Lab 07 - Recursive Functions

CO 222 Programming Methodology

Department of Computer Engineering

University of Peradeniya

January 8, 2024

---

1. Objective

The objective of this lab is to gain practical experience in implementing recursive functions using C programming. Recursive functions provide an elegant and powerful approach to problem-solving, breaking down complex tasks into simpler subproblems.

2. Introduction

Welcome to the Recursive Functions Lab. This lab is designed to guide you through the basics of recursion in C programming. Unlike iterative approaches, recursive solutions involve breaking down a problem into smaller instances of the same problem until reaching a base case. This technique leverages the call stack, allowing functions to call themselves. The lab aims to deepen your understanding of recursive thinking and its application to various programming challenges. Happy coding, and may the force be with you!

3. Fibonacci Sequence

Your first task is to implement a recursive function to generate the Fibonacci sequence. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones,

n	0	1	2	3	4	5	6	7	8	...
Fib(n)	0	1	1	2	3	5	8	13	21	...

The task given to you is to write a function called `fib(n)` where `n` is the n^{th} term of the fibonacci sequence. This function should be able to calculate the n^{th} term of the Fibonacci sequence using recursion assuming that `n` is a non-negative integer. Additionally, provide comments in your code to explain the logic behind the base cases employed to ensure the proper functioning of the recursive solution. You can use the above table to validate the correctness of your function.

Now, for fun try inputting a relatively large number (take 50 as an example) as ‘n’ in your program. As you may notice, the execution time tends to increase significantly as 'n' grows. (In case you didn’t know you can use Ctrl+C inside the terminal to forcibly exit from your program if needed!) This is primarily due to the recursive nature of the Fibonacci function, which results

in redundant calculations for smaller terms. But for now since you have completed the first task you can directly go to the second task in the next section but after you complete the lab please go through the additional reading section if you are curious about why this happens and how to solve this problem.

4. Evaluating a palindrome (But this time using recursion!)

Hope that you remember what a palindrome is from our previous lab. But in case you forgot, a **palindrome** is a word, number, phrase, or any other sequence of symbols that reads the same backward as forwards, such as *madam* or *racecar*. Your second task is to write a program that takes a string as input and determines whether it is a palindrome or not using a recursive function. Assume that the given string will only contain simple case letters or non-alphabetic characters.

I understand that for some of you, this task may seem straightforward. However, before diving into coding, I encourage you to take a moment and sketch out your recursive algorithm on a piece of paper. Visualize how you intend to check whether the given string is a palindrome, and clearly identify the base cases that are crucial for the proper functioning of your recursive solution.

Consider using diagrams or illustrations to represent the steps of your algorithm. (Don't need to be in pseudocode or flowchart form necessarily, any kind of figure that effectively communicates your thought process is acceptable) Once you have your algorithm outlined, share it with an instructor and explain your reasoning and the logic behind your recursive solution.

After reviewing and discussing your algorithm with an instructor, proceed to implement your solution in code. As always, test your program with various input strings to verify its correctness.

```
Enter a sentence: saippuakivikauppias
The string is a palindrome.
```

```
Enter a sentence: sasuke
The string is not a palindrome.
```

```
Enter a sentence: eevee
The string is a palindrome.
```

5. Binary Search

For the final task, you must implement a binary search algorithm in C. Binary search is an efficient algorithm for finding a specific value in a **sorted array**. The basic idea is to divide the array into two halves and recursively narrow down the search space until the target value is found or the search space is empty. For this you have to create a recursive function called `binarySearch` with the following parameters,

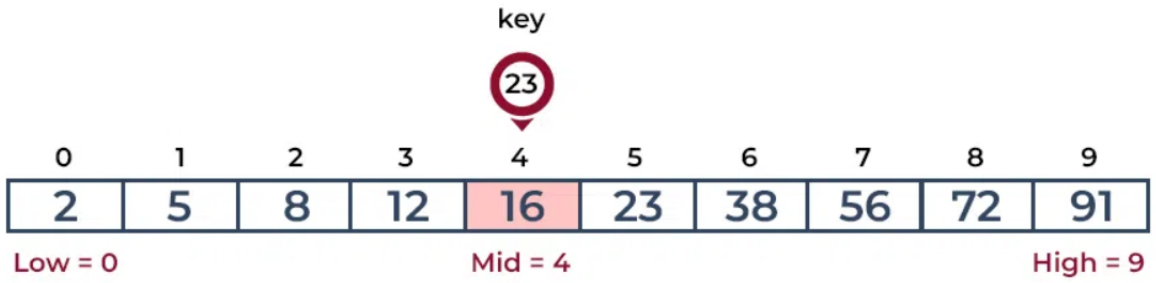
```
binarySearch(int arr[], int low, int high, int key)
```

Assume that you are given the number of elements inside a sorted array, non-negative integer value of each element inside the array and the target value you need to find. In this example we will try to find a target value (*key*) of 23 inside the sorted array [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

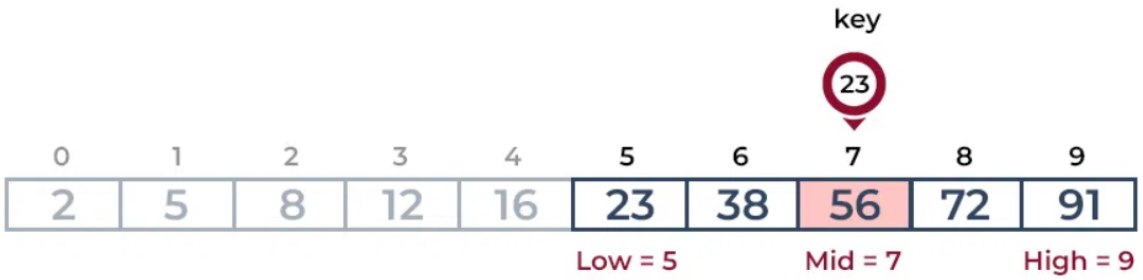
First, we have to find the index of the mid element in our array. This can be done by calculating,

$$Mid = (Low + High) // 2$$

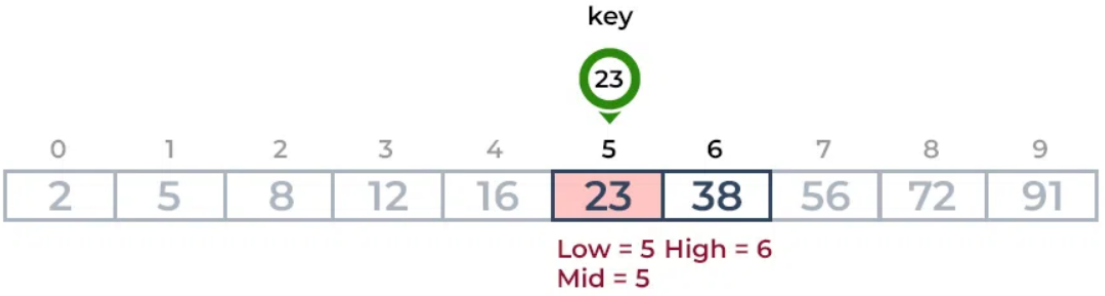
Be careful to use integer division as we are talking about indexes in an array.



Now we compare our key value with the value of the mid element. Since 23 is larger than 16 we have to move our search space to the right. Thus, we will recursively call our binary search function with a new Low index (Mid + 1) and High index (Same as previous)



This time key is less than the current Mid value of 56. Thus we have to move our search space to the left.



This time we see that our Mid value is equal to the key value, So we have found the target and thus, stop the search and return the index of the Mid element.

In case, even after your search space reduces to only one element, and you haven't found the key, it implies that the target value is not present in the array. This situation represents another crucial base case in the binary search algorithm. When the search space narrows down to a single element, and the target value is not yet found, it indicates that the key is not within the array and you must return -1 as output to signal the absence of the target value.

Now write a small program to test out your binary search function. The first input will be the number(n) of elements inside the array. And the next 'n' number of inputs will be each non-negative integer element inside the array given in the sorted order. Finally you have to give the value of the element you need to search for.

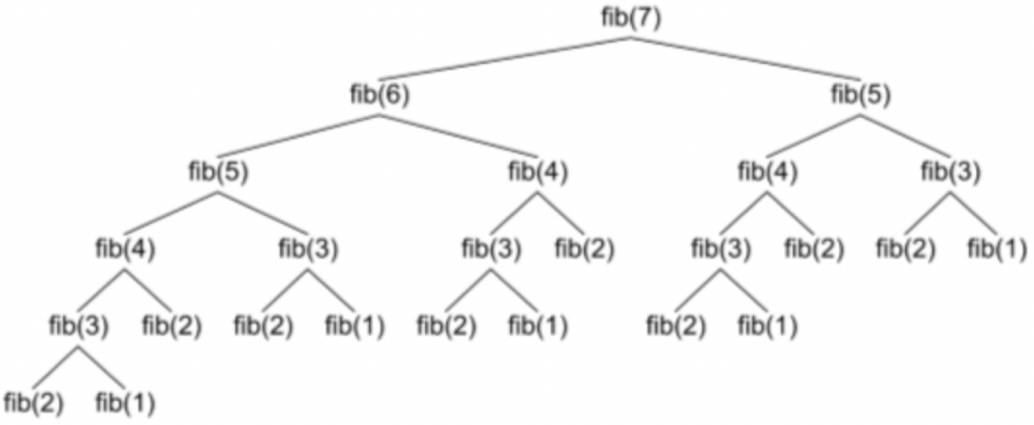
```
Enter the number of elements: 5
Enter the elements in sorted order:
1
3
5
6
9
Enter the element to search: 6
Element 6 found at index 3
```

```
Enter the number of elements: 5
Enter the elements in sorted order:
1
3
5
6
9
Enter the element to search: 12
Element 12 not found in the array
```

As a final word of caution, Since binary search is a very common algorithm, you might be tempted to find solutions online or use tools like ChatGPT. Yet, we strongly recommend trying to write the code on your own. This way, you'll get a better handle on how recursive algorithms work and steer clear of unintended plagiarism.

6. Additional Reading

In the above task one when you tried to find the fibonacci sequence you may have noticed that for relatively large numbers of 'n' the execution time tends to increase significantly. This behavior primarily happens due to the recursive nature of the Fibonacci function, which results in redundant calculations for smaller terms. For an example take a look at the following figure which shows the recursion tree (visual representation that illustrates the recursive calls made by a recursive algorithm where each node represents a recursive call, and branches emanating from a node denote subsequent recursive calls made within that call) for calculating `fib(7)`,



As you can see, we make many redundant function calls where we calculate the same Fibonacci value repeatedly. Thus, as our input 'n' becomes larger, our call stack will grow by 2^N (since every node will split into two sub-branches). The most intuitive solution to this problem is to memorize the Fibonacci values that we have previously calculated and reuse those values when needed to find the Fibonacci value of larger inputs. This way, we will only calculate each value once and thus save time.

For example, you can use a global variable to store the return value of each function call, and within the same function, check whether that variable already contains the needed value before performing the calculation. This approach is known as memoization. It is one of the common approaches for optimizing recursive algorithms and falls under the broader umbrella of concept known as dynamic programming. Which involves breaking down a problem into smaller subproblems and solving each subproblem only once, storing the results to avoid redundant computations and improve overall efficiency.

While you may not currently require this information at your current level of knowledge, gaining an understanding of it can be beneficial. Moreover, you will delve deeper into this concept next year when you follow the course Data Structures and Algorithms. Additionally, familiarity with this fundamental concept will prove invaluable, particularly if you intend to engage in competitive programming.

7. Submission

You need to complete and show your work for task 1 and 2 (section 3 & 4) to instructors during the lab hours. Additionally, you will need to submit the codes for all the completed tasks using HackerRank before the deadline.

You must submit all the code through FEeLS as well.

8. Important

As always, you should write your programs individually, and under no circumstance, you should copy somebody else's code. Submitting AI generated answers or copying someone else's code (including your groupmates') or showing your source code to anyone else will earn you zero marks for the whole lab exercise.

9. Deadline

Hackerrank: Submission link will be sent on the 11th 0f January 2024 and you should log in through the link within 24 hours (Otherwise link will not be working). After login, you will be given 72 hours to complete the assignment.

FEeLS: 11:55 P.M., 15th 0f January 2024

10. References

https://en.wikipedia.org/wiki/Palindrome

https://www.geeksforgeeks.org/binary-search/