

**Name:- Kuldeep**

**UID:- 22BCS10071**

**Section:- KPIT\_901/A**

## **Day 3**

### **Function \$ RECURSION DSA QUESTION**

#### **Very Easy**

#### **Q .Fibonacci Series Using Recursion**

The Fibonacci numbers, commonly denoted  $F(n)$  form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F(0) = 0, F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$$

Given  $n$ , calculate  $F(n)$ .

#### **Example 1:**

Input:  $n = 2$

Output: 1

Explanation:  $F(2) = F(1) + F(0) = 1 + 0 = 1$

#### **Example 2:**

Input:  $n = 3$

Output: 2

Explanation:  $F(3) = F(2) + F(1) = 1 + 1 = 2$ .

**Constraints:**

$0 \leq n \leq 30$

Code:-

```
#include <iostream>
```

```
using namespace std;
```

```
int fibonacci(int n) {  
    if (n == 0) return 0;  
    if (n == 1) return 1;  
    return fibonacci(n - 1) + fibonacci(n - 2);  
}
```

```
int main() {  
    int n = 2;  
    cout << "F(" << n << ") = " << fibonacci(n) << endl;  
    return 0;  
}
```

Output:-

```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp
code\winning camp question\" && g++ fabonica_series.
cpp -o fabonica_series && "c:\Users\Rohit
Thakur\Desktop\cpp code\winning camp
question\"fabonica_series
F(2) = 1

[Done] exited with code=0 in 0.474 seconds
```

## Easy

### **Q1 Find GCD of Number Using Function**

Given an integer array nums, return the greatest common divisor of the smallest number and largest number in nums.

The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.

#### **Example 1:**

Input: nums = [2,5,6,9,10]

Output: 2

Explanation:

The smallest number in nums is 2.

The largest number in nums is 10.

The greatest common divisor of 2 and 10 is 2.

**Example 2:**

Input: nums = [7,5,6,8,3]

Output: 1

Explanation:

The smallest number in nums is 3.

The largest number in nums is 8.

The greatest common divisor of 3 and 8 is 1.

**Example 3:**

Input: nums = [3,3]

Output: 3

Explanation:

The smallest number in nums is 3.

The largest number in nums is 3.

The greatest common divisor of 3 and 3 is 3.

**Constraints:**

$2 \leq \text{nums.length} \leq 1000$

$1 \leq \text{nums}[i] \leq 1000$

**Code:-**

```
#include <iostream>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
int gcd(int a, int b) {
```

```
    while (b != 0) {
```

```
        int temp = b;
```

```
        b = a % b;
```

```
        a = temp;
```

```
    }
```

```
    return a;
```

```
}
```

```
int findGCD(std::vector<int>& nums) {
```

```
    int minNum = *std::min_element(nums.begin(), nums.end());
```

```
    int maxNum = *std::max_element(nums.begin(), nums.end());
```

```
    return gcd(minNum, maxNum);
```

```
}
```

```
int main() {
```

```
    std::vector<int> nums = {2, 5, 6, 9, 10};
```

```
    std::cout << findGCD(nums) << std::endl; // Output: 2
```

```
    return 0;
}
```

**Output:-**

```
[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp
code\winning camp question\" && g++ tempCodeRunnerFi
cpp -o tempCodeRunnerFile && "c:\Users\Rohit
Thakur\Desktop\cpp code\winning camp
question\tempCodeRunnerFile
2
[Done] exited with code=0 in 0.586 seconds
```

## Medium

### **Q: Longest Substring Without Repeating Characters**

You are working on building a text editor application. One of the features you're focusing on is a "word suggestion" system. The editor needs to identify the longest sequence of characters typed without repeating any letters to suggest potential words or phrases. To accomplish this, you must efficiently find the length of the longest substring of unique characters as the user types.

#### **Description:**

Write a function that takes a string as input and returns the length of the longest substring without repeating characters. A substring is a contiguous sequence of characters within the string.

#### **Example 1:**

Input: "abcabcbb"

Output: 3

Explanation: The longest substring without repeating characters is "abc", which has length 3.

### Constraints:

- The input string will have a length between 1 and  $10^4$ .
- The characters in the string are printable ASCII characters.

---

This problem is a classic example of the **sliding window** technique,

Code:-

```
#include <iostream>

#include <unordered_map>

#include <string>

int lengthOfLongestSubstring(const std::string& s) {

    std::unordered_map<char, int> charIndexMap;

    int maxLength = 0;

    int start = 0;

    for (int i = 0; i < s.length(); ++i) {

        if (charIndexMap.find(s[i]) != charIndexMap.end()) {

            start = std::max(start, charIndexMap[s[i]] + 1);

        }

        charIndexMap[s[i]] = i;

        maxLength = std::max(maxLength, i - start + 1);

    }
```

```

    }

    return maxLength;
}

int main() {

    std::string input = "abcabcbb";

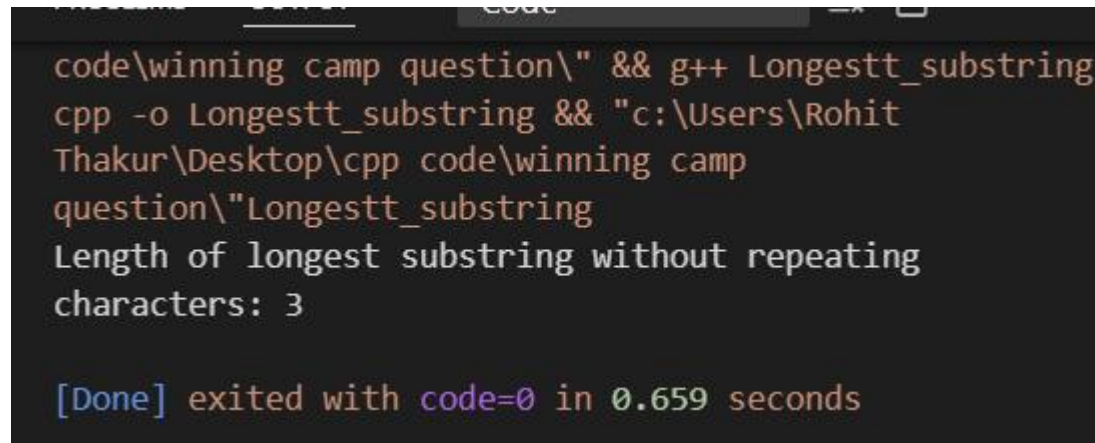
    std::cout << "Length of longest substring without repeating characters: "

        << lengthOfLongestSubstring(input) << std::endl;

    return 0;
}

```

Output:-



```

code\winning camp question\" && g++ Longestt_substring
cpp -o Longestt_substring && "c:\Users\Rohit
Thakur\Desktop\cpp code\winning camp
question\"Longestt_substring
Length of longest substring without repeating
characters: 3

[Done] exited with code=0 in 0.659 seconds

```

[Hard](#)

**Q. Wildcard Matching**



Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '\*' where:

'?' Matches any single character.

'\*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

### **Example 1:**

Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".

### **Example 2:**

Input: s = "aa", p = "\*"

Output: true

Explanation: '\*' matches any sequence.

### **Example 3:**

Input: s = "cb", p = "?a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

### **Constraints:**

$0 \leq s.length, p.length \leq 2000$

s contains only lowercase English letters.

p contains only lowercase English letters, '?' or '\*'.

**Code:-**

```
#include <iostream>
```

```
#include <vector>
```

```
#include <string>
```

```
bool isMatch(const std::string& s, const std::string& p) {
```

```
    int sLen = s.size(), pLen = p.size();
```

```
    std::vector<std::vector<bool>> dp(sLen + 1, std::vector<bool>(pLen + 1, false));
```

```
    dp[0][0] = true;
```

```
    for (int j = 1; j <= pLen; ++j) {
```

```
        if (p[j - 1] == '*') {
```

```
            dp[0][j] = dp[0][j - 1];
```

```
        }
```

```
    }
```

```
    for (int i = 1; i <= sLen; ++i) {
```

```
        for (int j = 1; j <= pLen; ++j) {
```

```
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
```

```

        dp[i][j] = dp[i - 1][j - 1];
    } else if (p[j - 1] == '*') {
        dp[i][j] = dp[i][j - 1] || dp[i - 1][j];
    }
}

}

return dp[sLen][pLen];
}

int main() {
    std::string s = "aa";
    std::string p = "a";

    std::cout << std::boolalpha << isMatch(s, p) << std::endl; // Output: false

    return 0;
}

```

**Output:-**

```

[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp
code\winning camp question\" && g++ wildcard_matching.
cpp -o wildcard_matching && "c:\Users\Rohit
Thakur\Desktop\cpp code\winning camp
question\"wildcard_matching
false

[Done] exited with code=0 in 0.626 seconds

```

## Very Hard

### **Q. Special Binary String**

Special binary strings are binary strings with the following two properties:

The number of 0's is equal to the number of 1's.

Every prefix of the binary string has at least as many 1's as 0's.

You are given a special binary string  $s$ .

A move consists of choosing two consecutive, non-empty, special substrings of  $s$ , and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.

#### **Example 1:**

Input:  $s = "11011000"$

Output:  $"11100100"$

Explanation: The strings  $"10"$  [occurring at  $s[1]$ ] and  $"1100"$  [at  $s[3]$ ] are swapped.

This is the lexicographically largest string possible after some number of swaps.

#### **Example 2:**

Input: s = "10"

Output: "10"

**Constraints:**

1 <= s.length <= 50

s[i] is either '0' or '1'.

s is a special binary string.

**Code:-**

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
std::string makeLargestSpecial(std::string s) {
```

```
    std::vector<std::string> special;
```

```
    int count = 0, last = 0;
```

```
    for (int i = 0; i < s.size(); ++i) {
```

```
        count += (s[i] == '1') ? 1 : -1;
```

```
        if (count == 0) {
```

```
            special.push_back("1" + makeLargestSpecial(s.substr(last + 1, i - last - 1)) + "0");
```

```
            last = i + 1;
```

```

    }
}

std::sort(special.rbegin(), special.rend());

std::string result;

for (const auto& str : special) {
    result += str;
}

return result;
}

int main() {
    std::string s = "11011000";

    std::string result = makeLargestSpecial(s);

    std::cout << result << std::endl; // Output: "11100100"

    return 0;
}

```

**Output:-**

```

[Running] cd "c:\Users\Rohit Thakur\Desktop\cpp
code\winning camp question\" && g++
special_binary_search.cpp -o special_binary_search &&
"c:\Users\Rohit Thakur\Desktop\cpp code\winning camp
question\special_binary_search
11100100

[Done] exited with code=0 in 0.759 seconds

```