

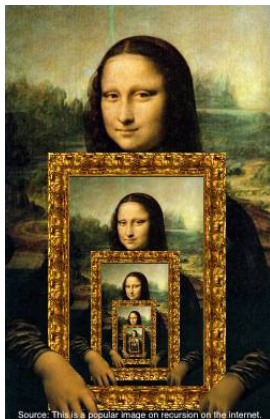
ACOL106: Data Structures and Algorithms

Ragesh Jaiswal, IITD

Introduction

Digression: Binary Search → Recursive Functions

- Recursion: Self reference.



- In our context, we talk about recursive functions.

Introduction

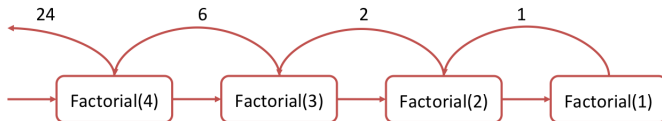
Digression: Binary Search → Recursive Functions

- Recursive function: A function that makes a call to itself.

Algorithm

`Factorial( $n$ )`

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)



Introduction

Digression: Binary Search → Recursive Functions

- Recursive function: A function that makes a call to itself.

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

- Base case: Returns result for small value of inputs. Defines the recursion termination condition.
- Reduction step: Assuming that the function returns correct value for smaller inputs use function calls on smaller inputs to compute the result on the given input.

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions?

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**
- Question: Is it always possible to avoid recursive functions?

Algorithm

Factorial(n)

- If ($n = 0$ or $n = 1$) return(1)
- $f \leftarrow \text{Factorial}(n - 1)$
- return($n \cdot f$)

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**
- Question: Is it always possible to avoid recursive functions?
Yes.

Algorithm

Factorial-iterative(n)

- $f \leftarrow 1$
- for $i = 1$ to n
 - $f \leftarrow f \cdot i$
- return(f)

- In fact, there is some efficiency advantage in not using recursive functions.

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**
- Question: Is it always possible to avoid recursive functions?
Yes.
- In fact, there is some efficiency advantage in not using recursive functions as function calls involve various time/space overheads.
- Why is recursion used then?

Introduction

Digression: Binary Search → Recursive Functions

- Question: How do we prove correctness of recursive functions? **Induction**
- Question: Is it always possible to avoid recursive functions?
Yes.
- In fact, there is some efficiency advantage in not using recursive functions as function calls involve various time/space overheads.
- Why is recursion used then?
 - In many cases, using recursion makes the program much simpler and easy to understand and analyse.
 - Many problems in Computer Science have inherent recursive structures (e.g., Fibonacci sequence)

Introduction

Digression: Binary Search → Recursive Functions → Fibonacci Sequence

- The Fibonacci sequence is defined in the following recursive manner:
 - Base case: $F(0) = 0, F(1) = 1$
 - For all $n > 1$, $F(n) = F(n-1) + F(n-2)$
- So, the sequence is:
 - $F(0) = 0$
 - $F(1) = 1$
 - $F(2) = F(1) + F(0) = 1 + 0 = 1$
 - $F(3) = F(2) + F(1) = 1 + 1 = 2$
 - $F(4) = F(3) + F(2) = 2 + 1 = 3$
 - $\vdots$

Introduction

Digression: Binary Search → Recursive Functions → Fibonacci Sequence

- The Fibonacci sequence is defined in the following recursive manner:
 - Base case: $F(0) = 0$, $F(1) = 1$
 - For all $n > 1$, $F(n) = F(n-1) + F(n-2)$
- So, the sequence is: 0, 1, 1, 2, 3, 5, 8, 13, ...
- The problem itself is defined in a recursive manner. So, it is natural to write a recursive method to solve this.

Algorithm

Recursive-Fib(n)

If ($n = 0$ or $n = 1$) return(n)

- return(Recursive-Fib($n-1$) + Recursive-Fib($n-2$))

Introduction

Digression: Binary Search → Recursive Functions → Fibonacci Sequence

- The Fibonacci sequence is defined in the following recursive manner:
 - Base case: $F(0) = 0, F(1) = 1$
 - For all $n > 1$, $F(n) = F(n-1) + F(n-2)$
- So, the sequence is: 0, 1, 1, 2, 3, 5, 8, 13, ...
- The problem itself is defined in a recursive manner. So, it is natural to write a recursive method to solve this.

Algorithm

Rfib(n)

If ($n = 0$ or $n = 1$) return(n)

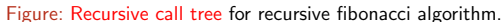
- return(Rfib($n-1$) + Rfib($n-2$))

- How do we analyse the running time of the above algorithm?

Digression: Binary Search $\rightarrow$ Recursive Functions $\rightarrow$ Fibonacci Sequence

$$\text{Rfib}(n)$$

- How do we analyse the running time of the above algorithm?



End