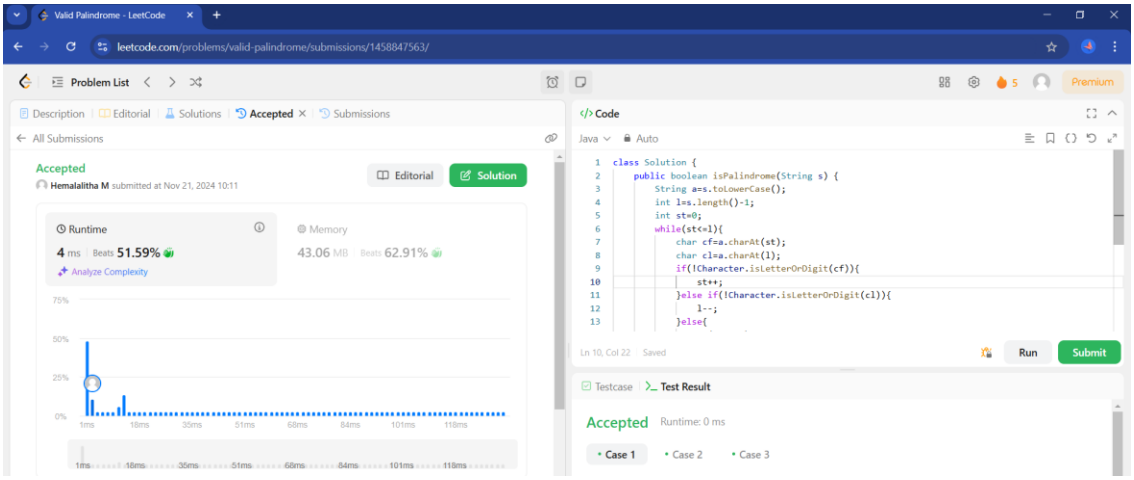
1. Valid palindrome:

CODE:

```
class Solution {
    public boolean isPalindrome(String s) {
        String a=s.toLowerCase();
        int l=s.length()-1;
        int st=0;
        while(st<=l){
            char cf=a.charAt(st);
            char cl=a.charAt(l);
            if(!Character.isLetterOrDigit(cf)){
                st++;
            }else if(!Character.isLetterOrDigit(cl)){
                l--;
            }else{
                if(cf!=cl) return false;
                st++;
                l--;
            }
        }
        return true;
    }
}
```

OUTPUT:



Time complexity: O(n)

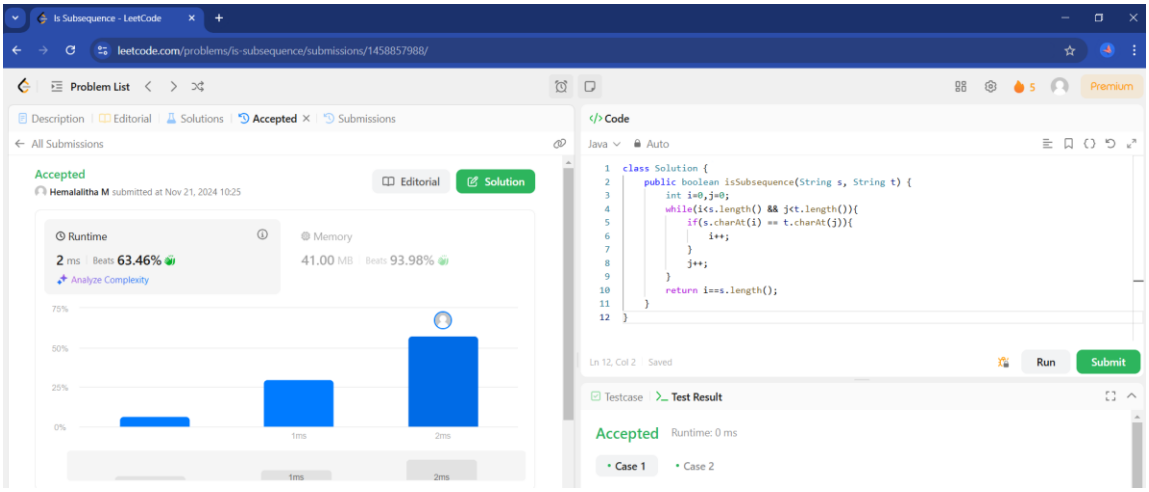
2. Is subsequence:

