T.BASHEERA

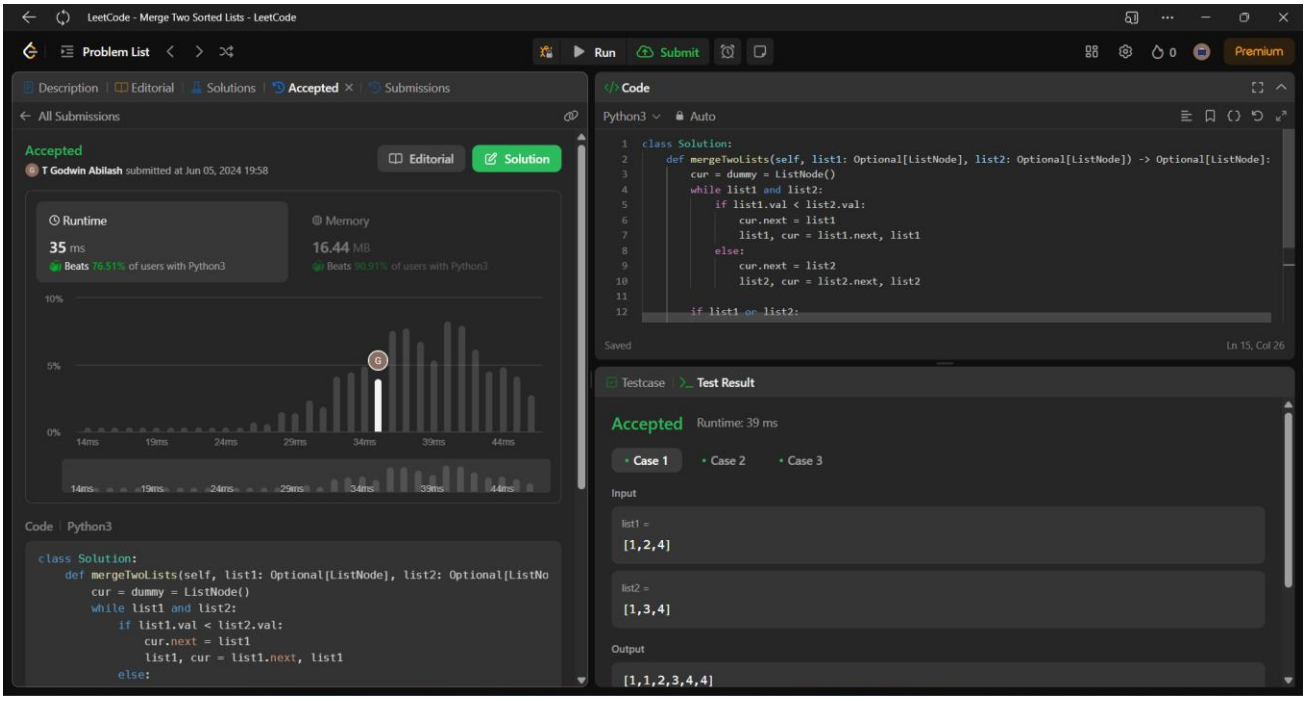
192311074

ASSIGNMENT - 3

21. MERGE TWO SORTED LIST

```
CLASS SOLUTION:
    DEF MERGETWOLISTS(SELF, LIST1: OPTIONAL[LISTNODE], LIST2:
OPTIONAL[LISTNODE]) -> OPTIONAL[LISTNODE]:
        CUR = DUMMY = LISTNODE()
        WHILE LIST1 AND LIST2:
            IF LIST1.VAL < LIST2.VAL:
                CUR.NEXT = LIST1
                LIST1, CUR = LIST1.NEXT, LIST1
            ELSE:
                CUR.NEXT = LIST2
                LIST2, CUR = LIST2.NEXT, LIST2
        IF LIST1 OR LIST2:
            CUR.NEXT = LIST1 IF LIST1 ELSE LIST2
        RETURN DUMMY.NEXT
```

Screenshot:



