



Assignment - 4

Student Name: Aniket Gupta

UID: 22BCS13824

Branch: BE-CSE

Section/Group: 22BCS_IOT-606/B

Semester: 6th

Date of Submission: 18/03/25

Subject Name: Advanced programming

Subject Code: 22CSP-351

Lab II

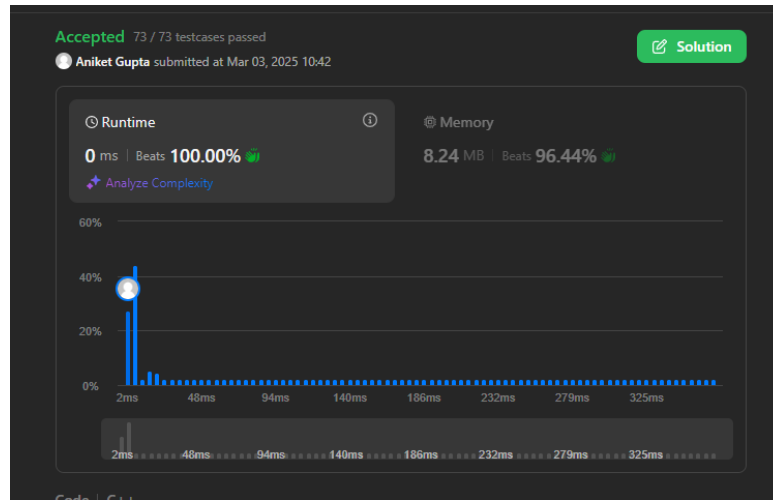
PROBLEM 1:

1. **Aim:** Longest Nice Substring
2. **Objective:** A string *s* is nice if, for every letter of the alphabet that *s* contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

3. **Code:**

```
class Solution {  
    public String longestNiceSubstring(String s) {  
        Set<Character> charSet = new HashSet<>();  
        for (int i = 0; i < s.length(); i++) {  
            charSet.add(s.charAt(i));  
        }  
        for (int i = 0; i < s.length(); i++) {  
            if (charSet.contains(Character.toUpperCase(s.charAt(i))) &&  
                charSet.contains(Character.toLowerCase(s.charAt(i)))) {  
                continue;  
            }  
            String s1 = longestNiceSubstring(s.substring(0, i));  
            String s2 = longestNiceSubstring(s.substring(i+1));  
            return s1.length() >= s2.length() ? s1 : s2;  
        }  
        return s;  
    }  
}
```

4. **Output:**



5. Time complexity: $O(n \log n)$

Space Complexity: $O(n)$

PROBLEM 2:

1. Aim: Maximum Subarray

2. Objective: Given an integer array `nums`, find the subarray find the largest sum and return its sum

3. Code:

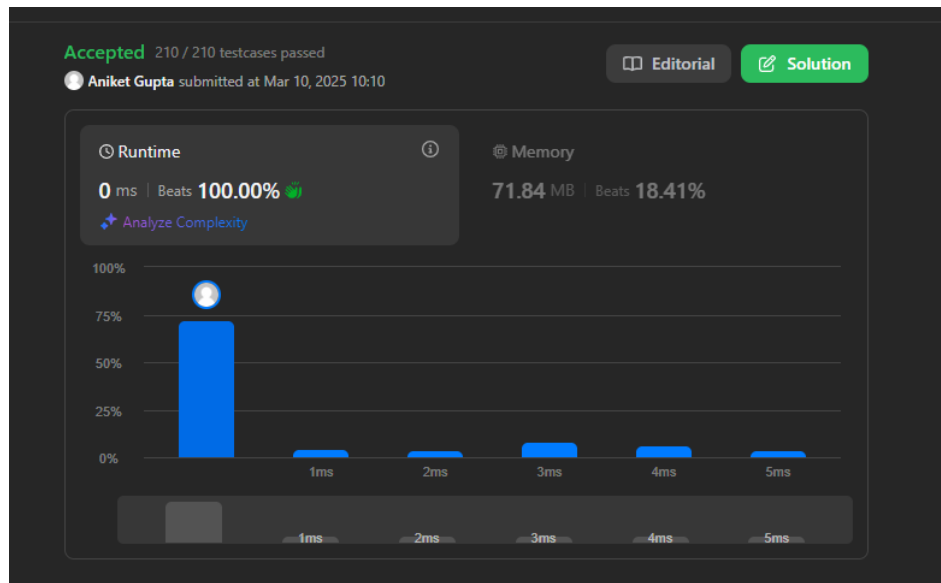
```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxi = Integer.MIN_VALUE;
        int sum = 0;

        for (int num : nums) {
            sum += num;
            maxi = Math.max(maxi, sum);

            if (sum < 0) {
                sum = 0;
            }
        }

        return maxi;
    }
}
```