CODE:

```
class Solution {
    public boolean isSubsequence(String s, String t) {
        int i=0,j=0;
        while(i<s.length() && j<t.length()){
            if(s.charAt(i) == t.charAt(j)){
                i++;
            }
            j++;
        }
        return i==s.length();
    }
```

}

OUTPUT:



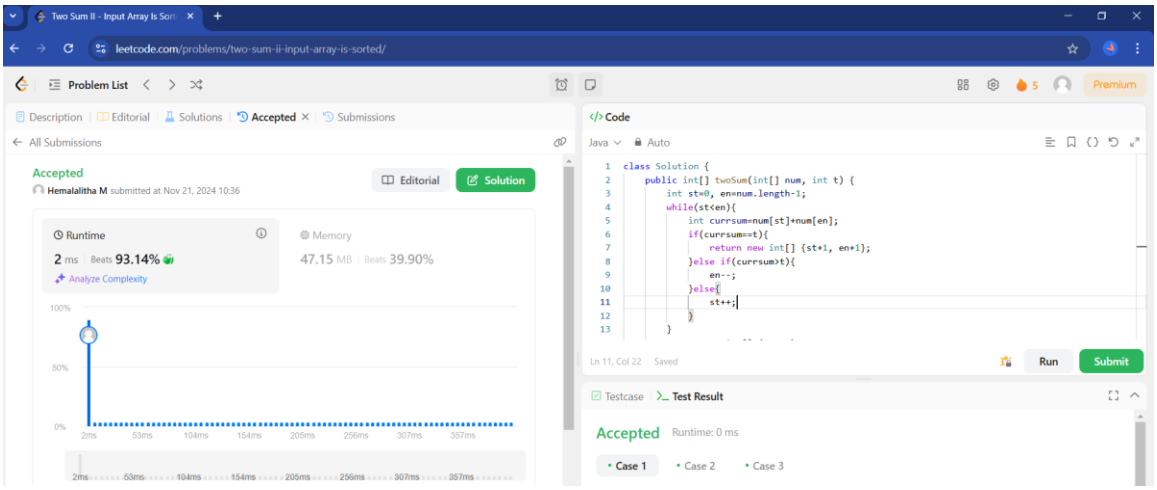
Time complexity: O(n)

3. Two sum 2:

CODE:

```
class Solution {
    public int[] twoSum(int[] num, int t) {
        int st=0, en=num.length-1;
        while(st<en){
            int currsum=num[st]+num[en];
            if(currsum==t){
                return new int[] {st+1, en+1};
            }else if(currsum>t){
                en--;
            }else{
                st++;
            }
        }
        return new int[] {-1,-1};
    }
}
```

OUTPUT:



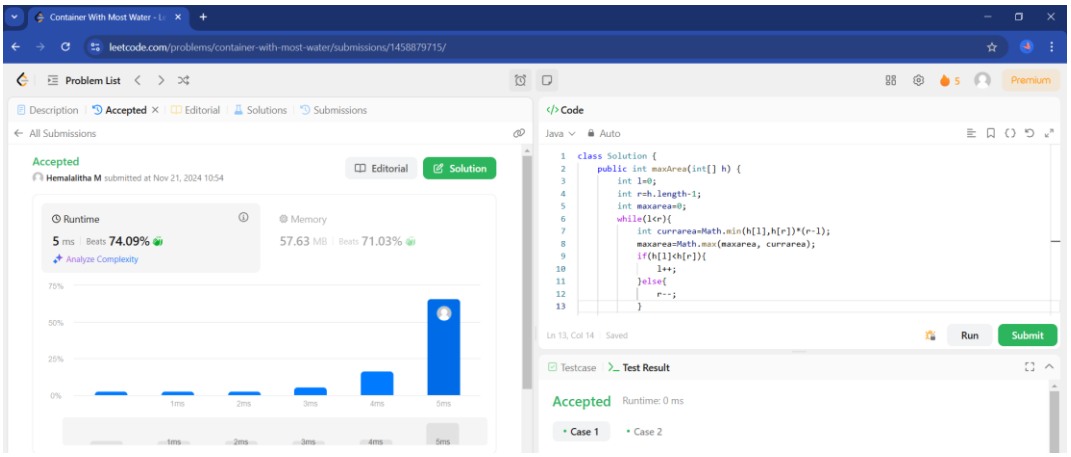
Time complexity: O(n)

