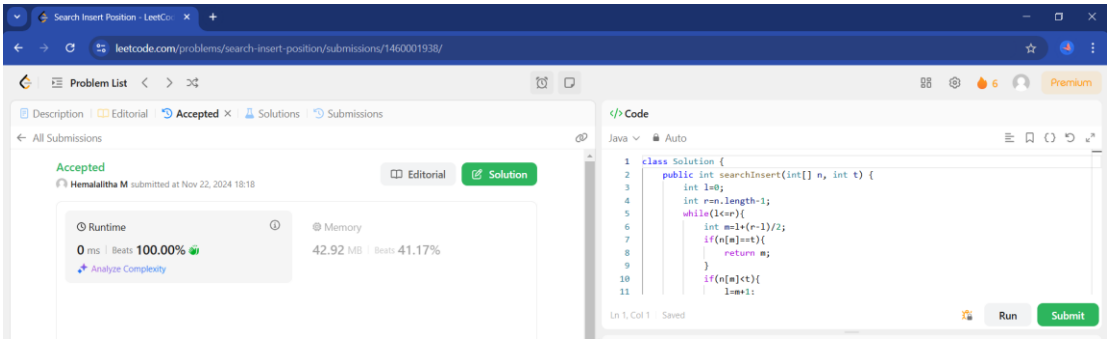
**1. Search insert position:**

**CODE:**

```
class Solution {
    public int searchInsert(int[] n, int t) {
        int l=0;
        int r=n.length-1;
        while(l<=r){
            int m=l+(r-l)/2;
            if(n[m]==t){
                return m;
            }
            if(n[m]<t){
                l=m+1;
            }else{
                r=m-1;
            }
        }
        return l;
    }
}
```

**OUTPUT:**



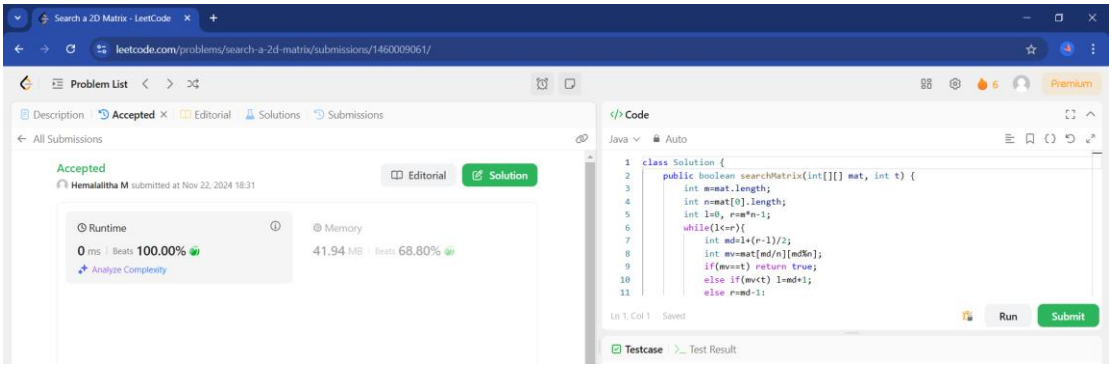

Time complexity: O(log n)

**2. Search a 2D matrix:**

**CODE:**

```
class Solution {
    public boolean searchMatrix(int[][] mat, int t) {
        int m=mat.length;
        int n=mat[0].length;
        int l=0, r=m*n-1;
        while(l<=r){
            int md=l+(r-l)/2;
            int mv=mat[md/n][md%n];
            if(mv==t) return true;
            else if(mv<t) l=md+1;
            else r=md-1;
        }
        return false;
    }
}
```

**OUTPUT:**



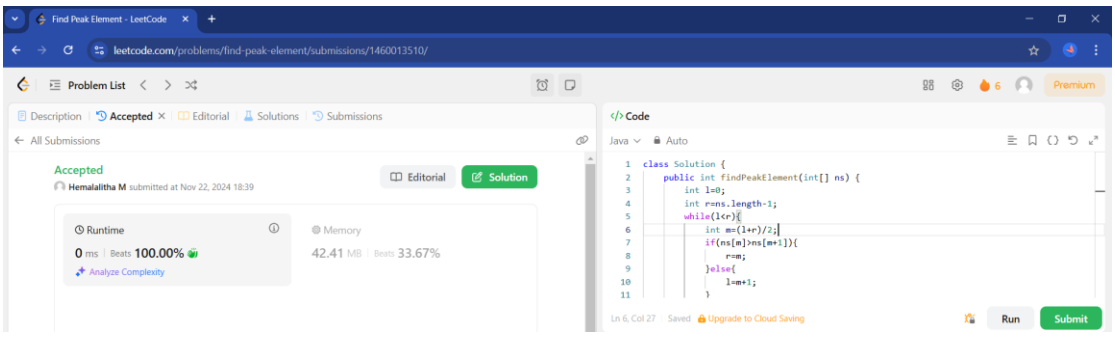

Time complexity: O(log(m*n))

**3. Find peak element:**

**CODE:**

```
class Solution {
    public int findPeakElement(int[] ns) {
        int l=0;
        int r=ns.length-1;
        while(l<r){
            int m=(l+r)/2;
            if(ns[m]>ns[m+1]){
                r=m;
            }else{
                l=m+1;
            }
        }
        return l;
    }
}
```

**OUTPUT:**




Time complexity: O(log n)

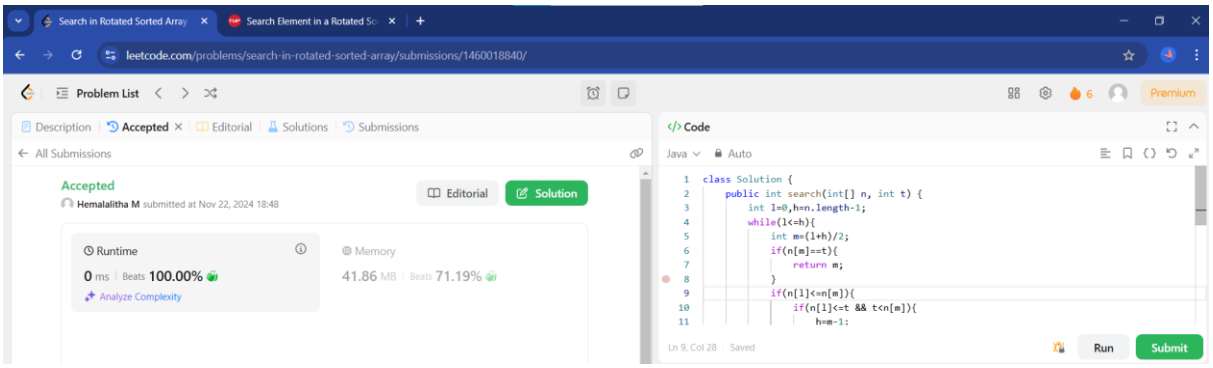
**4. Search in rotated sorted array:**

**CODE:**

```
class Solution {
    public int search(int[] n, int t) {
        int l=0,h=n.length-1;
        while(l<=h){
            int m=(l+h)/2;
            if(n[m]==t){
                return m;
            }
```

```
            if(n[l]<=n[m]){
                if(n[l]<=t && t<n[m]){
                    h=m-1;
                }else{
                    l=m+1;
                }
            }else{
                if(n[m]<t && t<=n[h]){
                    l=m+1;
                }else{
                    h=m-1;
                }
            }
        }
        return -1;
    }
}
```

**OUTPUT:**



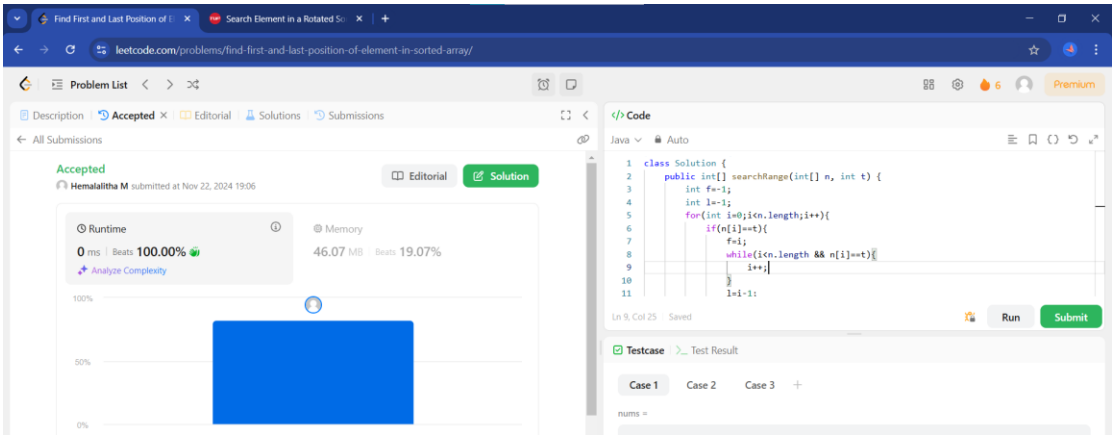

Time complexity: O(log n)

**5. Find first and last position of element in sorted array:**

**CODE:**

```
class Solution {
    public int[] searchRange(int[] n, int t) {
        int f=-1;
        int l=-1;
        for(int i=0;i<n.length;i++){
            if(n[i]==t){
                f=i;
                while(i<n.length && n[i]==t){
                    i++;
                }
                l=i-1;
                return new int[]{f,l};
            }
        }
        return new int[]{f,l};
    }
}
```

