



Full Name: Deepthi M

Email: deepthiim95@gmail.com

Test Name: Mock Test

Taken On: 14 Aug 2025 12:47:05 IST

Time Taken: 6 min 51 sec/ 90 min

Invited by: Ankush

Invited on: 14 Aug 2025 12:46:38 IST

Skills Score:

Tags Score:

- Algorithms280/280
- Core CS280/280
- Data Structures105/105
- Easy280/280
- LCM105/105
- Least Common Multiple105/105
- Math105/105
- Problem Solving105/105
- Strings175/175
- gcd105/105
- greatest common divisor105/105
- problem-solving280/280
- sets105/105

100%

280/280

scored in **Mock Test** in 6 min 51 sec on 14 Aug 2025 12:47:05 IST

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review it in detail here -

	Question Description	Time Taken	Score	Status
Q1	Palindrome Index > Coding	2 min 16 sec	105/ 105	⚠
Q2	Between Two Sets > Coding	2 min 7 sec	105/ 105	✅
Q3	Anagram > Coding	2 min 16 sec	70/ 70	✅

## QUESTION 1



Needs Review

Score 105

## Palindrome Index &gt; Coding

Strings

Algorithms

Easy

problem-solving

Core CS

Problem Solving

## QUESTION DESCRIPTION

Given a string of lowercase letters in the range `ascii[a-z]`, determine the index of a character that can be removed to make the string a **palindrome**. There may be more than one solution, but any will do. If the word is already a palindrome or there is no solution, return `-1`. Otherwise, return the index of a character to remove.

## Example

 **$s = \text{"bcbc"}$** 

Either remove 'b' at index **0** or 'c' at index **3**.

## Function Description

Complete the `palindromeIndex` function in the editor below.

`palindromeIndex` has the following parameter(s):

- string s*: a string to analyze

## Returns

- int*: the index of the character to remove or **-1**

## Input Format

The first line contains an integer  **$q$** , the number of queries.

Each of the next  **$q$**  lines contains a query string  **$s$** .

## Constraints

- $1 \leq q \leq 20$
- $1 \leq \text{length of } s \leq 10^5 + 5$
- All characters are in the range `ascii[a-z]`.

## Sample Input

STDIN	Function
-----	-----
3	$q = 3$
aaab	$s = \text{'aaab'}$ (first query)
baa	$s = \text{'baa'}$ (second query)
aaa	$s = \text{'aaa'}$ (third query)

## Sample Output

```
3
0
-1
```

## Explanation

*Query 1: "aaab"*

Removing 'b' at index **3** results in a palindrome, so return **3**.

*Query 2: "baa"*

Removing 'b' at index **0** results in a palindrome, so return **0**.

*Query 3: "aaa"*

This string is already a palindrome, so return **-1**. Removing any one of the characters would result in a palindrome, but this test comes first.

**Note:** The custom checker logic for this challenge is available [here](#).

## CANDIDATE ANSWER

Language used: C

```
1  #include <assert.h>
2  #include <ctype.h>
3  #include <limits.h>
4  #include <math.h>
5  #include <stdbool.h>
6  #include <stddef.h>
7  #include <stdint.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 char* readline();
13 char* ltrim(char*);
14 char* rtrim(char*);
15
16 int parse_int(char*);
17
18 // Helper function to check if a substring is a palindrome
19 bool isPalindrome(char* str, int start, int end) {
20     while (start < end) {
21         if (str[start] != str[end]) {
22             return false;
23         }
24         start++;
25         end--;
26     }
27     return true;
28 }
29
30 /*
31  * Complete the 'palindromeIndex' function below.
32  *
33  * The function is expected to return an INTEGER.
34  * The function accepts STRING s as parameter.
35  */
36
37 int palindromeIndex(char* s) {
38     int len = strlen(s);
39     int start = 0;
40     int end = len - 1;
41
42     while (start < end) {
43         if (s[start] != s[end]) {
44             // Found a mismatch. Check if removing start or end character
45             makes it a palindrome.
46             // Check if removing the character at 'start' makes the rest of
47             the string a palindrome.
48             if (isPalindrome(s, start + 1, end)) {
49                 return start;
50             }
51             // Check if removing the character at 'end' makes the rest of the
52             string a palindrome.
53             if (isPalindrome(s, start, end - 1)) {
54                 return end;
55             }
56             // If neither works, it's not possible to form a palindrome by
57             removing one character.
58 }
```

```

59         // The problem statement implies one solution will exist if it's
60         not already a palindrome,
61         // but this is a failsafe.
62         return -1;
63     }
64     start++;
65     end--;
66 }
67
68 // The string is already a palindrome.
69 return -1;
70 }
71
72 int main()
73 {
74     FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
75
76     int q = parse_int(ltrim(rtrim(readline())));
77
78     for (int q_itr = 0; q_itr < q; q_itr++) {
79         char* s = readline();
80
81         int result = palindromeIndex(s);
82
83         fprintf(fptr, "%d\n", result);
84     }
85
86     fclose(fptr);
87
88     return 0;
89 }
90
91 char* readline() {
92     size_t alloc_length = 1024;
93     size_t data_length = 0;
94
95     char* data = malloc(alloc_length);
96
97     while (true) {
98         char* cursor = data + data_length;
99         char* line = fgets(cursor, alloc_length - data_length, stdin);
100
101         if (!line) {
102             break;
103         }
104
105         data_length += strlen(cursor);
106
107         if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
108         {
109             break;
110         }
111
112         alloc_length <= 1;
113
114         data = realloc(data, alloc_length);
115
116         if (!data) {
117             data = '\0';
118
119             break;
120         }
121     }
122 }

```

```

12     if (data[data_length - 1] == '\n') {
12         data[data_length - 1] = '\0';
12
12         data = realloc(data, data_length);
12
12         if (!data) {
12             data = '\0';
12         }
12     } else {
12         data = realloc(data, data_length + 1);
12
12         if (!data) {
12             data = '\0';
12         } else {
12             data[data_length] = '\0';
12         }
12     }
12
12     return data;
12 }
12
12 char* ltrim(char* str) {
12     if (!str) {
12         return '\0';
12     }
12
12     if (!*str) {
12         return str;
12     }
12
12     while (*str != '\0' && isspace(*str)) {
12         str++;
12     }
12
12     return str;
12 }
12
12 char* rtrim(char* str) {
12     if (!str) {
12         return '\0';
12     }
12
12     if (!*str) {
12         return str;
12     }
12
12     char* end = str + strlen(str) - 1;
12
12     while (end >= str && isspace(*end)) {
12         end--;
12     }
12
12     *(end + 1) = '\0';
12
12     return str;
12 }
12
12 int parse_int(char* str) {
12     char* endptr;
12     int value = strtol(str, &endptr, 10);
12
12     if (endptr == str || *endptr != '\0') {
12         exit(EXIT_FAILURE);
2

```

```

    }

    return value;
}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✔ Success	0	0.009 sec	7.13 KB
Testcase 2	Medium	Hidden case	✔ Success	5	0.0077 sec	7.38 KB
Testcase 3	Medium	Hidden case	✔ Success	5	0.0077 sec	7.38 KB
Testcase 4	Medium	Hidden case	✔ Success	5	0.0092 sec	7.38 KB
Testcase 5	Medium	Hidden case	✔ Success	5	0.0086 sec	7.13 KB
Testcase 6	Medium	Hidden case	✔ Success	5	0.0077 sec	7.5 KB
Testcase 7	Medium	Hidden case	✔ Success	5	0.0076 sec	7.5 KB
Testcase 8	Medium	Hidden case	✔ Success	5	0.0082 sec	7.75 KB
Testcase 9	Hard	Hidden case	✔ Success	10	0.0081 sec	7 KB
Testcase 10	Hard	Hidden case	✔ Success	10	0.01 sec	7.38 KB
Testcase 11	Hard	Hidden case	✔ Success	10	0.008 sec	7.38 KB
Testcase 12	Hard	Hidden case	✔ Success	10	0.0087 sec	7.25 KB
Testcase 13	Hard	Hidden case	✔ Success	10	0.007 sec	7.25 KB
Testcase 14	Hard	Hidden case	✔ Success	10	0.0072 sec	7.25 KB
Testcase 15	Hard	Hidden case	✔ Success	10	0.0092 sec	7.38 KB

No Comments

## QUESTION 2



Correct Answer

Score 105

## Between Two Sets > Coding

Math

Algorithms

Easy

gcd

Data Structures

LCM

sets

problem-solving

Core CS

greatest common divisor

Least Common Multiple

### QUESTION DESCRIPTION

There will be two arrays of integers. Determine all integers that satisfy the following two conditions:

1. The elements of the first array are all factors of the integer being considered
2. The integer being considered is a factor of all elements of the second array

These numbers are referred to as being *between* the two arrays. Determine how many such numbers exist.

#### Example

$a = [2, 6]$

$b = [24, 36]$

There are two numbers between the arrays: **6** and **12**.

$6\%2 = 0$ ,  $6\%6 = 0$ ,  $24\%6 = 0$  and  $36\%6 = 0$  for the first value.

$12\%2 = 0$ ,  $12\%6 = 0$  and  $24\%12 = 0$ ,  $36\%12 = 0$  for the second value. Return **2**.

#### Function Description

Complete the `getTotalX` function in the editor below. It should return the number of integers that are between the sets.

`getTotalX` has the following parameter(s):

- `int a[n]`: an array of integers
- `int b[m]`: an array of integers

#### Returns

- *int*: the number of integers that are between the sets

### Input Format

The first line contains two space-separated integers,  $n$  and  $m$ , the number of elements in arrays  $a$  and  $b$ .

The second line contains  $n$  distinct space-separated integers  $a[i]$  where  $0 \leq i < n$ .

The third line contains  $m$  distinct space-separated integers  $b[j]$  where  $0 \leq j < m$ .

### Constraints

- $1 \leq n, m \leq 10$
- $1 \leq a[i] \leq 100$
- $1 \leq b[j] \leq 100$

### Sample Input

```
2 3
2 4
16 32 96
```

### Sample Output

```
3
```

### Explanation

2 and 4 divide evenly into 4, 8, 12 and 16.

4, 8 and 16 divide evenly into 16, 32, 96.

4, 8 and 16 are the only three numbers for which each element of  $a$  is a factor and each is a factor of all elements of  $b$ .

## CANDIDATE ANSWER

Language used: C

```
1 #include <assert.h>
2 #include <ctype.h>
3 #include <limits.h>
4 #include <math.h>
5 #include <stdbool.h>
6 #include <stddef.h>
7 #include <stdint.h>
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 char* readline();
13 char* ltrim(char*);
14 char* rtrim(char*);
15 char** split_string(char*);
16
17 int parse_int(char*);
18
19 /*
20  * Complete the 'getTotalX' function below.
21  *
22  * The function is expected to return an INTEGER.
23  * The function accepts following parameters:
24  * 1. INTEGER_ARRAY a
25  * 2. INTEGER_ARRAY b
26  */
27
```

```

28 int getTotalX(int a_count, int* a, int b_count, int* b) {
29     int count = 0;
30
31     // Constraints state values are between 1 and 100.
32     // The range of numbers to check is from the largest element of 'a' to
33 the smallest element of 'b'.
34     int max_a = 0;
35     for (int i = 0; i < a_count; i++) {
36         if (a[i] > max_a) {
37             max_a = a[i];
38         }
39     }
40
41     int min_b = 101; // Since values are <= 100
42     for (int i = 0; i < b_count; i++) {
43         if (b[i] < min_b) {
44             min_b = b[i];
45         }
46     }
47
48     // Iterate through all numbers in the possible range.
49     for (int i = max_a; i <= min_b; i++) {
50         bool condition1 = true;
51         bool condition2 = true;
52
53         // Condition 1: All elements of 'a' are factors of 'i'.
54         for (int j = 0; j < a_count; j++) {
55             if (i % a[j] != 0) {
56                 condition1 = false;
57                 break;
58             }
59         }
60
61         // If condition 1 is met, check condition 2.
62         if (condition1) {
63             // Condition 2: 'i' is a factor of all elements of 'b'.
64             for (int k = 0; k < b_count; k++) {
65                 if (b[k] % i != 0) {
66                     condition2 = false;
67                     break;
68                 }
69             }
70         }
71
72         if (condition1 && condition2) {
73             count++;
74         }
75     }
76
77     return count;
78 }
79
80 int main()
81 {
82     FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
83
84     char** first_multiple_input = split_string(rtrim(readline()));
85
86     int n = parse_int(*(first_multiple_input + 0));
87
88     int m = parse_int(*(first_multiple_input + 1));
89
90     char** arr_temp = split_string(rtrim(readline()));

```



```

91 int* arr = malloc(n * sizeof(int));
92
93 for (int i = 0; i < n; i++) {
94     int arr_item = parse_int(*(arr_temp + i));
95
96     *(arr + i) = arr_item;
97 }
98
99 char** brr_temp = split_string(rtrim(readline()));
100
101 int* brr = malloc(m * sizeof(int));
102
103 for (int i = 0; i < m; i++) {
104     int brr_item = parse_int(*(brr_temp + i));
105
106     *(brr + i) = brr_item;
107 }
108
109 int total = getTotalX(n, arr, m, brr);
110
111 fprintf(fp, "%d\n", total);
112
113 fclose(fp);
114
115 return 0;
116 }
117
118 char* readline() {
119     size_t alloc_length = 1024;
120     size_t data_length = 0;
121
122     char* data = malloc(alloc_length);
123
124     while (true) {
125         char* cursor = data + data_length;
126         char* line = fgets(cursor, alloc_length - data_length, stdin);
127
128         if (!line) {
129             break;
130         }
131
132         data_length += strlen(cursor);
133
134         if (data_length < alloc_length - 1 || data[data_length - 1] != '\n')
135         {
136             break;
137         }
138
139         alloc_length <= 1;
140
141         data = realloc(data, alloc_length);
142
143         if (!data) {
144             data = '\0';
145
146             break;
147         }
148     }
149
150     if (data[data_length - 1] == '\n') {
151         data[data_length - 1] = '\0';
152
153         data = realloc(data, data_length);

```

```

15         if (!data) {
16             data = '\0';
17         }
18     } else {
19         data = realloc(data, data_length + 1);
20
21         if (!data) {
22             data = '\0';
23         } else {
24             data[data_length] = '\0';
25         }
26     }
27 }
28
29 return data;
30 }
31
32 char* ltrim(char* str) {
33     if (!str) {
34         return '\0';
35     }
36
37     if (!*str) {
38         return str;
39     }
40
41     while (*str != '\0' && isspace(*str)) {
42         str++;
43     }
44
45     return str;
46 }
47
48 char* rtrim(char* str) {
49     if (!str) {
50         return '\0';
51     }
52
53     if (!*str) {
54         return str;
55     }
56
57     char* end = str + strlen(str) - 1;
58
59     while (end >= str && isspace(*end)) {
60         end--;
61     }
62
63     *(end + 1) = '\0';
64
65     return str;
66 }
67
68 char** split_string(char* str) {
69     char** splits = NULL;
70     char* token = strtok(str, " ");
71
72     int spaces = 0;
73
74     while (token) {
75         splits = realloc(splits, sizeof(char*) * ++spaces);
76
77         if (!splits) {
78             return splits;

```

```

21     }
22
23     splits[spaces - 1] = token;
24
25     token = strtok(NULL, " ");
26 }
27
28 return splits;
29 }
30
31 int parse_int(char* str) {
32     char* endptr;
33     int value = strtol(str, &endptr, 10);
34
35     if (endptr == str || *endptr != '\0') {
36         exit(EXIT_FAILURE);
37     }
38
39     return value;
40 }
41
42 }
43
44
45
46

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	✓ Success	0	0.0139 sec	7.38 KB
Testcase 2	Easy	Hidden case	✓ Success	15	0.0069 sec	7.13 KB
Testcase 3	Easy	Hidden case	✓ Success	15	0.0092 sec	6.88 KB
Testcase 4	Easy	Hidden case	✓ Success	15	0.007 sec	7.25 KB
Testcase 5	Easy	Hidden case	✓ Success	15	0.0086 sec	7 KB
Testcase 6	Easy	Hidden case	✓ Success	15	0.009 sec	7.25 KB
Testcase 7	Easy	Hidden case	✓ Success	15	0.0089 sec	7.25 KB
Testcase 8	Easy	Hidden case	✓ Success	15	0.0079 sec	7.13 KB
Testcase 9	Easy	Sample case	✓ Success	0	0.0073 sec	7.25 KB

No Comments

### QUESTION 3



Correct Answer

Score 70

## Anagram > Coding

Strings

Algorithms

Easy

problem-solving

Core CS

### QUESTION DESCRIPTION

Two words are *anagrams* of one another if their letters can be rearranged to form the other word.

Given a string, split it into two contiguous substrings of equal length. Determine the minimum number of characters to change to make the two substrings into anagrams of one another.

#### Example

$s = \text{abccde}$

Break  $s$  into two parts: 'abc' and 'cde'. Note that all letters have been used, the substrings are contiguous and their lengths are equal. Now you can change 'a' and 'b' in the first substring to 'd' and 'e' to have 'dec' and 'cde' which are anagrams. Two changes were necessary.

#### Function Description

Complete the *anagram* function in the editor below.

anagram has the following parameter(s):

- *string s*: a string

### Returns

- *int*: the minimum number of characters to change or -1.

### Input Format

The first line will contain an integer, *q*, the number of test cases.

Each test case will contain a string *s*.

### Constraints

- $1 \leq q \leq 100$
- $1 \leq |s| \leq 10^4$
- *s* consists only of characters in the range `ascii[a-z]`.

### Sample Input

```
6
aaabbb
ab
abc
mnop
xyyx
xaxbbbx
```

### Sample Output

```
3
1
-1
2
0
1
```

### Explanation

*Test Case #01:* We split *s* into two strings *S1*='aaa' and *S2*='bbb'. We have to replace all three characters from the first string with 'b' to make the strings anagrams.

*Test Case #02:* You have to replace 'a' with 'b', which will generate "bb".

*Test Case #03:* It is not possible for two strings of unequal length to be anagrams of one another.

*Test Case #04:* We have to replace both the characters of first string ("mn") to make it an anagram of the other one.

*Test Case #05:* *S1* and *S2* are already anagrams of one another.

*Test Case #06:* Here *S1* = "xaxb" and *S2* = "bbxx". You must replace 'a' from *S1* with 'b' so that *S1* = "xbxb".

### CANDIDATE ANSWER

Language used: **C**

```
1 #include <assert.h>
2 #include <ctype.h>
3 #include <limits.h>
4 #include <math.h>
5 #include <stdbool.h>
6 #include <stddef.h>
7 #include <stdint.h>
8 #include <stdio.h>
```

```

9 #include <stdlib.h>
10 #include <string.h>
11
12 char* readline();
13 char* ltrim(char*);
14 char* rtrim(char*);
15
16 int parse_int(char*);
17
18
19
20 /*
21  * Complete the 'anagram' function below.
22  *
23  * The function is expected to return an INTEGER.
24  * The function accepts STRING s as parameter.
25  */
26
27 int anagram(char* s) {
28     int len = strlen(s);
29
30     // If the string length is odd, it's impossible to split into two equal-
31     length substrings.
32     if (len % 2 != 0) {
33         return -1;
34     }
35
36     int half_len = len / 2;
37     char* s1 = (char*)malloc((half_len + 1) * sizeof(char));
38     char* s2 = (char*)malloc((half_len + 1) * sizeof(char));
39
40     // Split the string into two halves.
41     strncpy(s1, s, half_len);
42     s1[half_len] = '\0';
43     strncpy(s2, s + half_len, half_len);
44     s2[half_len] = '\0';
45
46     int freq1[26] = {0};
47     int freq2[26] = {0};
48
49     // Count character frequencies for both substrings.
50     for (int i = 0; i < half_len; i++) {
51         freq1[s1[i] - 'a']++;
52         freq2[s2[i] - 'a']++;
53     }
54
55     int changes = 0;
56     // Compare frequencies to find the number of characters that need to be
57     changed.
58     for (int i = 0; i < 26; i++) {
59         if (freq1[i] > freq2[i]) {
60             changes += freq1[i] - freq2[i];
61         }
62     }
63
64     free(s1);
65     free(s2);
66
67     return changes;
68 }
69
70 int main()
71 {

```

```





72 FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
73
74 int q = parse_int(ltrim(rtrim(readline())));
75
76 for (int q_itr = 0; q_itr < q; q_itr++) {
77     char* s = readline();
78
79     int result = anagram(s);
80
81     fprintf(fptr, "%d\n", result);
82 }
83
84 fclose(fptr);
85
86 return 0;
87 }
88
89 char* readline() {
90     size_t alloc_length = 1024;
91     size_t data_length = 0;
92
93     char* data = malloc(alloc_length);
94
95     while (true) {
96         char* cursor = data + data_length;
97         char* line = fgets(cursor, alloc_length - data_length, stdin);
98
99         if (!line) {
100             break;
101         }
102
103         data_length += strlen(cursor);
104
105         if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
106     {
107         break;
108     }
109
110     alloc_length <= 1;
111
112     data = realloc(data, alloc_length);
113
114     if (!data) {
115         data = '\0';
116
117         break;
118     }
119
120     if (data[data_length - 1] == '\n') {
121         data[data_length - 1] = '\0';
122
123         data = realloc(data, data_length);
124
125         if (!data) {
126             data = '\0';
127         }
128     } else {
129         data = realloc(data, data_length + 1);
130
131         if (!data) {
132             data = '\0';
133         } else {
134

```

```

13         data[data_length] = '\0';
14     }
15 }
16
17 return data;
18 }
19
20 char* ltrim(char* str) {
21     if (!str) {
22         return '\0';
23     }
24
25     if (!*str) {
26         return str;
27     }
28
29     while (*str != '\0' && isspace(*str)) {
30         str++;
31     }
32
33     return str;
34 }
35
36 char* rtrim(char* str) {
37     if (!str) {
38         return '\0';
39     }
40
41     if (!*str) {
42         return str;
43     }
44
45     char* end = str + strlen(str) - 1;
46
47     while (end >= str && isspace(*end)) {
48         end--;
49     }
50
51     *(end + 1) = '\0';
52
53     return str;
54 }
55
56 int parse_int(char* str) {
57     char* endptr;
58     int value = strtol(str, &endptr, 10);
59
60     if (endptr == str || *endptr != '\0') {
61         exit(EXIT_FAILURE);
62     }
63
64     return value;
65 }

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Hidden case	 Success	5	0.007 sec	7.38 KB
Testcase 2	Easy	Hidden case	 Success	5	0.0069 sec	7.25 KB
Testcase 3	Easy	Hidden case	 Success	5	0.0079 sec	7.25 KB
Testcase 4	Easy	Hidden case	 Success	5	0.0075 sec	7.25 KB

Testcase 5	Easy	Hidden case	✔ Success	5	0.0066 sec	7.25 KB
Testcase 6	Easy	Hidden case	✔ Success	5	0.017 sec	8.13 KB
Testcase 7	Easy	Hidden case	✔ Success	5	0.0084 sec	7.88 KB
Testcase 8	Easy	Hidden case	✔ Success	5	0.018 sec	8.25 KB
Testcase 9	Easy	Hidden case	✔ Success	5	0.0089 sec	7.63 KB
Testcase 10	Easy	Hidden case	✔ Success	5	0.0102 sec	8.13 KB
Testcase 11	Easy	Hidden case	✔ Success	5	0.0089 sec	7.38 KB
Testcase 12	Easy	Hidden case	✔ Success	5	0.0203 sec	8.25 KB
Testcase 13	Easy	Hidden case	✔ Success	5	0.0125 sec	8.13 KB
Testcase 14	Easy	Hidden case	✔ Success	5	0.0102 sec	7.75 KB
Testcase 15	Easy	Sample case	✔ Success	0	0.0068 sec	7.13 KB
Testcase 16	Easy	Sample case	✔ Success	0	0.0074 sec	6.88 KB

No Comments

PDF generated at: 14 Aug 2025 07:25:50 UTC