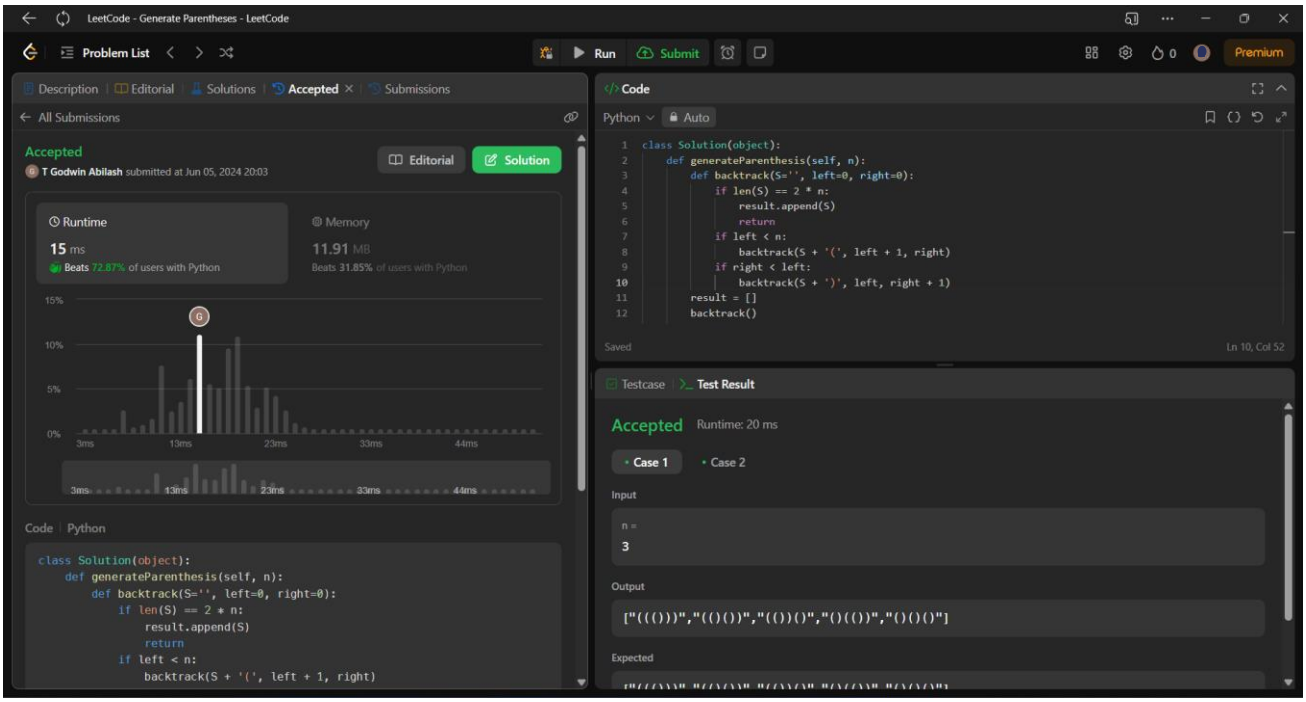
Time Complexity: O(n)

22. GENERATE PARENTHESIS

Code:

```
CLASS SOLUTION(OBJECT):
    DEF GENERATEPARENTHESIS(SELF, N):
        DEF BACKTRACK(S='', LEFT=0, RIGHT=0):
            IF LEN(S) == 2 * N:
                RESULT.APPEND(S)
                RETURN
            IF LEFT < N:
                BACKTRACK(S + '(', LEFT + 1, RIGHT)
            IF RIGHT < LEFT:
                BACKTRACK(S + ')', LEFT, RIGHT + 1)
        RESULT = []
        BACKTRACK()
        RETURN RESULT
```

Screenshot for I/O:



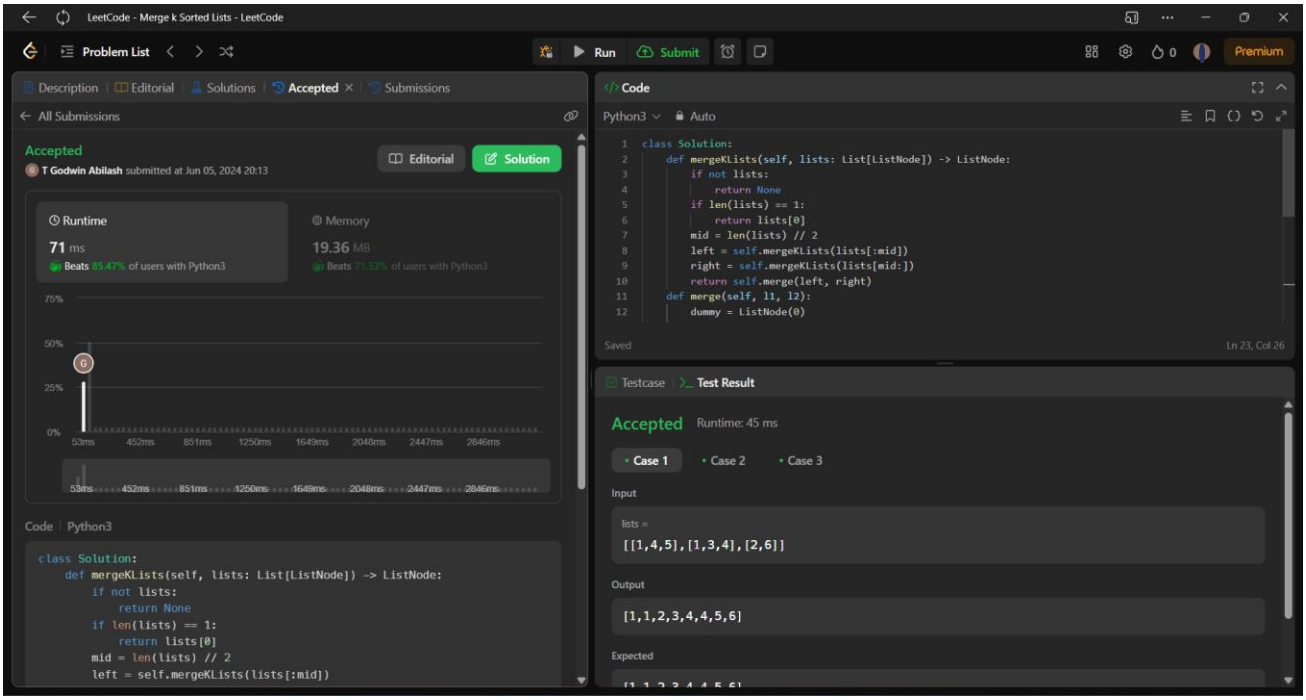
Time Complexity: O(n)

23. MERGE K SORTED LISTS

Code:

```
CLASS SOLUTION:
    DEF MERGEKLISTS(SELF, LISTS: LIST[LISTNODE]) -> LISTNODE:
        IF NOT LISTS:
            RETURN NONE
        IF LEN(LISTS) == 1:
            RETURN LISTS[0]
        MID = LEN(LISTS) // 2
        LEFT = SELF.MERGEKLISTS(LISTS[:MID])
        RIGHT = SELF.MERGEKLISTS(LISTS[MID:])
        RETURN SELF.MERGE(LEFT, RIGHT)
    DEF MERGE(SELF, L1, L2):
        DUMMY = LISTNODE(0)
        CURR = DUMMY
        WHILE L1 AND L2:
            IF L1.VAL < L2.VAL:
                CURR.NEXT = L1
                L1 = L1.NEXT
            ELSE:
                CURR.NEXT = L2
                L2 = L2.NEXT
            CURR = CURR.NEXT
        CURR.NEXT = L1 OR L2
        RETURN DUMMY.NEXT
```

Screenshot:



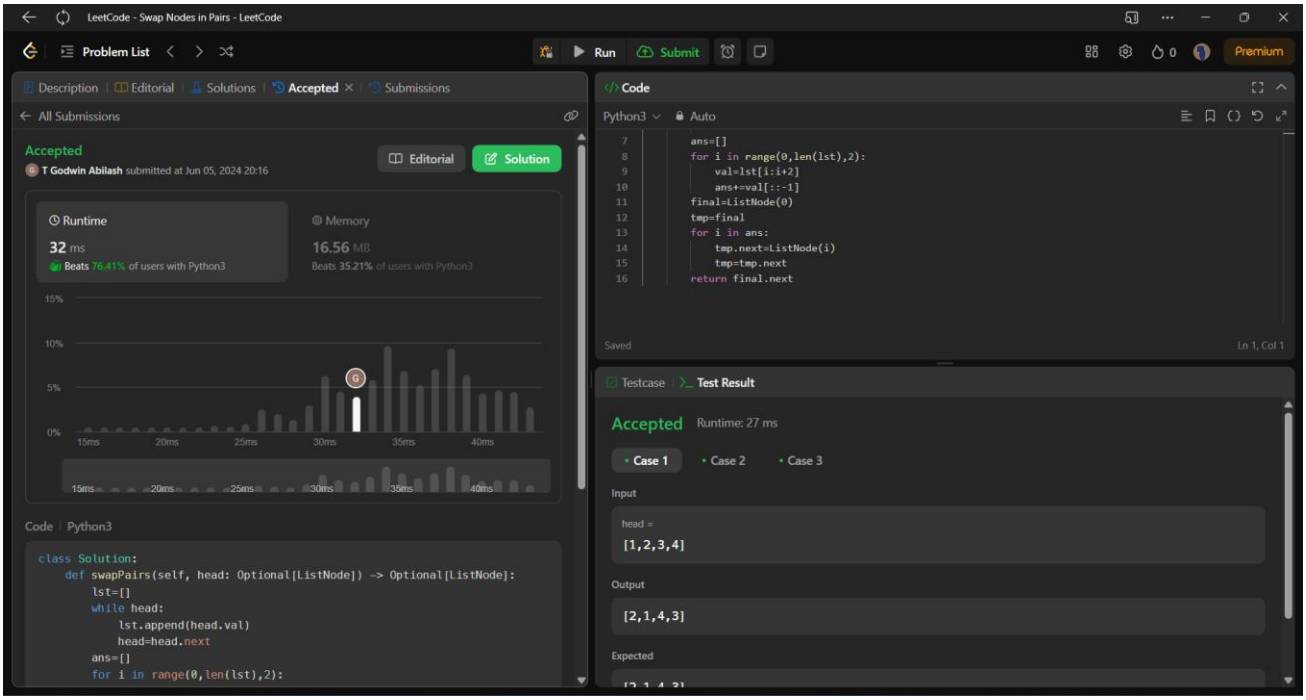
Time Complexity: O(n)

24. SWAP NODES IN PAIRS

Code:

```
CLASS SOLUTION:
    DEF SWAPPAIRS(SELF, HEAD: OPTIONAL[LISTNODE]) -> OPTIONAL[LISTNODE]:
        LST=[]
        WHILE HEAD:
            LST.APPEND(HEAD.VAL)
            HEAD=HEAD.NEXT
        ANS=[]
        FOR I IN RANGE(0,LEN(LST),2):
            VAL=LST[I:I+2]
            ANS+=VAL[::-1]
        FINAL=LISTNODE(0)
        TMP=FINAL
        FOR I IN ANS:
            TMP.NEXT=LISTNODE(I)
            TMP=TMP.NEXT
        RETURN FINAL.NEXT
```

Screenshot:



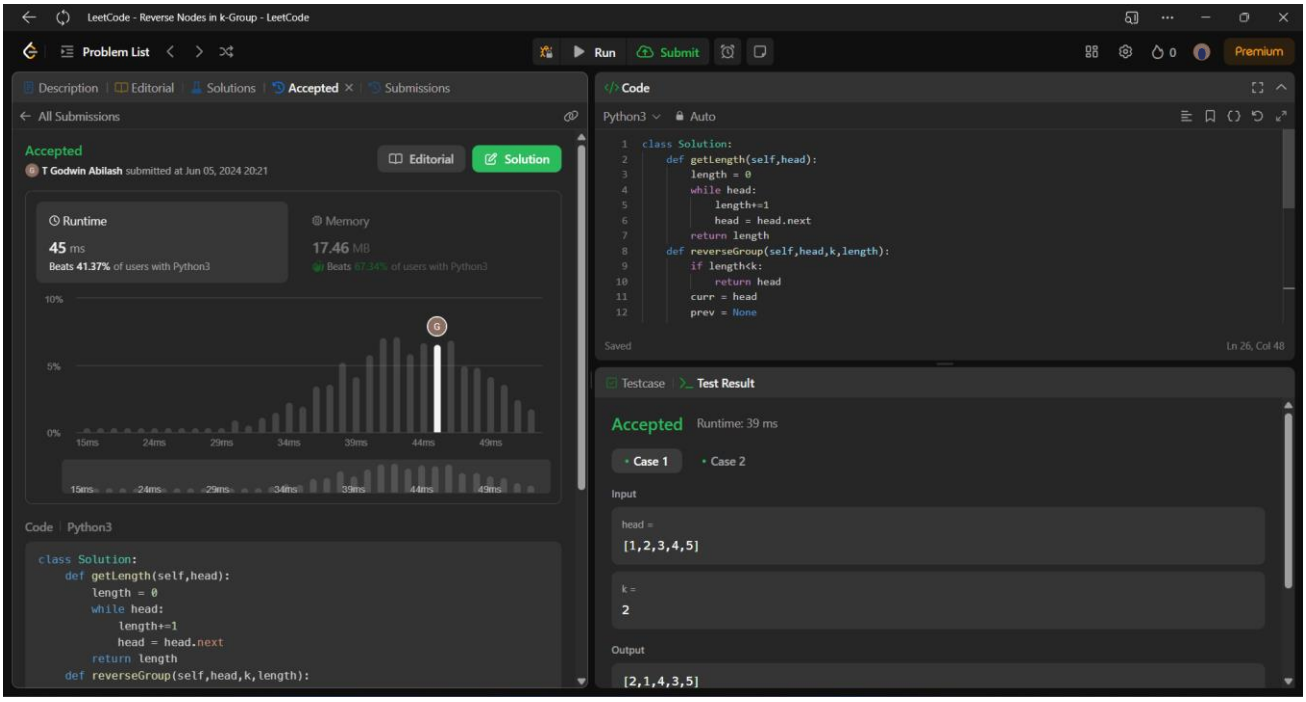
Time Complexity: O(n)

