Mathematical Analysis of Recursive Algorithms

CS 350 – Algorithms and Complexity
Paul Doliotis
Portland State University

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.
- **Recursive data types**: data types that are defined using references to themselves.
- Example?

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.
- **Recursive data types**: data types that are defined using references to themselves.
- Example? Nodes in the implementation of linked lists.

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.
- **Recursive data types**: data types that are defined using references to themselves.
- Example? Nodes in the implementation of linked lists.
- In all recursive concepts, there is one or more **base cases**. No recursive concept can be understood without understanding its base cases.

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.
- **Recursive data types**: data types that are defined using references to themselves.
- Example? Nodes in the implementation of linked lists.
- In all recursive concepts, there is one or more **base cases**. No recursive concept can be understood without understanding its base cases.
- What is the base case for nodes?

Recursion

- Recursion is a fundamental concept in computer science.
- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- **Recursive functions**: functions that call themselves.
- **Recursive data types**: data types that are defined using references to themselves.
- Example? Nodes in the implementation of linked lists.
- In all recursive concepts, there is one or more **base cases**. No recursive concept can be understood without understanding its base cases.
- What is the base case for nodes?
  - A node pointing to NULL.

Recursive Algorithms

- **Recursive algorithms**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- A recursive algorithm can always be implemented both using recursive functions, and without recursive functions.
- Example of a recursive function:

Recursive Algorithms

- **<u>Recursive algorithms</u>**: algorithms that solve a problem by solving one or more smaller instances of the same problem.
- A recursive algorithm can always be implemented both using recursive functions, and without recursive functions.
- Example of a recursive function: the factorial.
  - How is factorial(3) evaluated?

Recursive Definition:

```
int factorial(int N)
{
   if (N == 0) return 1;
   return N*factorial(N-1);
}
```

Non-Recursive Definition :

```
int factorial(int N)
{
  int result = 1;
  int i;
  for (i = 2; i <= N; i++) result *= i;
  return result;
}
```

Runtime Analysis of Recursive Algorithms

- Let's discuss runtime analysis for the recursive version.
- First we need to decide on parameter *n* indicating input size

Runtime Analysis of Recursive Algorithms

- Let’s discuss runtime analysis for the recursive version.
- First we need to decide on parameter *n* indicating input size
- Second, define the basic operation (multiplication). Other ideas?

Runtime Analysis of Recursive Algorithms

- Let’s discuss runtime analysis for the recursive version.
- First we need to decide on parameter *n* indicating input size
- Second, define the basic operation (multiplication). Other ideas?
- Third, determine if best, average, worst case are different. Usually the most useful is the worst case.

Runtime Analysis of Recursive Algorithms

- Third, determine if best, average, worst case are different. Most often the most useful is the worst case.
- Fourth, setup recurrence relation for the number of times the basic operation is performed.
  - Let's denote with *M(n)* the number of multiplications
  - Need to find Initial condition
- Finally, we have to solve the recurrence relation

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n-1) + 1$, $n>0$ (recurrence relation for runtime function)

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n-1) + 1$, $n>0$ (recurrence relation for runtime function)
- Why $M(n-1)$?
  - To compute F($n-1$) in the algorithm

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n-1) + 1$, $n>0$ (recurrence relation for runtime function)
- Why $M(n-1)$?
  – To compute F($n-1$) in the algorithm
- Why constant 1 in the recursive formula?
  – To multiply F($n-1$) by n

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n-1) + 1$, $n>0$ (recurrence relation for runtime function)
- Why $M(n-1)$?
  - To compute F(n-1) in the algorithm
- Why constant 1 in the recursive formula?
  - To multiply F(n-1) by n
- What about the initial condition?
  - M(0) = 0, no multiplication when n = 0 (check pseudocode to see why this is true)

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n\text{-}1) + 1\ ,\ n>0$ (recurrence relation for runtime function)
- Why $M(n\text{-}1)$?
  - To compute F(*n-1*) in the algorithm
- Why constant 1 in the recursive expression?
  - To multiply F(*n-1*) by *n*
- What about the initial condition?
  - M(0) = 0, no multiplication when n = 0 (check pseudocode to see why this is true)
- Now let’s solve the recurrence relation!

Runtime Analysis of Recursive Algorithms

- $M(n) = M(n-1) + 1$, $n>0$ and $M(0) = 0$
- $M(n) = M(n-1) + 1$

$$
\begin{aligned}
&= [M(n-2) + 1] + 1 \\
&= M(n-2) + 2 \\
&\quad \ldots. \\
&= M(n-i) + i, \; i <= n \quad \text{(\underline{generalise})}
\end{aligned}
$$

Put $i=n$: $M(n) = M(0) + n$

$$= n$$

Mathematical Analysis

1. Decide on a parameter (or parameters) for measuring the size of the input
2. Identify the Algorithm's basic operation
3. Determine if best, worst, and average case will be different
4. Describe Recurrence relation for the number of basic operations (Do not forget initial conditions)
5. Solve the recurrence relation and order of growth

Example

- Solve this recurrence relation:

$$x(n) = x(n-1) + 5 \text{ , } n>1$$
$$x(1) = 0$$

Example

- Solve this recurrence relation:

$$x(n) = x(n-1) + n \,,\; n>0$$
$$x(0) = 0$$

Recursive Vs. Non-Recursive Implementations

- In some cases, recursive functions are much easier to read.
  - They make crystal clear the mathematical structure of the algorithm.
- However, any recursive function can also be written in a non-recursive way.
- Oftentimes recursive functions run slower. Why?

Recursive Vs. Non-Recursive Implementations

- In some cases, recursive functions are much easier to read.
  - They make crystal clear the mathematical structure of the algorithm.
- However, any recursive function can also be written in a non-recursive way.
- Oftentimes recursive functions run slower. Why?
  - Recursive functions generate many function calls.
  - The CPU has to pay a price (perform a certain number of operations) for each function call.
- Non-recursive implementations are oftentimes somewhat uglier (and more buggy, harder to debug) but more efficient.
  - Compromise: make first version recursive, second non-recursive.