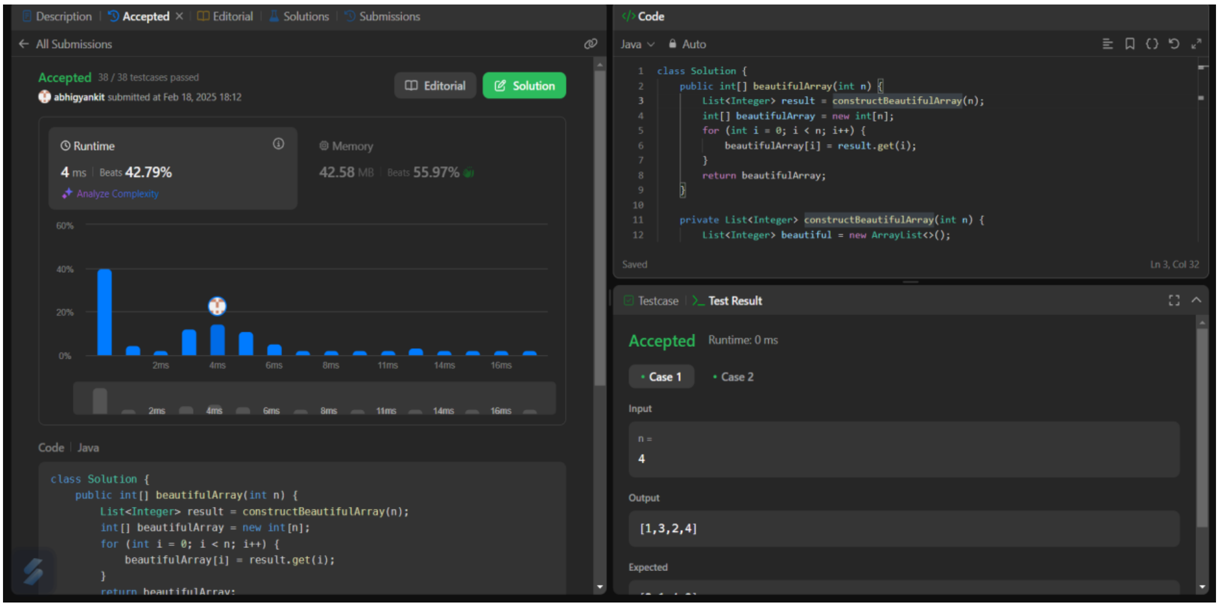
Assignment 4

Name: Abhigyan	**Uid: 22BCS10097**
Branch: BE_CSE	**Semester: 6th**
Section: IOT_637-B	**Subject: AP Lab II**

932. Beautiful Array

```
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = constructBeautifulArray(n);
        int[] beautifulArray = new int[n];
        for (int i = 0; i < n; i++) {
            beautifulArray[i] = result.get(i);
        }
        return beautifulArray;
    }
    private List<Integer> constructBeautifulArray(int n) {
        List<Integer> beautiful = new ArrayList<>();
        beautiful.add(1); // Base case for recursion
        while (beautiful.size() < n) {
            List<Integer> temp = new ArrayList<>();
            for (int num : beautiful) {
                if (2 * num - 1 <= n) {
                    temp.add(2 * num - 1);
                }
            }
            for (int num : beautiful) {
                if (2 * num <= n) {
                    temp.add(2 * num);
                }
            }
            beautiful = temp;
        }
        return beautiful;
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        int n = 5;
        int[] result = solution.beautifulArray(n);
        System.out.println("Beautiful Array for n = " + n + ": " +
            Arrays.toString(result));
    }
}
```



Accepted 38 / 38 testcases passed

abhigyankit submitted at Feb 18, 2025 18:12

Runtime 4 ms | Beats 42.79%

Memory 42.58 MB | Beats 55.97%

Code | Java

```
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = constructBeautifulArray(n);
        int[] beautifulArray = new int[n];
        for (int i = 0; i < n; i++) {
            beautifulArray[i] = result.get(i);
        }
        return beautifulArray;
```

```
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = constructBeautifulArray(n);
        int[] beautifulArray = new int[n];
        for (int i = 0; i < n; i++) {
            beautifulArray[i] = result.get(i);
        }
        return beautifulArray;
    }

    private List<Integer> constructBeautifulArray(int n) {
        List<Integer> beautiful = new ArrayList<>();
```

Saved

Ln 3, Col 32

Accepted Runtime: 0 ms

Input

n =

4

Output

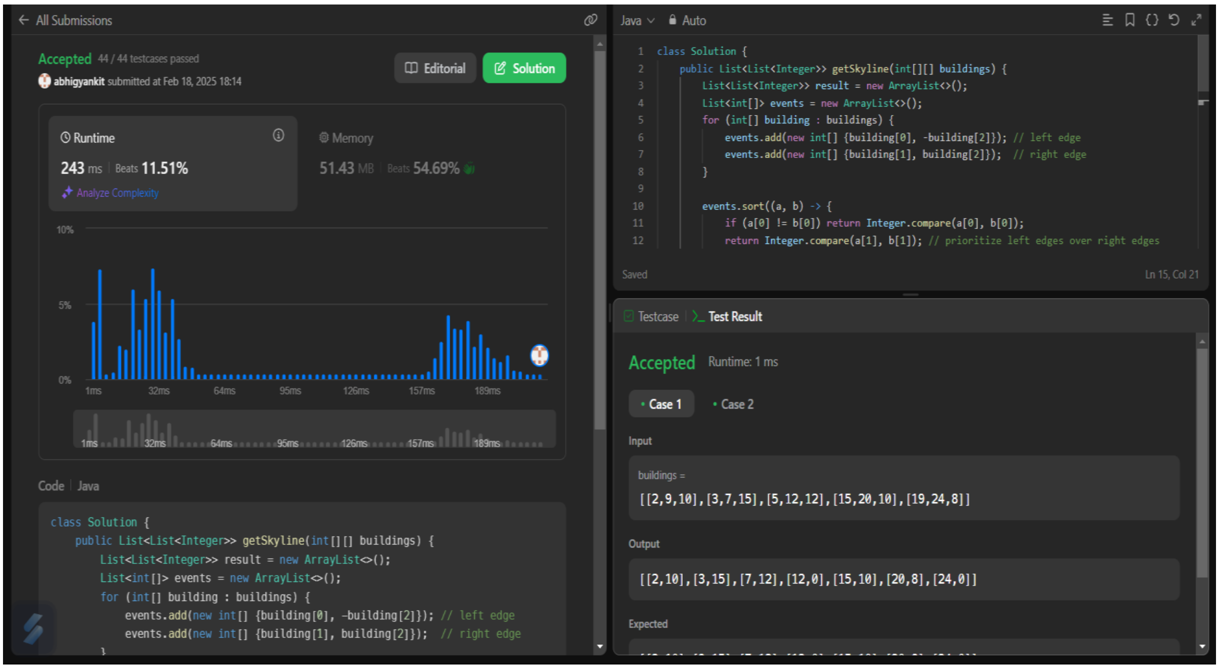
```
[1,3,2,4]
```

Expected

218. The Skyline Problem

```
class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<List<Integer>> result = new ArrayList<>();
        List<int[]> events = new ArrayList<>();
        for (int[] building : buildings) {
            events.add(new int[] {building[0], -building[2]}); //
left edge
            events.add(new int[] {building[1], building[2]});  //
right edge
```

```
        }
        events.sort((a, b) -> {
            if (a[0] != b[0]) return Integer.compare(a[0], b[0]);
            return Integer.compare(a[1], b[1]); // prioritize left
edges over right edges
        });
        // Step 3: Use a max-heap to track heights
        PriorityQueue<Integer> heights = new
PriorityQueue<>(Collections.reverseOrder());
        heights.add(0); // Start with ground level
        int prevMaxHeight = 0;
        // Step 4: Process events
        for (int[] event : events) {
            if (event[1] < 0) { // Left edge
                heights.add(-event[1]);
            } else { // Right edge
                heights.remove(event[1]);
            }
            // Current maximum height
            int currMaxHeight = heights.peek();
            // Step 5: If height changes, record the skyline point
            if (currMaxHeight != prevMaxHeight) {
                result.add(Arrays.asList(event[0], currMaxHeight));
                prevMaxHeight = currMaxHeight;
            }
        }
                return result;
    }
}
```



Accepted 44 / 44 testcases passed

abhigyankit submitted at Feb 18, 2025 18:14

Runtime

243 ms | Beats 11.51%

Analyze Complexity

Memory

51.43 MB | Beats 54.69%

Code | Java

```
class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<List<Integer>> result = new ArrayList<>();
        List<int[]> events = new ArrayList<>();
        for (int[] building : buildings) {
            events.add(new int[] {building[0], -building[2]}); // left edge
            events.add(new int[] {building[1], building[2]});  // right edge
        }
```

```
class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<List<Integer>> result = new ArrayList<>();
        List<int[]> events = new ArrayList<>();
        for (int[] building : buildings) {
            events.add(new int[] {building[0], -building[2]}); // left edge
            events.add(new int[] {building[1], building[2]});  // right edge
        }

        events.sort((a, b) -> {
            if (a[0] != b[0]) return Integer.compare(a[0], b[0]);
            return Integer.compare(a[1], b[1]); // prioritize left edges over right edges
```

Saved

Ln 15, Col 21

Accepted Runtime: 1 ms

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]]
```

Expected