25. REVERSE NODES IN K-GROUP

Code:

```
CLASS SOLUTION:
    DEF GETLENGTH(SELF,HEAD):
        LENGTH = 0
        WHILE HEAD:
            LENGTH+=1
            HEAD = HEAD.NEXT
        RETURN LENGTH
    DEF REVERSEGROUP(SELF,HEAD,K,LENGTH):
        IF LENGTH<K:
            RETURN HEAD
        CURR = HEAD
        PREV = NONE
        NEXT = NONE
        COUNT = 0
        WHILE CURR AND COUNT<K:
            NEXT = CURR.NEXT
            CURR.NEXT = PREV
            PREV = CURR
            CURR = NEXT
            COUNT+=1
        IF NEXT:
            HEAD.NEXT = SELF.REVERSEGROUP(NEXT,K,LENGTH-K)
        RETURN PREV
    DEF REVERSEKGROUP(SELF, HEAD: OPTIONAL[LISTNODE], K: INT) -> OPTIONAL[LISTNODE]:
        LENGTH = SELF.GETLENGTH(HEAD)
        RETURN SELF.REVERSEGROUP(HEAD,K,LENGTH)
```

Screenshot:



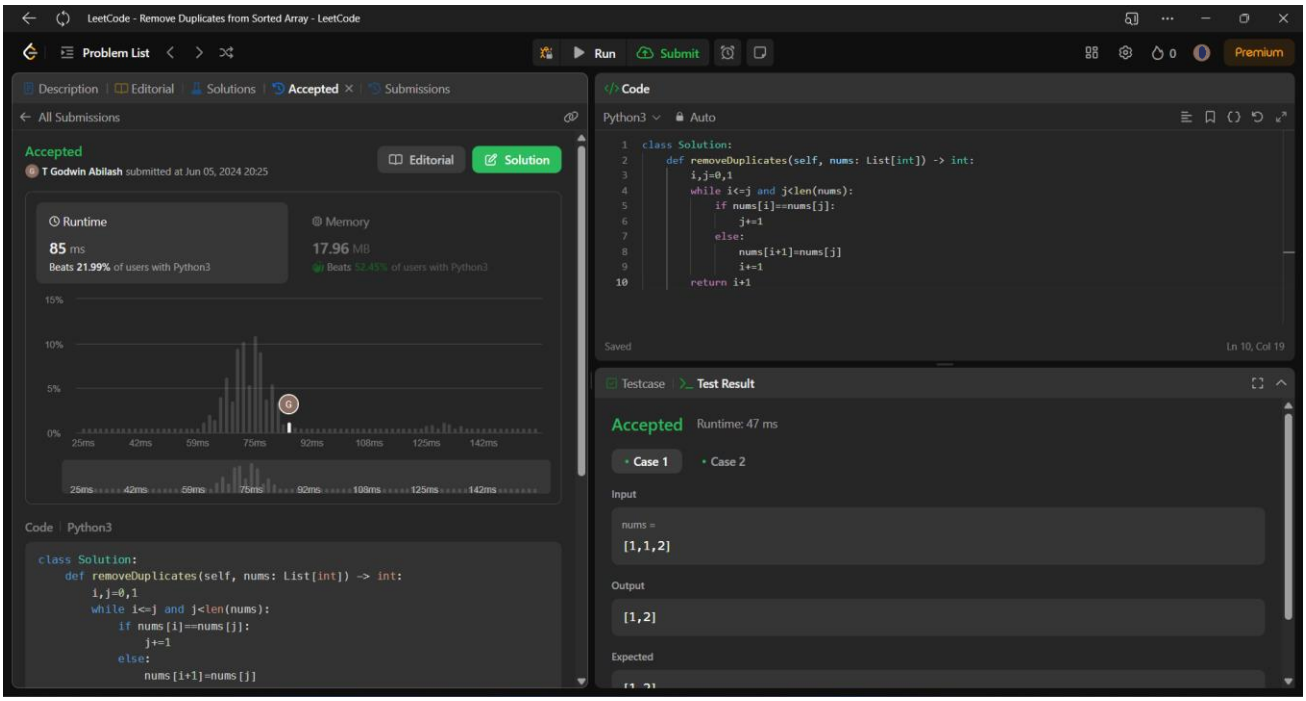
Time Complexity: O(n)