4. Container with most water:

CODE:

```
class Solution {
    public int maxArea(int[] h) {
        int l=0;
        int r=h.length-1;
        int maxarea=0;
        while(l<r){
            int currarea=Math.min(h[l],h[r])*(r-l);
            maxarea=Math.max(maxarea, currarea);
            if(h[l]<h[r]){
                l++;
            }else{
                r--;
            }
        }
        return maxarea;
    }
}
```

OUTPUT:



Time complexity: O(n)

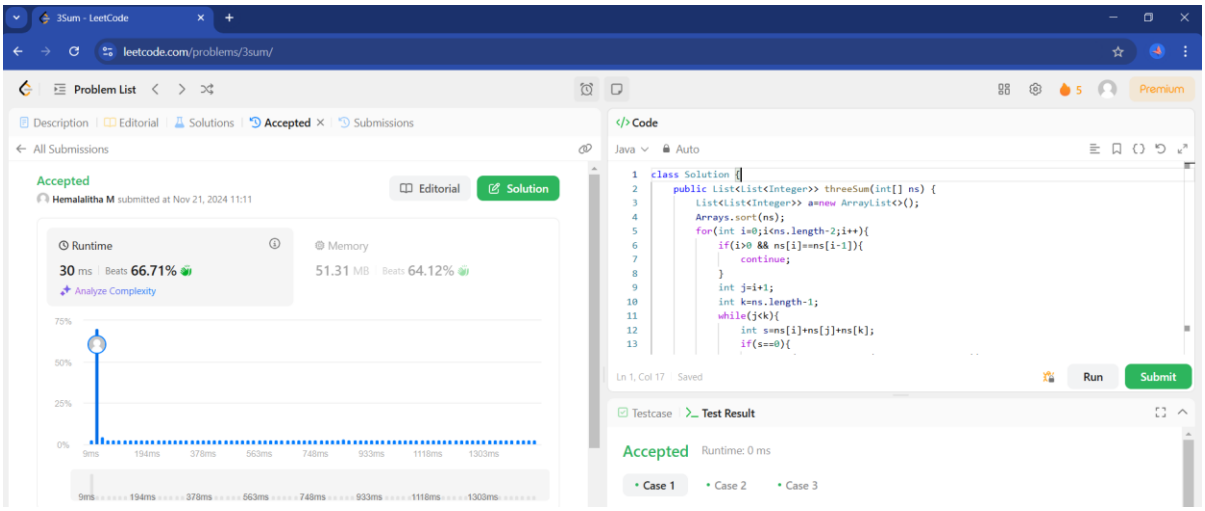
5. 3 sum:

CODE:

```
class Solution {
    public List<List<Integer>> threeSum(int[] ns) {
        List<List<Integer>> a=new ArrayList<>();
        Arrays.sort(ns);
        for(int i=0;i<ns.length-2;i++){
            if(i>0 && ns[i]==ns[i-1]){
                continue;
            }
            int j=i+1;
            int k=ns.length-1;
            while(j<k){
                int s=ns[i]+ns[j]+ns[k];
                if(s==0){
                    a.add(Arrays.asList(ns[i],ns[j],ns[k]));
                    while(j<k && ns[j]==ns[j+1]){
                        j++;
```

```
                    }
                    while(j<k && ns[k]==ns[k-1]){
                        k--;
                    }
                    j++;
                    k--;
                }else if(s<0){
                    j++;
                }else{
                    k--;
                }
            }
        }
        return a;
    }
}
```

OUTPUT:



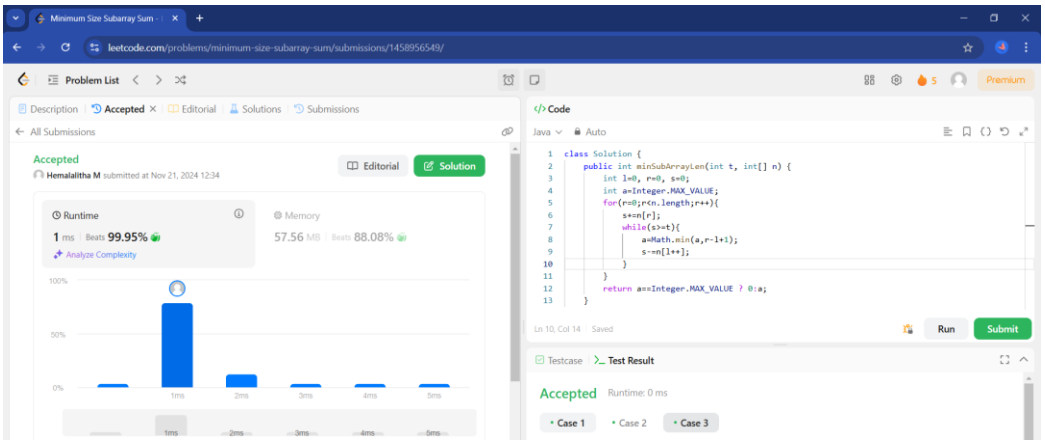
Time complexity: O(n^2)

6. Minimum size subarray sum:

CODE:

```
class Solution {
    public int minSubArrayLen(int t, int[] n) {
        int l=0, r=0, s=0;
        int a=Integer.MAX_VALUE;
        for(r=0;r<n.length;r++){
            s+=n[r];
            while(s>=t){
                a=Math.min(a,r-l+1);
                s-=n[l++];
            }
        }
        return a==Integer.MAX_VALUE ? 0:a;
    }
}
```

OUTPUT:



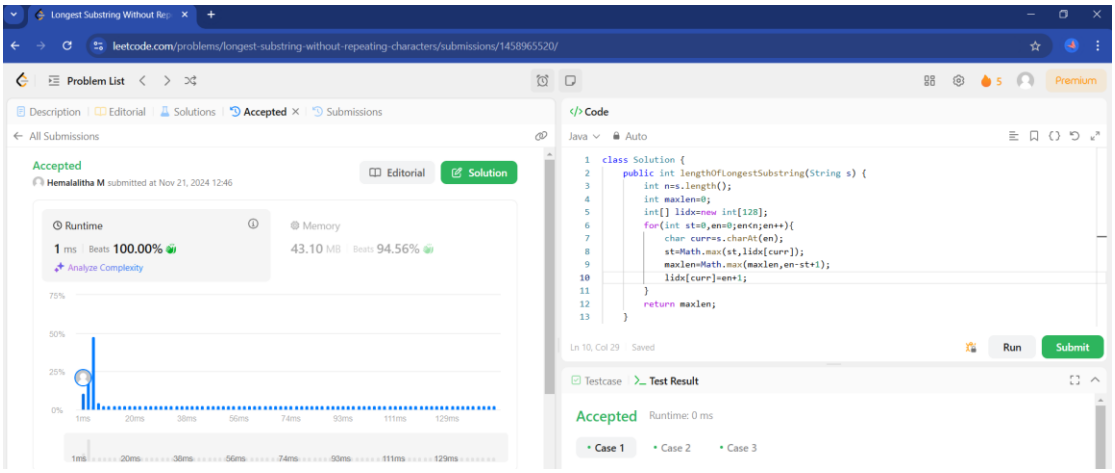
Time complexity: O(n)

