

# the substitution method

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

### 1 solving recurrences

number of calls for a plain recursive Fibonacci

### 2 solving recurrences

the substitution method

a boundary condition

when things are not straightforward

MCS 360 Lecture 38

Introduction to Data Structures

Dimitris Diochnos and Jan Verschelde, 19 November 2010

# the substitution method

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

1 solving recurrences  
number of calls for a plain recursive Fibonacci

2 solving recurrences  
the substitution method  
a boundary condition  
when things are not straightforward

## solving recurrences

solving  
recurrences

number of calls for a  
plain recursive  
Fibonacci

solving  
recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

Recall the straightforward recursive algorithm to compute the Fibonacci numbers  $f_n$  following

$$f_0 = 0, f_1 = 1, \text{ and } f_n = f_{n-1} + f_{n-2}, \text{ for } n > 1.$$

Denote by  $c_n = \# \text{calls to compute } f_n$ .

$$c_2 = 2, c_3 = 4 = 2^2, c_4 = 8 = 2^3$$

Following the recursion:  $c_n = c_{n-1} + c_{n-2} + 2 = \dots$ .

In Lecture 22 we then just claimed that  $c_n$  is  $O(2^n)$ .

# the substitution method

## solving recurrences

number of calls for a plain recursive Fibonacci

## solving recurrences

### the substitution method

a boundary condition when things are not straightforward

1 solving recurrences  
number of calls for a plain recursive Fibonacci

2 solving recurrences  
the substitution method  
a boundary condition  
when things are not straightforward

# the substitution method

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method

a boundary condition  
when things are not  
straightforward

The substitution method for solving recurrences consists of two steps:

- 1 Guess the form of the solution.
- 2 Use mathematical induction to find constants in the form and show that the solution works.

The inductive hypothesis is applied to smaller values, similar like recursive calls bring us closer to the base case.

The substitution method is powerful to establish lower or upper bounds on a recurrence.

## an example

The recurrence relation for the cost of a divide-and-conquer method is

$$T(n) = 2T(\lfloor n/2 \rfloor) + n.$$

Our induction hypothesis is  $T(n)$  is  $O(n \log_2(n))$   
or  $T(n) \leq cn \log_2(n)$  for some constant  $c$ , independent of  $n$ .

Assume the hypothesis holds for all  $m < n$  and substitute:

$$\begin{aligned} T(n) &\leq 2(c \lfloor n/2 \rfloor \log_2(\lfloor n/2 \rfloor)) + n \\ &\leq cn \log_2(n/2) + n \\ &= cn \log_2(n) - cn \log_2(2) + n \\ &= cn \log_2(n) - cn + n \\ &\leq cn \log_2(n) \end{aligned}$$

as long as  $c \geq 1$ .

## applied to recursive Fibonacci

### solving recurrences

number of calls for a plain recursive Fibonacci

### solving recurrences

the substitution method

a boundary condition when things are not straightforward

Denote by  $c_n = \# \text{calls}$  to compute the  $n$ -th Fibonacci number in a plain recursive manner.

The recurrence is  $c_n = c_{n-1} + c_{n-2} + 2$ .

Our induction hypothesis:  $c_n$  is  $O(2^n)$  or  $c_n \leq \gamma 2^n$  for some constant  $\gamma$ , independent of  $n$ .

Assuming the induction hypothesis holds for all  $m < n$ , we substitute:

$$\begin{aligned} c_n &\leq \gamma 2^{n-1} + \gamma 2^{n-2} + 2 \\ &= \gamma 2^{n-2}(2 + 1) + 2 \\ &\leq \gamma 2^{n-2}(2 + 2), \quad \text{for } n > 2, \gamma \geq 1 \\ &\leq \gamma 2^n \end{aligned}$$

so the upper bound on  $c_n$  holds.

# the substitution method

## solving recurrences

number of calls for a plain recursive Fibonacci

## solving recurrences

the substitution method

**a boundary condition**

when things are not straightforward

1 solving recurrences  
number of calls for a plain recursive Fibonacci

2 solving recurrences  
the substitution method  
**a boundary condition**  
when things are not straightforward

# a boundary condition

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method

a boundary condition

when things are not  
straightforward

We derived the upper bound for the recurrence for the recursive Fibonacci, requiring  $n > 2$ .

Note:

- Mostly we are interested only in asymptotic values for  $n$ , that is: for sufficiently large values of  $n$ .
- The boundary condition on  $n$  is a constant.

## revisiting our earlier example

### solving recurrences

number of calls for a  
plain recursive  
Fibonacci

### solving recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

In our earlier example we showed the *induction step* of

$$T(n) = 2T(\lfloor n/2 \rfloor) + n,$$

i.e.  $T(n) \leq cn \log_2(n)$  when  $c \geq 1$ .

- We should also show that the base case holds!

Assuming that  $T(1) = 1$ , we would like to show

$$T(1) \leq c \cdot 1 \cdot \log_2(1) = c \cdot 0 = 0,$$

which is impossible when  $T(1) > 0$ .

We only want to show that  $T(n) \leq cn \log_2(n)$  for *sufficiently large* values of  $n$ ; i.e.  $\forall n \geq n_0$ .

$\implies$  try  $n_0 > 1$ .

## the base case

solving  
recurrences

number of calls for a  
plain recursive  
Fibonacci

solving  
recurrences

the substitution  
method

a boundary condition

when things are not  
straightforward

$$T(n) = 2T(\lfloor n/2 \rfloor) + n.$$

We have:

$$T(1) = 1 \Rightarrow \begin{cases} T(2) = 4 \\ T(3) = 5 \end{cases}$$

We want to satisfy simultaneously

$$\begin{cases} 4 = T(2) \leq c \cdot 2 \cdot \log_2(2) \\ 5 = T(3) \leq c \cdot 3 \cdot \log_2(3) \end{cases}$$

$$\Rightarrow \begin{cases} c \geq 2 \\ c \geq \frac{5}{3 \log_2(3)} \approx 1.052 \end{cases} \Rightarrow c \geq 2.$$

- We have to check *both*  $T(2)$  and  $T(3)$  *simultaneously* because of the nature of the recursive equation.

## a lower bound - part 1 of 4

solving

recurrences

number of calls for a  
plain recursive  
Fibonacci

solving

recurrences

the substitution  
method

a boundary condition

when things are not  
straightforward

We want to show  $T(n) \geq cn \log_2(n)$ .

Assume that  $n$  is a power of 2. We have:

$$\begin{aligned}
 T(n) &\geq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n && \text{(Induction Hypothesis)} \\
 &= cn \lg(n/2) + n && (n \text{ is a power of } 2) \\
 &= cn \lg n - cn \lg 2 + n \\
 &= cn \lg n - (c - 1)n \\
 &\geq cn \lg n,
 \end{aligned}$$

as long as  $c \leq 1$ .

We also want to satisfy the boundary condition ( $T(2) = 4$ ).

$$T(2) \geq c \cdot 2 \cdot \lg 2 = 2 \cdot c$$

In other words, it is enough if  $c \leq 2$ . By the requirement  $c \leq 1$  for the induction step we choose  $c = 1$ .

## a lower bound - part 2 of 4

solving  
recurrences

number of calls for a  
plain recursive  
Fibonacci

solving  
recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

We will prove that  $T(n)$  is strictly increasing.

For the base case note that  $T(1) = 1 < 4 = T(2)$ .

Assuming that for all  $k \leq n$  it holds  $T(k) > T(k - 1)$ , we want to show that  $T(n + 1) > T(n)$ . We distinguish cases for  $n + 1$ .

$(n + 1)$  is odd

Say  $n + 1 = 2m + 1$ . Then, it holds

$$\begin{aligned}
 T(2m + 1) &= 2T(\lfloor (2m + 1)/2 \rfloor) + 2m + 1 && \text{(Definition)} \\
 &= 2T(m) + 2m + 1 \\
 &= T(2m) + 1 && \text{(Definition)} \\
 &> T(2m).
 \end{aligned}$$

## a lower bound - part 3 of 4

solving  
recurrencesnumber of calls for a  
plain recursive  
Fibonaccisolving  
recurrencesthe substitution  
method

## a boundary condition

when things are not  
straightforward $(n + 1)$  is evenSay  $n + 1 = 2m$ . Then, it holds

$$T(2m) = 2T(\lfloor(2m)/2\rfloor) + 2m \quad \text{(Definition)}$$

$$= 2T(m) + 2m$$

$$> 2T(m - 1) + 2m \quad \text{(Ind. Hyp.)}$$

$$= 2T(\lfloor(2m - 1)/2\rfloor) + (2m - 1) + 1$$

$$= T(2m - 1) + 1 \quad \text{(Definition)}$$

$$> T(2m - 1).$$

Note that the induction hypothesis is used only when  $(n + 1)$  is even!

## a lower bound - part 4 of 4

solving  
recurrencesnumber of calls for a  
plain recursive  
Fibonaccisolving  
recurrencesthe substitution  
method

a boundary condition

when things are not  
straightforward

Consider the binary expansion of  $n > 0$ ; i.e.  $n = \sum_{i=0}^{\infty} b_i 2^i$ .

Set

$$k = \max_i \left\{ b_i = 1 : n = \sum_{i=0}^{\infty} b_i 2^i \right\} .$$

Define the function

$$g(n) = \begin{cases} n \cdot \lg n & , \quad n \text{ is a power of } 2, \\ k \cdot 2^k & , \quad \text{otherwise; } k \text{ as defined above} \end{cases} . \quad (1)$$

In other words,  $g(n)$  has “jumps” on the values that it takes when  $n$  is a power of 2, and remains constant until the next power of 2. From the previous analysis it now follows that  $T(n) = \Omega(g(n))$ .

19 Nov 2010

$$g(n) \leq T(n) \leq 2n \log_2(n)$$

## solving recurrences

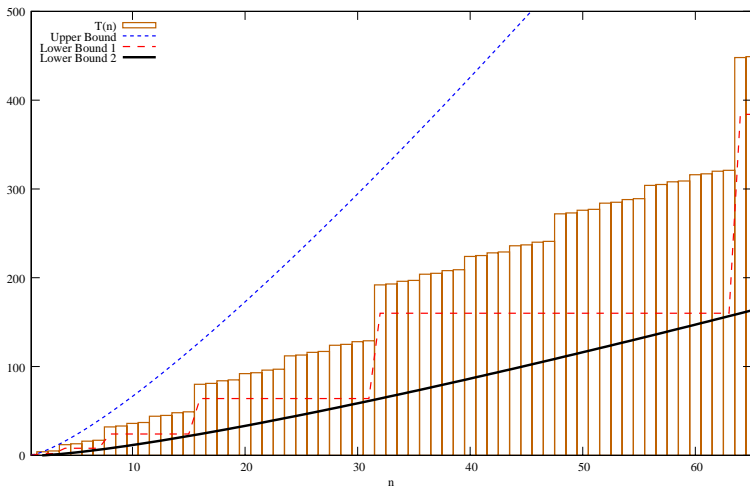
number of calls for a plain recursive Fibonacci

## solving recurrences

the substitution method

**a boundary condition**

when things are not straightforward



# the substitution method

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method  
a boundary condition  
**when things are not  
straightforward**

1 solving recurrences  
number of calls for a plain recursive Fibonacci

2 solving recurrences  
the substitution method  
a boundary condition  
**when things are not straightforward**

solving  
recurrencesnumber of calls for a  
plain recursive  
Fibonaccisolving  
recurrencesthe substitution  
method  
a boundary condition  
when things are not  
straightforward

Consider the recurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1.$$

Our guess is  $O(n)$ , so we try to show  $T(n) \leq cn$ .

$$\begin{aligned} T(n) &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

which does *not* imply that  $T(n) \leq cn$ , for any  $c$ .

- We need to show the *exact form*!

Ideas to overcome the hurdle:

- 1 Revise our guess; say  $T(n) = O(n^2)$ .
  - However, our original guess was correct!
- 2 Sometimes it is easier to prove something stronger!

## try a stronger bound

solving  
recurrencesnumber of calls for a  
plain recursive  
Fibonaccisolving  
recurrencesthe substitution  
method  
a boundary condition  
when things are not  
straightforward

We will attempt to show  $T(n) = cn - b$ , where  $b$  is another constant.

We have:

$$\begin{aligned} T(n) &\leq (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b \quad \text{for } b \geq 1 \end{aligned}$$

- We still have to specify  $c$ .

Assume that  $T(1) = 1$ . We want

$$T(1) = 1 \leq c \cdot 1 - b$$

Hence, it is enough to set  $c = 2$  and  $b = 1$ .

## changing variables

solving  
recurrences

number of calls for a  
plain recursive  
Fibonacci

solving  
recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

Consider the recurrence

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log_2(n) .$$

Rename  $m = \log_2(n)$ . We have:

$$T(2^m) = 2T(2^{m/2}) + m .$$

Define  $S(m) = T(2^m)$ . We get:

$$S(m) = 2S(m/2) + m .$$

Hence, the solution is  $O(m \log_2(m))$ , or with substitution

$$O(\log_2(n) \cdot \log_2(\log_2(n))) .$$

## want an exact solution?

solving  
recurrencesnumber of calls for a  
plain recursive  
Fibonaccisolving  
recurrencesthe substitution  
method  
a boundary conditionwhen things are not  
straightforward

Using Maple:

```
rsolve(c(n) = c(n-1) + c(n-2) + 2, c(n));
returns
```

$$\begin{aligned} & \left( \frac{1}{2}c(0) + \frac{1}{10}c(0) - \frac{1}{5}\sqrt{5}c(1) \right) \left( -\frac{1}{2}\sqrt{5} + \frac{1}{2} \right)^n \\ & + \left( -\frac{1}{10}c(0)\sqrt{5} + \frac{1}{2}c(0) + \frac{1}{5}c(1)\sqrt{5} \right) \left( \frac{1}{2}\sqrt{5} + \frac{1}{2} \right)^n \\ & + \frac{4}{5}\sqrt{5} \frac{\left( -\frac{2}{\sqrt{5}+1} \right)^n}{\sqrt{5}+1} - \frac{4}{5}\sqrt{5} \frac{\left( \frac{-2}{-\sqrt{5}+1} \right)^n}{-\sqrt{5}+1} - 2 \end{aligned}$$

# Summary + Assignments

## solving recurrences

number of calls for a  
plain recursive  
Fibonacci

## solving recurrences

the substitution  
method  
a boundary condition  
when things are not  
straightforward

We covered §4.3 of *Introduction to Algorithms*, 3rd edition by Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson.

## Assignments:

- 1 Apply the substitution method to show that the solution of  $T(n) = T(n - 1) + n$  is  $O(n^2)$ .
- 2 Consider the recurrence defined by  $T(n) = T(\lceil n/2 \rceil) + 1$ . Show that the solution to this recurrence is  $O(\log_2(n))$ .