4. Output:



5. Time Complexity: $O(n)$

Space Complexity: $O(1)$

PROBLEM 3:

- Aim:** Reverse Pairs
- Objective:** Given an integer array `nums`, return the number of reverse pairs in the array.

3. Code:

```
class Solution {
    public int reversePairs(int[] nums) {
        return mergeSortAndCount(nums, 0, nums.length - 1);
    }

    private int mergeSortAndCount(int[] arr, int start, int end) {
        if (start >= end) return 0;

        int mid = (start + end) / 2;
        int count = mergeSortAndCount(arr, start, mid) + mergeSortAndCount(arr, mid + 1, end);

        int j = mid + 1;
        for (int i = start; i <= mid; i++) {
```

```

        while (j <= end && (long) arr[i] > 2L * arr[j]) j++; // Ensure no integer overflow
        count += (j - (mid + 1));
    }

    merge(arr, start, mid, end);
    return count;
}

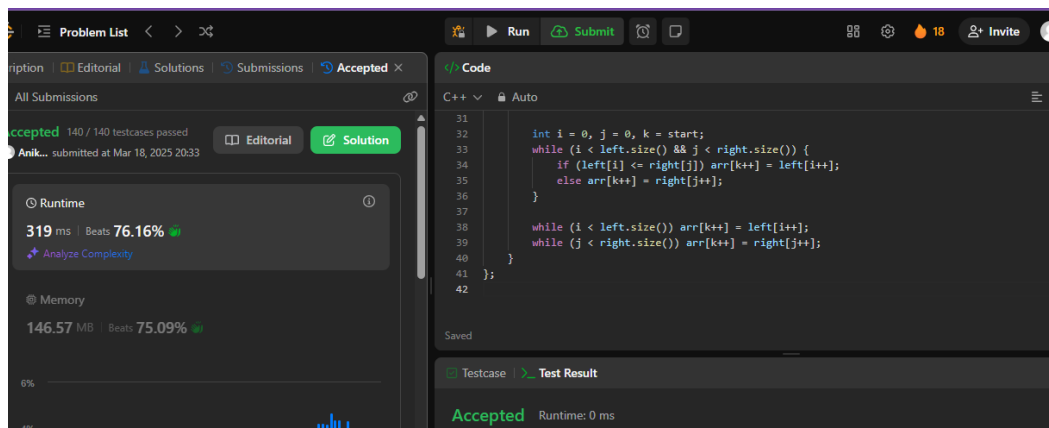
private void merge(int[] arr, int start, int mid, int end) {
    int n1 = mid - start + 1, n2 = end - mid;
    int[] left = new int[n1], right = new int[n2];

    for (int i = 0; i < n1; i++) left[i] = arr[start + i];
    for (int j = 0; j < n2; j++) right[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = start;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) arr[k++] = left[i++];
        else arr[k++] = right[j++];
    }
    while (i < n1) arr[k++] = left[i++];
    while (j < n2) arr[k++] = right[j++];
}
}

```

4. Output:



```

31
32
33
34
35
36
37
38
39
40
41
42

```

Runtime: 319 ms | Beats: 76.16%
Memory: 146.57 MB | Beats: 75.09%

Accepted Runtime: 0 ms

5. Time complexity: $O(N \log N)$ Space complexity: $O(n)$



PROBLEM 4:

1. **Aim:** Longest increasing subsequence II.
2. **Objective:** 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.

3. **Code:**

```
class Solution {

    int N = 100001;
    int[] seg = new int[2*N];

    void update(int pos, int val){
        pos += N;
        seg[pos] = val;

        while (pos > 1) {
            pos >>= 1;
            seg[pos] = Math.max(seg[2*pos], seg[2*pos+1]);
        }
    }

    int query(int lo, int hi){
        lo += N;
        hi += N;
        int res = 0;

        while (lo < hi) {
            if ((lo & 1) == 1) {
                res = Math.max(res, seg[lo++]);
            }
        }
    }
}
```



```
        if ((hi & 1) == 1) {  
            res = Math.max(res, seg[--hi]);  
        }  
        lo >>= 1;  
        hi >>= 1;  
    }  
    return res;  
}
```

```
public int lengthOfLIS(int[] A, int k) {  
    int ans = 0;  
    for (int i = 0; i < A.length; ++i) {  
        int l = Math.max(0, A[i] - k);  
        int r = A[i];  
        int res = query(l, r) + 1;  
        ans = Math.max(res, ans);  
        update(A[i], res);  
    }  
    return ans;  
}
```

4. Output:

The screenshot displays a code editor interface for a C++ solution. The code defines a class `Solution` with a static constant `N = 100001` and a `vector<int> seg`. The `update` method takes `pos` and `val`, updates `seg[pos]` to `val`, and then iterates upwards from `pos` to `N`, updating `seg[pos]` to the maximum of its current value and `seg[2 * pos]`. The `Test Result` section shows the solution is **Accepted** with a runtime of 3 ms. The `Runtime` section shows 51 ms and 90.12% beats, while the `Memory` section shows 120.17 MB and 42.39% beats. A bar chart at the bottom left shows the distribution of runtime and memory usage across test cases.

```
1 class Solution {  
2 private:  
3     static const int N = 100001;  
4     vector<int> seg;  
5  
6     void update(int pos, int val) {  
7         pos += N;  
8         seg[pos] = val;  
9         while (pos > 1) {  
10             pos >>= 1;  
11             seg[pos] = max(seg[2 * pos], seg[2 * pos + 1]);  
12         }  
13     }
```

Accepted 84 / 84 testcases passed
Aniket Gupta submitted at Mar 18, 2025 20:41

Runtime
51 ms | Beats 90.12%
Analyze Complexity

Memory
120.17 MB | Beats 42.39%

Test Result
Accepted Runtime: 3 ms
Case 1 Case 2 Case 3



5. Time Complexity: $O(N \log N)$

Space Complexity: $O(n)$

6. Learning Outcome:

1. Recursion and Divide & Conquer: Used recursive approaches to solve problems like Longest Nice Substring and Reverse Pairs, breaking them into smaller subproblems.
2. Kadane's Algorithm: Implemented an $O(N)$ solution for Maximum Subarray Sum, maintaining a running sum while discarding negative contributions.
3. Merge Sort for Counting Inversions: Used a modified merge sort ($O(N \log N)$) to efficiently count reverse pairs while merging sorted subarrays.
4. Segment Tree for Range Queries: Implemented Segment Tree to efficiently compute Longest Increasing Subsequence with constraints in $O(N \log N)$.
5. Efficient Data Structures: Used HashSet for quick lookups in Longest Nice Substring, improving performance for character presence checks.
6. Handling Edge Cases and Overflow: Prevented integer overflow in Reverse Pairs using `(long) arr[i] > 2L * arr[j]`, ensuring correct calculations.