7. Longest substring without repeating characters:

CODE:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n=s.length();
        int maxlen=0;
        int[] lidx=new int[128];
        for(int st=0,en=0;en<n;en++){
            char curr=s.charAt(en);
            st=Math.max(st,lidx[curr]);
            maxlen=Math.max(maxlen,en-st+1);
            lidx[curr]=en+1;
        }
        return maxlen;
    }
}
```

OUTPUT:



Time complexity: O(n)

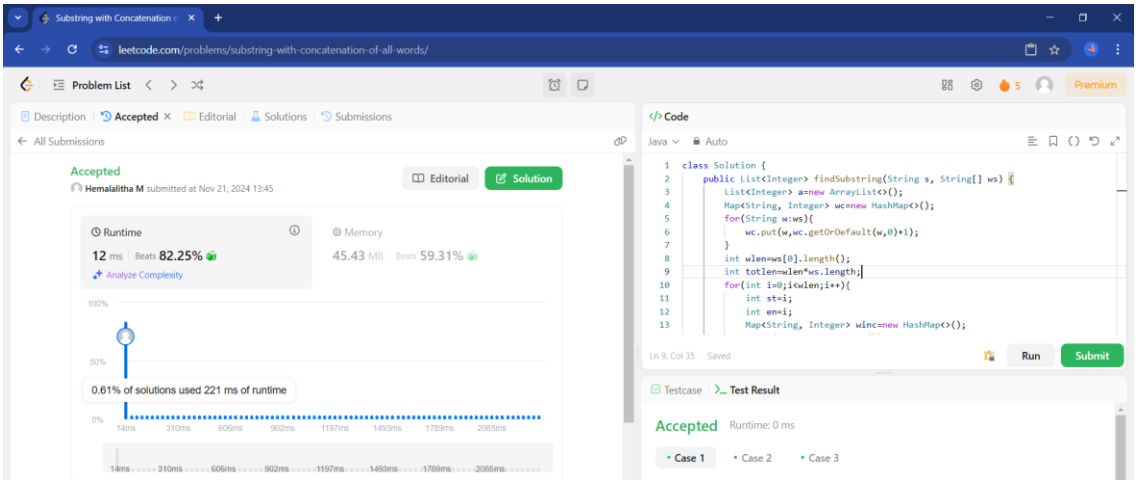
8. Substring with concatenation of all words:

CODE:

```
class Solution {
    public List<Integer> findSubstring(String s, String[] ws) {
        List<Integer> a=new ArrayList<>();
        Map<String, Integer> wc=new HashMap<>();
```

```
        for(String w:ws){
            wc.put(w,wc.getOrDefault(w,0)+1);
        }
        int wlen=ws[0].length();
        int totlen=wlen*ws.length;
        for(int i=0;i<wlen;i++){
            int st=i;
            int en=i;
            Map<String, Integer> winc=new HashMap<>();
            while(en+wlen<=s.length()){
                String w=s.substring(en,en+wlen);
                en+=wlen;
                if(wc.containsKey(w)){
                    winc.put(w,winc.getOrDefault(w,0)+1);
                    while(winc.get(w)>wc.get(w)){
                        String stw=s.substring(st,st+wlen);
                        winc.put(stw,winc.get(stw)-1);
                        st+=wlen;
                    }
                    if(en-st==totlen){
                        a.add(st);
                    }
                }else{
                    winc.clear();
                    st=en;
                }
            }
        }
        return a;
    }
}
```

OUTPUT:



Time complexity: O(n*m)

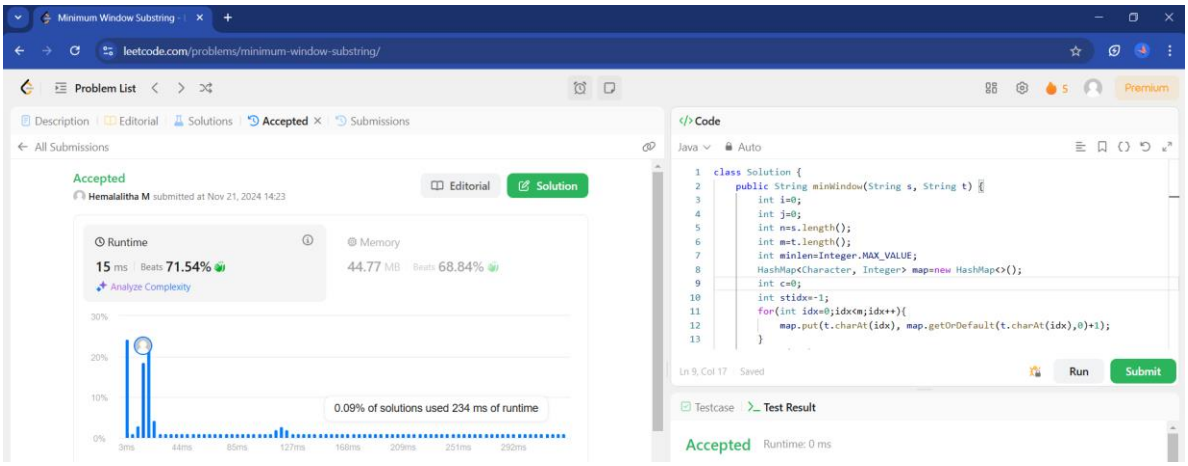
9. Minimum window substring:

CODE:

```
class Solution {
    public String minWindow(String s, String t) {
        int i=0;
```

