base) % MOD;
    }

    return result;
}
}

```



Q7 - Given the integer  $n$ , return *any beautiful array* nums of length  $n$ . There will be at least one valid answer for the given  $n$ .

Code-

```
import java.util.*;
```

```
public class Solution {

    public int[] beautifulArray(int n) {

        List<Integer> result = new ArrayList<>();
        result.add(1);

        while (result.size() < n) {

            List<Integer> next = new ArrayList<>();

            for (int num : result) {

                if (num * 2 - 1 <= n) next.add(num * 2 - 1);

            }

            for (int num : result) {

                if (num * 2 <= n) next.add(num * 2);

            }

            result = next;

        }

    }

}
```



```

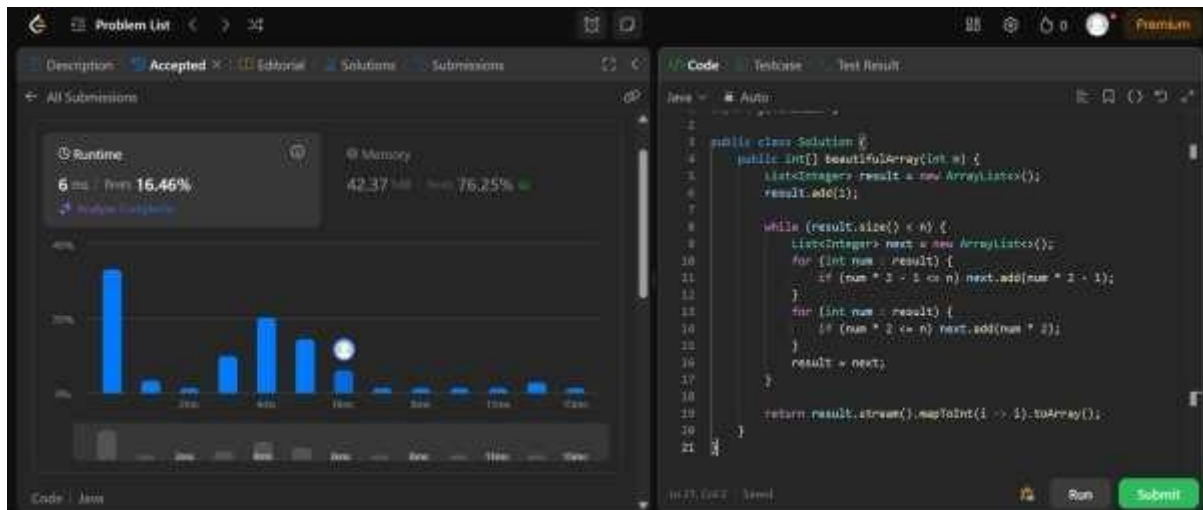
    }

    return result.stream().mapToInt(i -> i).toArray();

}

}

```



Q8 – A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the skyline formed by these buildings collectively*.

Code –

```
import java.util.*;
```

```

public class Solution {

    public List<List<Integer>> getSkyline(int[][] buildings) {

        List<int[]> points = new ArrayList<>();

        for (int[] building : buildings) {

            points.add(new int[]{building[0], -building[2]}); // Start of building

            points.add(new int[]{building[1], building[2]}); // End of building

        }
    }
}

```

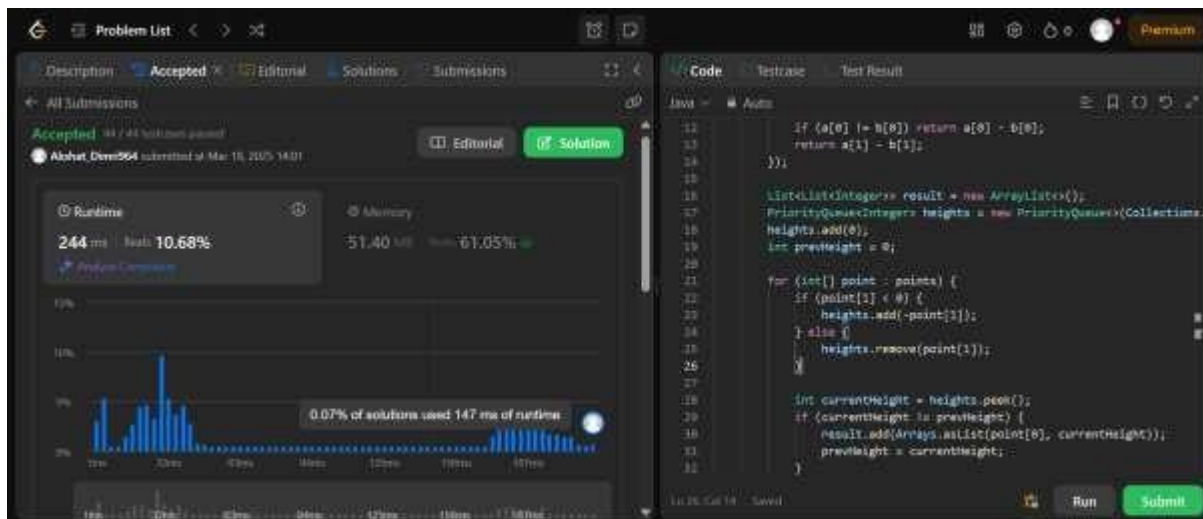
```
Collections.sort(points, (a, b) -> {  
    if (a[0] != b[0]) return a[0] - b[0];  
    return a[1] - b[1];  
});
```

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

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