26. Remove Duplicate from Sorted Array

Code:

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        i,j=0,1
        while i<=j and j<len(nums):
            if nums[i]==nums[j]:
                j+=1
            else:
                nums[i+1]=nums[j]
                i+=1
        return i+1
```

Screenshot:



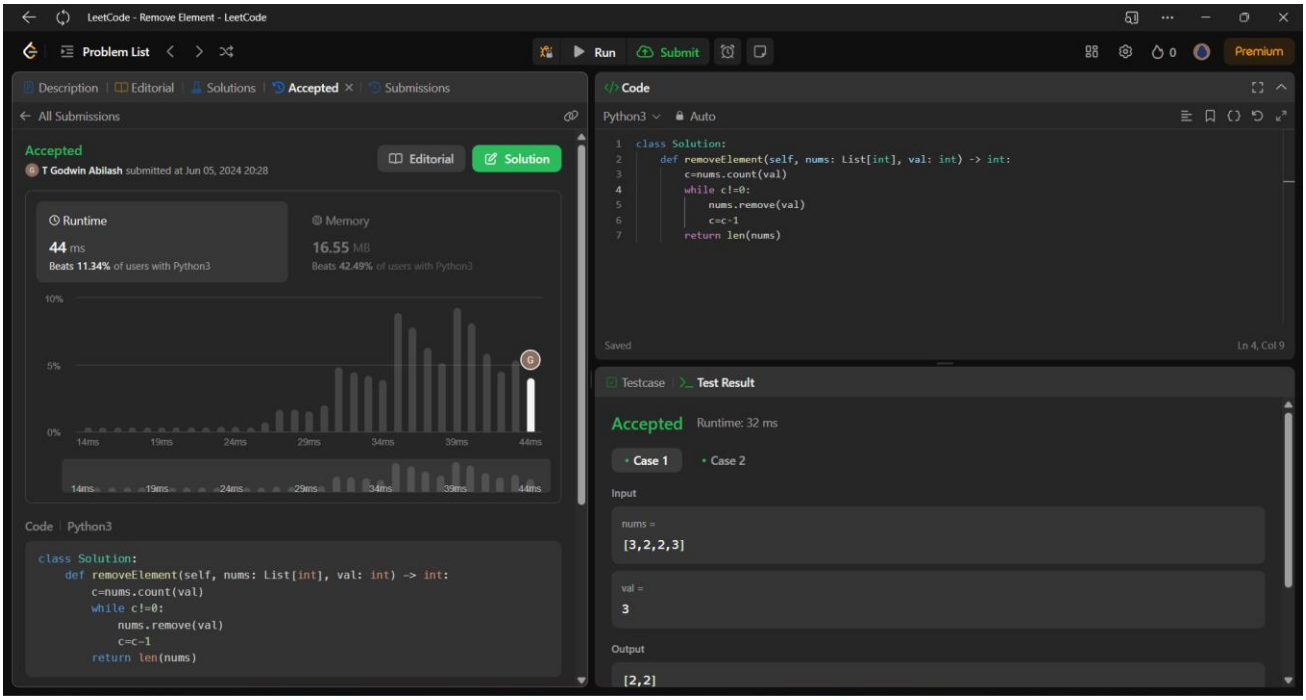
Time Complexity: O(n)

27. Remove Element

Code:

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        c=nums.count(val)
        while c!=0:
            nums.remove(val)
            c=c-1
        return len(nums)
```