```
        int j=0;
        int n=s.length();
        int m=t.length();
        int minlen=Integer.MAX_VALUE;
        HashMap<Character, Integer> map=new HashMap<>();
        int c=0;
        int stidx=-1;
        for(int idx=0;idx<m;idx++){
            map.put(t.charAt(idx), map.getOrDefault(t.charAt(idx),0)+1);
        }
        while(j<n){
            if(map.getOrDefault(s.charAt(j),0)>0){
                c++;
            }
            map.put(s.charAt(j),map.getOrDefault(s.charAt(j),0)-1);
            while(c==m){
                if(j-i+1<minlen){
                    minlen=j-i+1;
                    stidx=i;
                }
                map.put(s.charAt(i),map.get(s.charAt(i))+1);
                if(map.get(s.charAt(i))>0){
                    c--;
                }
                i++;
            }
            j++;
        }
        if(stidx==-1){
            return "";
        }
        return s.substring(stidx, stidx+minlen);
    }
}
```

OUTPUT:



Time complexity: O(n)

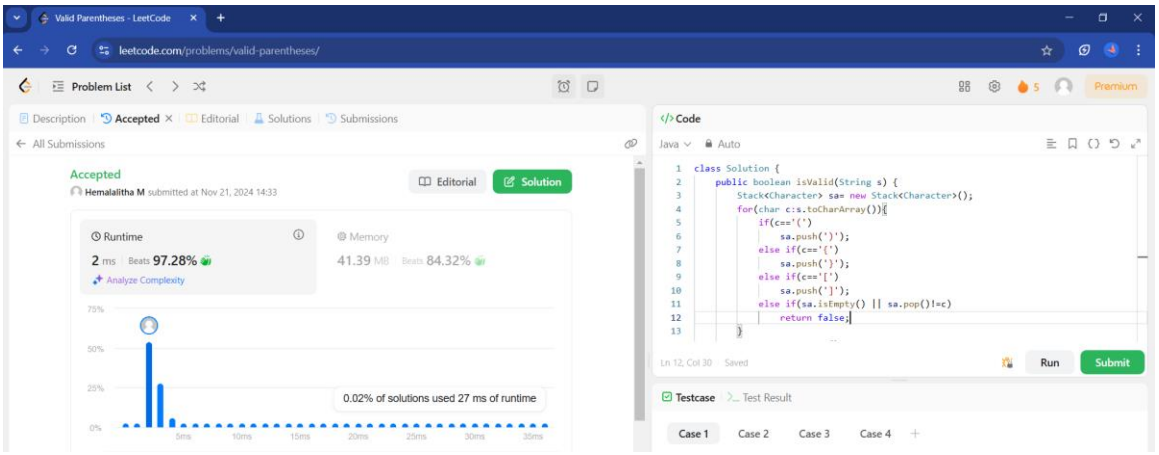
10. Valid parentheses:

CODE:

```
class Solution {
```

```
    public boolean isValid(String s) {
        Stack<Character> sa= new Stack<Character>();
        for(char c:s.toCharArray()){
            if(c=='(')
                sa.push(')');
            else if(c=='{')
                sa.push('}');
            else if(c=='[')
                sa.push(']');
            else if(sa.isEmpty() || sa.pop()!=c)
                return false;
        }
        return sa.isEmpty();
    }
}
```

OUTPUT:



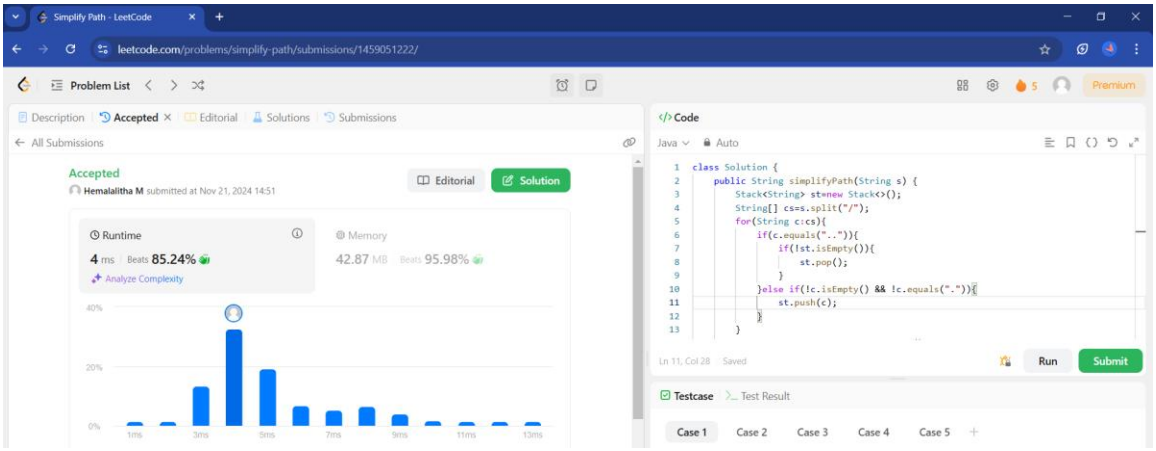
Time complexity: O(n)

11. Simplify path:

CODE:

```
class Solution {
    public String simplifyPath(String s) {
        Stack<String> st=new Stack<>();
        String[] cs=s.split("/");
        for(String c:cs){
            if(c.equals("..")){
                if(!st.isEmpty()){
                    st.pop();
                }
            }else if(!c.isEmpty() && !c.equals(".")){
                st.push(c);
            }
        }
        StringBuilder r=new StringBuilder();
        for(String d:st){
            r.append("/").append(d);
        }
        return r.length()>0?r.toString():"/";
    }
}
```

OUTPUT:



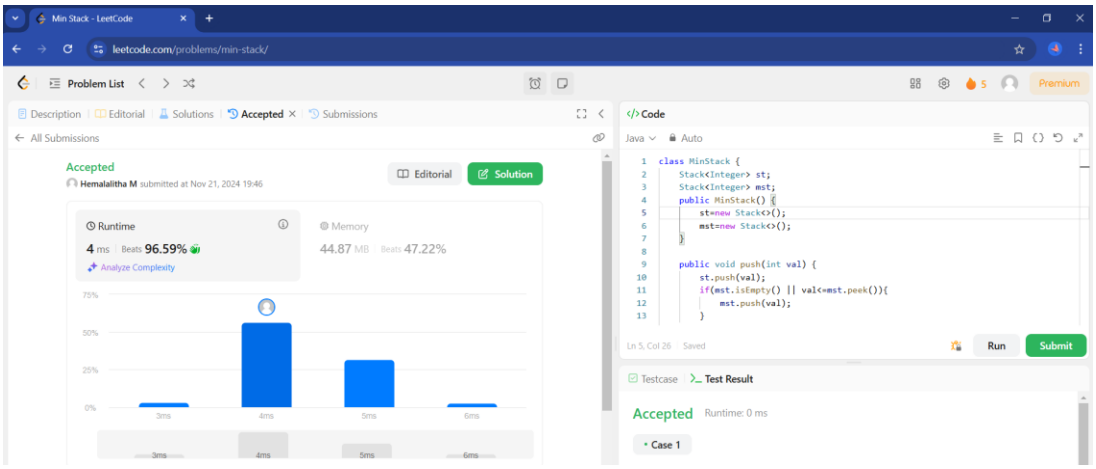
Time complexity: O(n)

12. Min stack:

CODE:

```
class MinStack {
    Stack<Integer> st;
    Stack<Integer> mst;
    public MinStack() {
        st=new Stack<>();
        mst=new Stack<>();
    }
    public void push(int val) {
        st.push(val);
        if(mst.isEmpty() || val<=mst.peek()){
            mst.push(val);
        }
    }
    public void pop() {
        if(st.pop().equals(mst.peek())){
            mst.pop();
        }
    }
    public int top() {
        return st.peek();
    }
    public int getMin() {
        return mst.peek();
    }
}
```

OUTPUT:



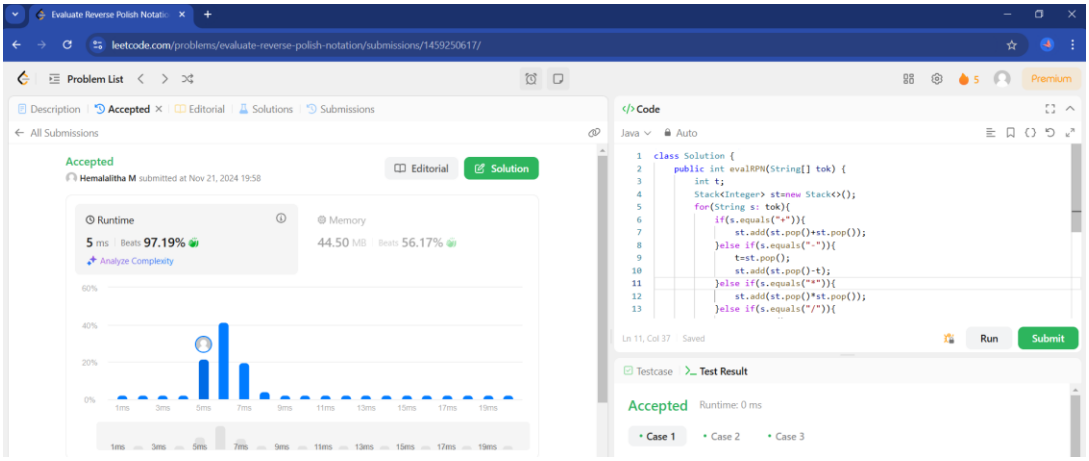
Time complexity: O(1)

13. Evaluate reverse polish notation:

CODE:

```
class Solution {
    public int evalRPN(String[] tok) {
        int t;
        Stack<Integer> st=new Stack<>();
        for(String s: tok){
            if(s.equals("+")){
                st.add(st.pop()+st.pop());
            }else if(s.equals("-")){
                t=st.pop();
                st.add(st.pop()-t);
            }else if(s.equals("*")){
                st.add(st.pop()*st.pop());
            }else if(s.equals("/")){
                t=st.pop();
                st.add(st.pop()/t);
            }else{
                st.add(Integer.parseInt(s));
            }
        }
        return st.pop();
    }
}
```

OUTPUT:



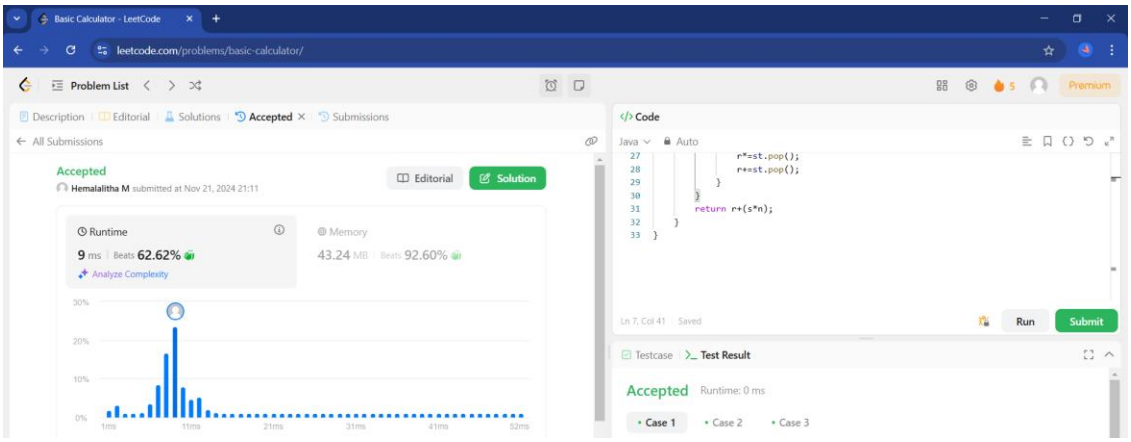
Time complexity: O(n)

14. Basic calculator:

CODE:

```
class Solution {
    public int calculate(String str) {
        int r=0;
        int s=1;
        int n=0;
        Stack<Integer> st=new Stack<>();
        for(int i=0;i<str.length();i++){
            char ch=str.charAt(i);
            if(Character.isDigit(ch)){
                n=n*10+(ch-'0');
            }else if(ch=='+'){
                r+=s*n;
                n=0;
                s=1;
            }else if(ch=='-'){
                r+=s*n;
                n=0;
                s=-1;
            }else if(ch=='('){
                st.push(r);
                st.push(s);
                r=0;
                s=1;
            }else if(ch==')'){
                r+=s*n;
                n=0;
                r*=st.pop();
                r+=st.pop();
            }
        }
        return r+(s*n);
    }
}
```

OUTPUT:



Time complexity: O(n)