```
int currentHeight = heights.peek();  
if (currentHeight != prevHeight) {  
    result.add(Arrays.asList(point[0], currentHeight));  
    prevHeight = currentHeight;  
}  
}
```

```
return result;  
}  
}
```



Q9 - Given an integer array *nums*, return *the number of **reverse pairs*** in the array.

Code –

```
public class Solution {

    public int reversePairs(int[] nums) {

        if (nums == null || nums.length < 2) return 0;

        return mergeSort(nums, 0, nums.length - 1);

    }

    private int mergeSort(int[] nums, int left, int right) {

        if (left >= right) return 0;

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

        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

        int j = mid + 1;

        for (int i = left; i <= mid; i++) {

            while (j <= right && nums[i] > 2L * nums[j]) j++;

            count += (j - (mid + 1));

        }

    }

}
```

```

merge(nums, left, mid, right);

return count;
}

```

```

private void merge(int[] nums, int left, int mid, int right) {

```

```

    int[] temp = new int[right - left + 1];

```

```

    int i = left, j = mid + 1, k = 0;

```

```

    while (i <= mid && j <= right) {
        if (nums[i] <= nums[j]) temp[k++] = nums[i++];
        else temp[k++] = nums[j++];
    }

```

```

    while (i <= mid) temp[k++] = nums[i++];

```

```

    while (j <= right) temp[k++] = nums[j++];

```

```

    System.arraycopy(temp, 0, nums, left, temp.length);

```

```

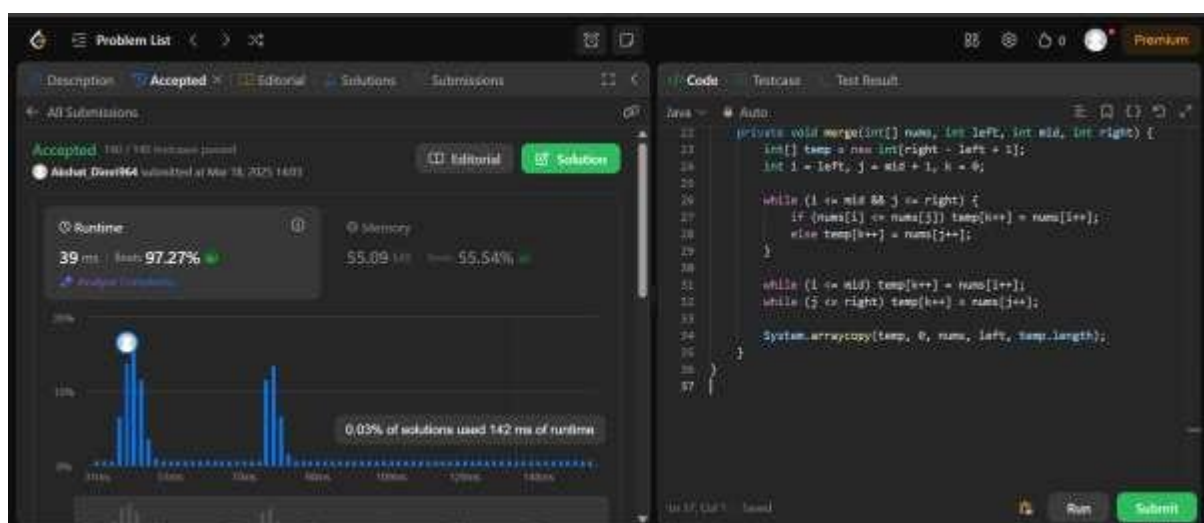
}

```

```

}

```



Q10 - You are given an integer array `nums` and an integer `k`.

Find the longest subsequence of `nums` that meets the following requirements:

- The subsequence is **strictly increasing** and
- The difference between adjacent elements in the subsequence is **at most** `k`.

Return *the length of the **longest subsequence** that meets the requirements*.

Code –

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        map<int, int> dp;
        int maxLen = 0;
        for (int num : nums) {
            int maxPrev = 0;
            for (int i = num - k; i <= num - 1; i++) {
                maxPrev = max(maxPrev, dp[i]);
            }
            dp[num] = maxPrev + 1;
            maxLen = max(maxLen, dp[num]);
        }
        return maxLen;
    }
};
```

Accepted 84 / 84 testcases passed

Akshat\_Dimri964 submitted at Mar 18, 2025 14:07

[Solution](#)

Runtime

83 ms | Beats 60.70%

[Analyze Complexity](#)

Memory

63.47 MB | Beats 60.29%