**OUTPUT:**



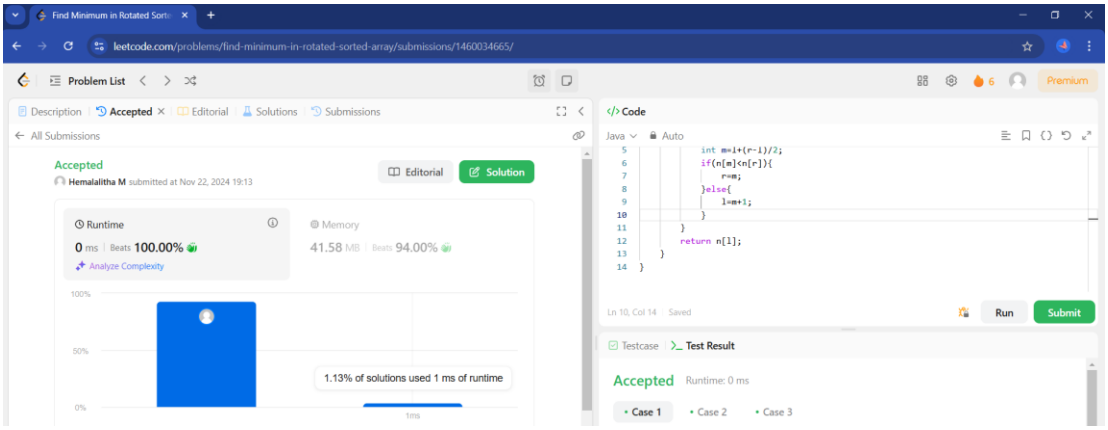

Time complexity: O(n)

**6. Find minimum in rotated sorted array:**

**CODE:**

```
class Solution {
    public int findMin(int[] n) {
        int l=0, r=n.length-1;
        while(l<r){
            int m=l+(r-l)/2;
            if(n[m]<n[r]){
                r=m;
            }else{
                l=m+1;
            }
        }
        return n[l];
    }
}
```

**OUTPUT:**




Time complexity: O(log n)

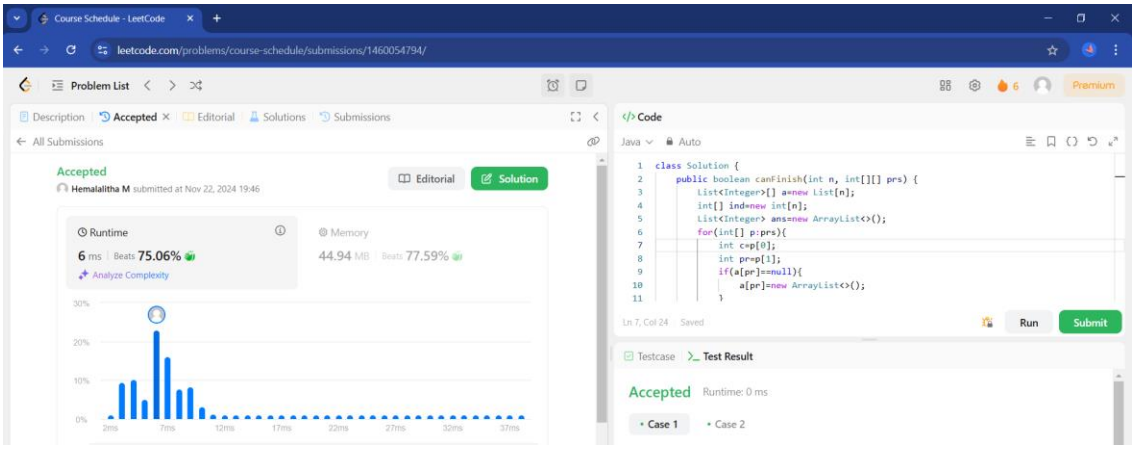
**7. Course schedule:**

**CODE:**

```
class Solution {
    public boolean canFinish(int n, int[][] prs) {
        List<Integer>[] a=new List[n];
        int[] ind=new int[n];
```

```
        List<Integer> ans=new ArrayList<>();
        for(int[] p:prs){
            int c=p[0];
            int pr=p[1];
            if(a[pr]==null){
                a[pr]=new ArrayList<>();
            }
            a[pr].add(c);
            ind[c]++;
        }
        Queue<Integer> q=new LinkedList<>();
        for(int i=0;i<n;i++){
            if(ind[i]==0){
                q.offer(i);
            }
        }
        while(!q.isEmpty()){
            int curr=q.poll();
            ans.add(curr);
            if(a[curr]!=null){
                for(int nx:a[curr]){
                    ind[nx]--;
                    if(ind[nx]==0){
                        q.offer(nx);
                    }
                }
            }
        }
        return ans.size()==n;
    }
}
```

**OUTPUT:**




Time complexity: O(m+n)