Screenshot:



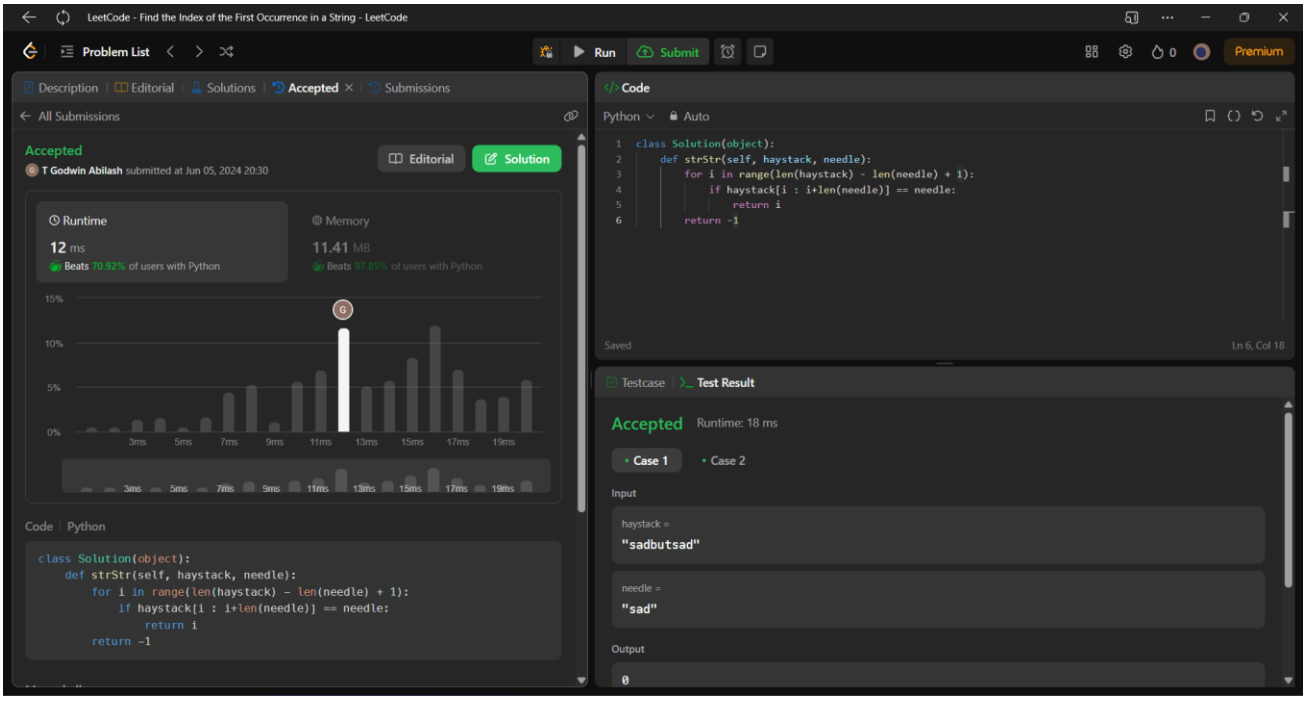
Time Complexity: O(n)

28. Find the Index of the First Occurrence in a String

Code:

```
class Solution(object):
    def strStr(self, haystack, needle):
        for i in range(len(haystack) - len(needle) + 1):
            if haystack[i : i+len(needle)] == needle:
                return i
        return -1
```

Screenshot:



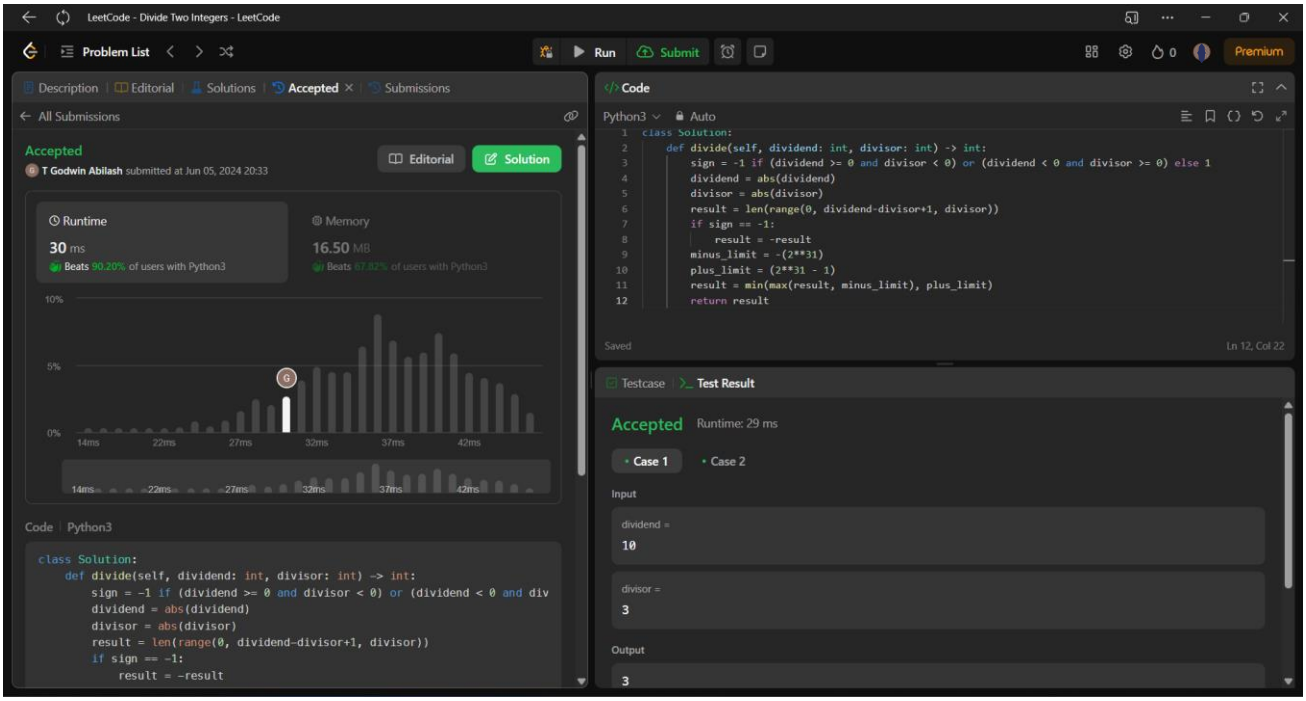
Time Complexity: O(n)

29. Divide Two Integers

Code:

```
class Solution:
    def divide(self, dividend: int, divisor: int) -> int:
        sign = -1 if (dividend >= 0 and divisor < 0) or (dividend < 0 and
divisor >= 0) else 1
        dividend = abs(dividend)
        divisor = abs(divisor)
        result = len(range(0, dividend-divisor+1, divisor))
        if sign == -1:
            result = -result
        minus_limit = -(2**31)
        plus_limit = (2**31 - 1)
        result = min(max(result, minus_limit), plus_limit)
        return result
```

Screenshot:



Time Complexity: O(n)

30. Substring with Concatenation of All Words

Code:

```
CLASS SOLUTION:
    DEF CALC(SELF ,I):
        CNT = 0
        IND = I
        WHILE (IND < I+SELF.PL):
            NEWS = SELF.S[IND : IND + SELF.N]
            IF NEWS IN SELF.DIC :
                SELF.DIC[NEWS] -= 1
                IF SELF.DIC[NEWS] == 0 :
                    CNT += 1
                IND += SELF.N
            ELSE :
                RETURN FALSE
        IF CNT == LEN(SELF.DIC) :
            RETURN TRUE
        ELSE :
            RETURN FALSE
    DEF FINDSUBSTRING(SELF, S: STR, WORDS: LIST[STR]) -> LIST[INT]:
         SELF.S = S
        SELF.N = LEN(WORDS[0])
        D = {}
        FOR X IN WORDS :
            IF X IN D :
                D[X] += 1
            ELSE :
                D[X] = 1
        SELF.PL = LEN(WORDS)*LEN(WORDS[0])
        ANS = []
        I = 0
        WHILE(I< LEN(S) - SELF.PL + 1):
            SELF.DIC = {X : D[X] FOR X IN D}
            IF SELF.CALC(I) :
                ANS += [I]
                I += 1
            ELSE :
                I += 1
        RETURN ANS
```

Screenshot: