Data Structures & Algorithms for Problem Solving (CS1.304)

Lecture # 03

Avinash Sharma
Center for Visual Information Technology (CVIT),
IIIT Hyderabad

Recursion

- Recursion is a mathematical technique that evaluates a function by calling the same function repeatedly on smaller inputs.
- Most programming languages support such a style of programming.
  - Often very elegant to study.
- Helps in problem solving too.

Recursion

- Computing the nth Fibonacci number.
- Let F(n) denote the nth Fibonacci number with F(0) = 0, and F(1) = 1.
- We know that F(n) is guided by the recurrence relation:

$$F(n) = F(n-1) + F(n-2).$$

- A program to achieve this computation is shown next.

Recursion

```
Program Fibonacci(n)
Begin
if n == 0  return 0;
if n == 1 return 1;
else return Fibonacci(n-1) + Fibonacci(n-2)
End.
```

- The program is neat and easy to understand.
- There is however one small pitfall.

Motivation

- Consider the following program to compute the n^{th} Fibonacci number.

```
Program Fibonacci(n)
Begin
if n == 0  return 0;
if n == 1 return 1;
else return Fibonacci(n-1) + Fibonacci(n-2)
End.
```

- Question: Compute the runtime of the program to obtain the n^{th} Fibonacci number. Include a brief justification.

Motivation

$T(n) = T(n-1) + T(n-2) + c$

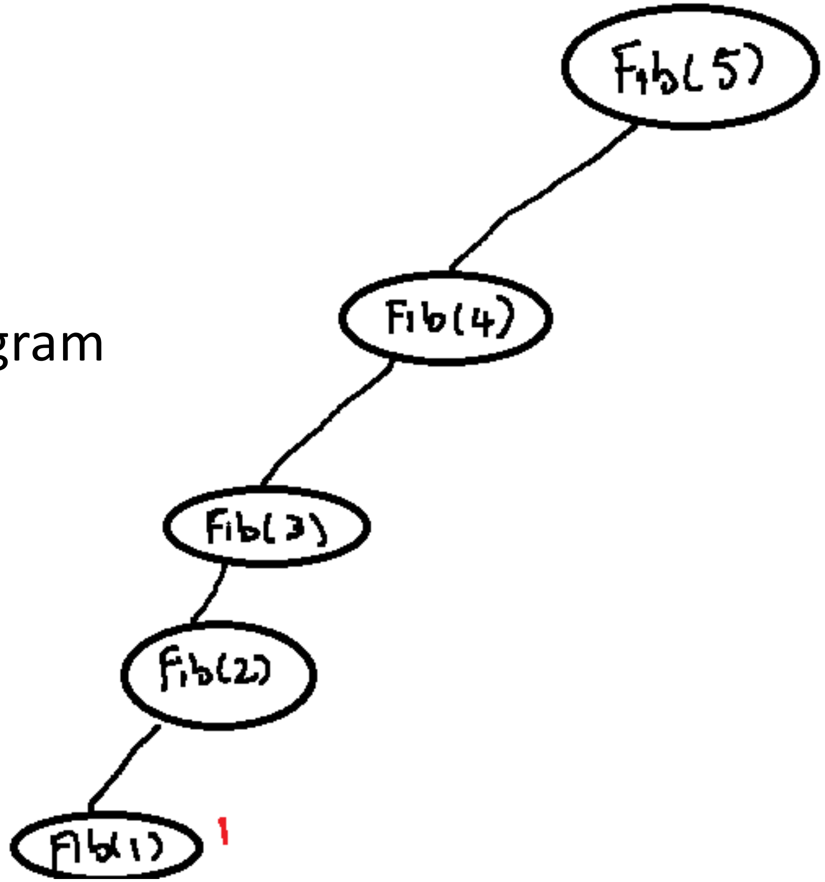
$= 2T(n-1) + c$ //assuming T(n-1) ~ T(n-2), for upper bound
$= 2*(2T(n-2) + c) + c = 4T(n-2) + 3c$
$= 8T(n-3) + 7c$
$= 2^k * T(n - k) + (2^k - 1)*c$

The value of k for which: n - k = 0 is k=n.
Hence,
$T(n) = 2^n * T(0) + (2^n - 1)*c$
$= 2^n * (1 + c) – c$
$\sim 2^n$

Similarly lower bound can be calculated as $\sim 2^{(n/2)}$

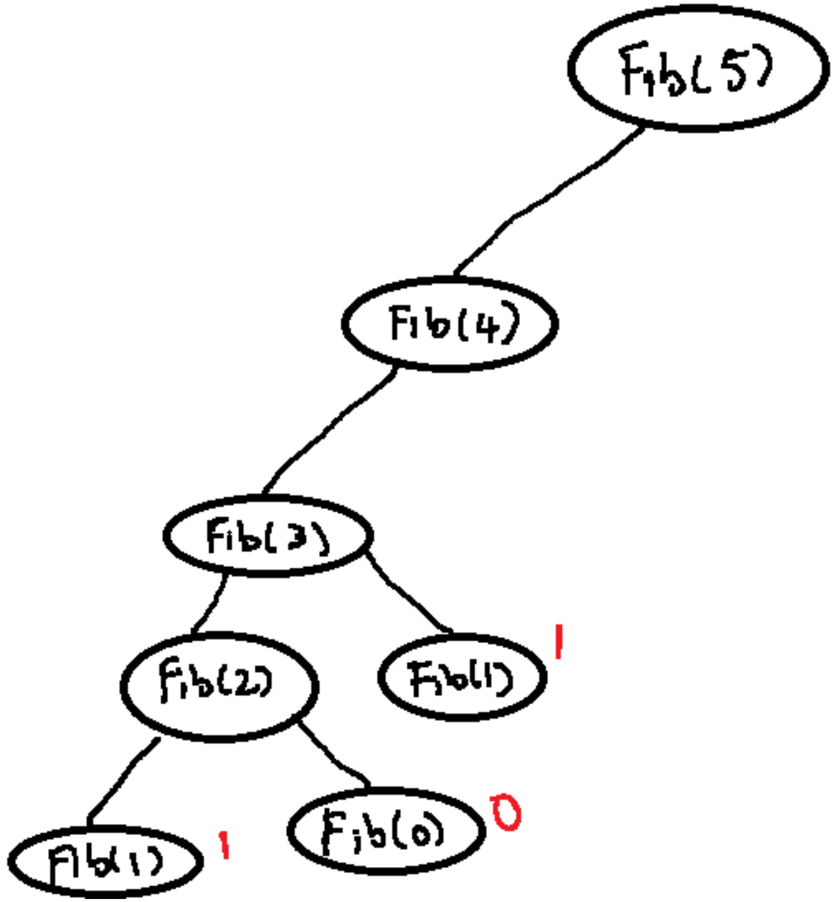
Recursion

- So, the program is not very efficient.
- Where is the program spending all this time?
- Consider the computations of the program as a tree of recursive calls when n = 5.



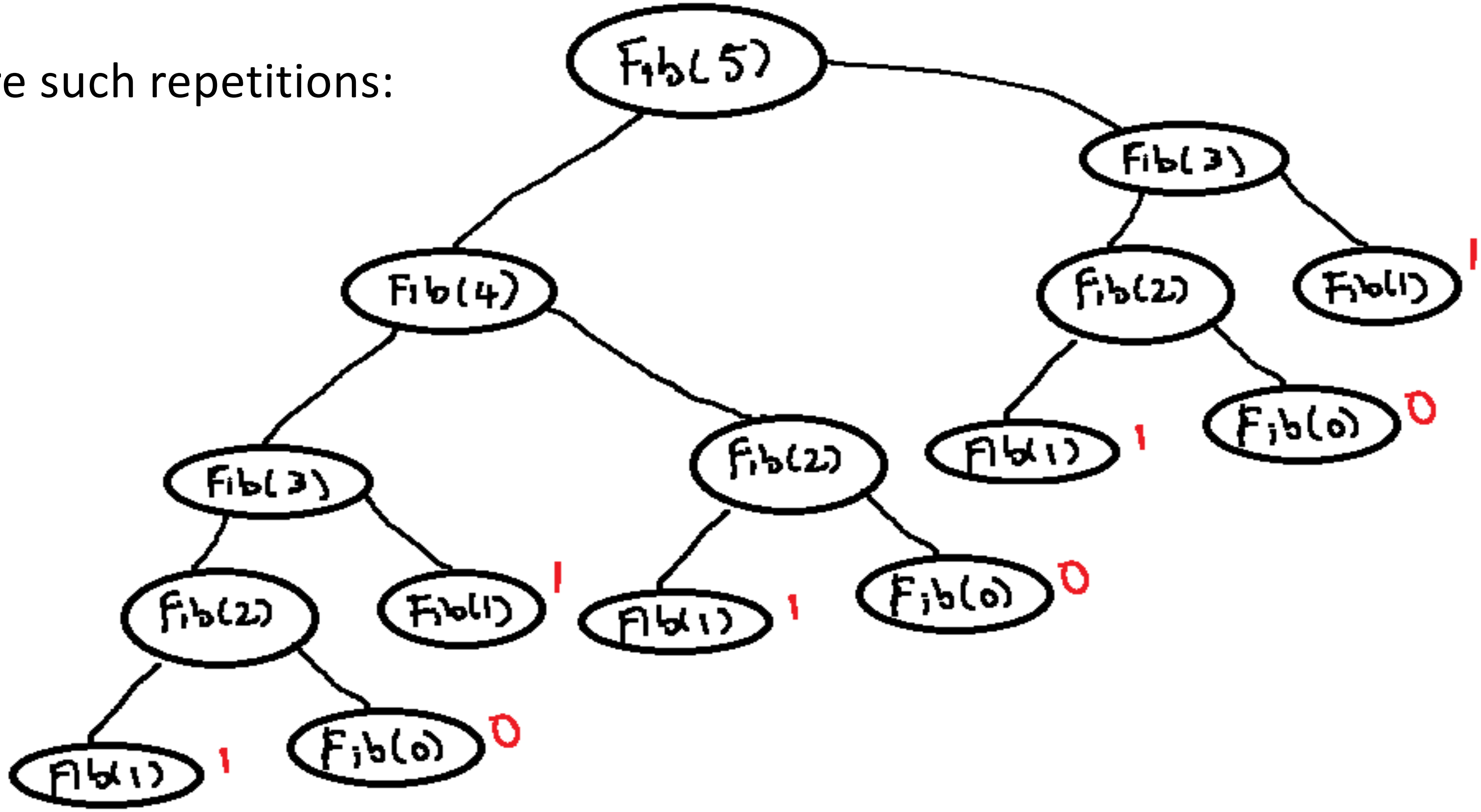
Recursion

- Continuing further, we notice that the call with n =1 is happening two times so far.



Recursion

- More such repetitions:



Recursion

- There is actually a much better way to write the program, still using recursion.
- Try it, and compute the runtime of the program.

Recursion

- Observations from the modified program.
  - Recursion alone is not enough.
  - Need to know how to optimize programs using recursion.
  - The particular technique needed in the simple example is called memoization.

Recursion

- Question: Are all recursive versions of a computation equivalent in their runtime?
  - Of course, when also written efficiently.
- In particular, reconsider the Fibonacci example again.
- There are several (recursive) methods to compute the nth Fibonacci number.

Recursion

- Let A = [0 1; 1 1] be a 2x2 matrix.
- It can be shown that $A^{n+1} = [F_{n-1}, F_n \; ; F_n \; F_{n+1}]$
- Which method will be faster ?

Recursion

- Recursive programs are slightly unfriendly to compilers.
- Some details follow.

Compilers and Recursion

- Let us recall how the compiler implements a function call.
- Let A and B be two functions and function A calls function B.
- To perform the call, the compiler has to introduce
  – A way to save the state of the function A at the line where the call to function B is placed.
  – A way to set the parameters for function B and jump to the first line of B
  – Capture the return value of function B
  – Pass the return value from B to function A,
  – And Restore the state of function A.

Compilers and Recursion

- In general, this process of saving the current state and creating a new execution state for each recursive call can introduce program overhead.
- In an extreme setting, consider a program such as:
- Finding the sum of the first n natural numbers.

```
int  AddN(int n)
{
if (n == 1) return 1;
return n + AddN(n – 1);
}
```

Compilers and Recursion

- Modern compliers are clever.
- They can notice this phenomenon where there are recursive calls at the last line of the program.
  - Called as tail recursion.
- With tail recursion, it can be noted that the state of the current call is nearly useless.
- So, a compiler can in fact automatically rework the program.

Other use of Recursion

- In problem solving and algorithm design, recursive solutions can be also quite useful and intuitive to design.
- Termed often as divide and conquer with applications to problems including
  - Sorting
  - Convex hull,
  - And many more.
- A variant of the divide and conquer is the partitioning technique.

Thank You