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:	Algorithms 280/280 Core CS 280/280 Data Structures 105/105 Easy 280/280 LCM 105/105 Least Common Multiple 105/105 Math 105/105 Problem Solving 105/105 Strings 175/175 gcd 105/105 greatest common divisor 105/105 problem-solving 280/280 sets 105/105

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 › 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 = 'aaab' (first query)
baa     s = 'baa'  (second query)
aaa     s = '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**

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);

// Helper function to check if a substring is a palindrome
bool isPalindrome(char* str, int start, int end) {
    while (start < end) {
        if (str[start] != str[end]) {
            return false;
        }
        start++;
        end--;
    }
    return true;
}

/*
 * Complete the 'palindromeIndex' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING s as parameter.
 */

int palindromeIndex(char* s) {
    int len = strlen(s);
    int start = 0;
    int end = len - 1;

    while (start < end) {
        if (s[start] != s[end]) {
            // Found a mismatch. Check if removing start or end character
makes it a palindrome.
            // Check if removing the character at 'start' makes the rest of
the string a palindrome.
            if (isPalindrome(s, start + 1, end)) {
                return start;
            }
            // Check if removing the character at 'end' makes the rest of the
string a palindrome.
            if (isPalindrome(s, start, end - 1)) {
                return end;
            }
            // If neither works, it's not possible to form a palindrome by
removing one character.

```

```
            // The problem statement implies one solution will exist if it's not already a palindrome,
            // but this is a failsafe.
            return -1;
        }
        start++;
        end--;
    }

    // The string is already a palindrome.
    return -1;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int q = parse_int(ltrim(rtrim(readline())));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        char* s = readline();

        int result = palindromeIndex(s);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }
```

```
    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
```

```
    }

    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 betwen 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**

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'getTotalX' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 * 1. INTEGER_ARRAY a
 * 2. INTEGER_ARRAY b
 */

```

```c
int getTotalX(int a_count, int* a, int b_count, int* b) {
    int count = 0;

    // Constraints state values are between 1 and 100.
    // The range of numbers to check is from the largest element of 'a' to
the smallest element of 'b'.
    int max_a = 0;
    for (int i = 0; i < a_count; i++) {
        if (a[i] > max_a) {
            max_a = a[i];
        }
    }

    int min_b = 101; // Since values are <= 100
    for (int i = 0; i < b_count; i++) {
        if (b[i] < min_b) {
            min_b = b[i];
        }
    }

    // Iterate through all numbers in the possible range.
    for (int i = max_a; i <= min_b; i++) {
        bool condition1 = true;
        bool condition2 = true;

        // Condition 1: All elements of 'a' are factors of 'i'.
        for (int j = 0; j < a_count; j++) {
            if (i % a[j] != 0) {
                condition1 = false;
                break;
            }
        }

        // If condition 1 is met, check condition 2.
        if (condition1) {
            // Condition 2: 'i' is a factor of all elements of 'b'.
            for (int k = 0; k < b_count; k++) {
                if (b[k] % i != 0) {
                    condition2 = false;
                    break;
                }
            }
        }

        if (condition1 && condition2) {
            count++;
        }
    }

    return count;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char** first_multiple_input = split_string(rtrim(readline()));

    int n = parse_int(*(first_multiple_input + 0));

    int m = parse_int(*(first_multiple_input + 1));

    char** arr_temp = split_string(rtrim(readline()));
```

```
    int* arr = malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        int arr_item = parse_int(*(arr_temp + i));

        *(arr + i) = arr_item;
    }

    char** brr_temp = split_string(rtrim(readline()));

    int* brr = malloc(m * sizeof(int));

    for (int i = 0; i < m; i++) {
        int brr_item = parse_int(*(brr_temp + i));

        *(brr + i) = brr_item;
    }

    int total = getTotalX(n, arr, m, brr);

    fprintf(fptr, "%d\n", total);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
{
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);
```

```
        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
```

```
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

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
xaxbbbxx
```

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**

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);



/*
 * Complete the 'anagram' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING s as parameter.
 */

int anagram(char* s) {
    int len = strlen(s);

    // If the string length is odd, it's impossible to split into two equal-length substrings.
    if (len % 2 != 0) {
        return -1;
    }

    int half_len = len / 2;
    char* s1 = (char*)malloc((half_len + 1) * sizeof(char));
    char* s2 = (char*)malloc((half_len + 1) * sizeof(char));

    // Split the string into two halves.
    strncpy(s1, s, half_len);
    s1[half_len] = '\0';
    strncpy(s2, s + half_len, half_len);
    s2[half_len] = '\0';

    int freq1[26] = {0};
    int freq2[26] = {0};

    // Count character frequencies for both substrings.
    for (int i = 0; i < half_len; i++) {
        freq1[s1[i] - 'a']++;
        freq2[s2[i] - 'a']++;
    }

    int changes = 0;
    // Compare frequencies to find the number of characters that need to be changed.
    for (int i = 0; i < 26; i++) {
        if (freq1[i] > freq2[i]) {
            changes += freq1[i] - freq2[i];
        }
    }

    free(s1);
    free(s2);

    return changes;
}

int main()
{
```

```c
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int q = parse_int(ltrim(rtrim(readline())));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        char* s = readline();

        int result = anagram(s);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n')
{
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
```

```
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

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