

Discrete Mathematical Structures CS 2233 Lecture Twenty

Prof. William Winsborough
April 16, 2009

Business

- **Recall**
 - Read
 - 4.4, 7.3
 - Turn in homework 9 today
- **Homework 10 due Thursday 4/23**
 - 4.4: 8, 10, 22
 - Let $T(n)$ be the number of additions performed by the recursive algorithm for calculating fibonacci(n), as presented in this lecture. Prove that $T(n) = O(2^n)$ by using strong induction to prove that $T(n) \leq 2^n$, for all $n \in \mathbb{N}$
 - 7.3: 2 (prove your answer is correct by using strong induction)

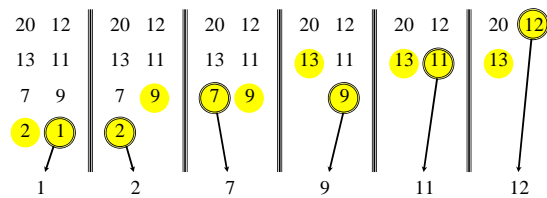
Merge sort

MERGE-SORT $A[1 \dots n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
3. “Merge” the 2 sorted lists.

Key subroutine: MERGE

Merging two sorted arrays



$\Theta(n)$ time to merge a total of n elements (linear time).

Analyzing merge sort

- | | |
|-------------|--|
| $T(n)$ | MERGE-SORT $A[1 \dots n]$
1. If $n = 1$, done.
2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
3. “Merge” the 2 sorted lists |
| $\Theta(1)$ | |
| $2T(n/2)$ | |

Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

Analyzing merge sort

1. **Divide:** Trivial.
2. **Conquer:** Recursively sort 2 subarrays.
3. **Combine:** Merge two sorted subarrays

$$T(n) = 2T(n/2) + D(n) + C(n)$$

subproblems subproblem size work dividing work combining

1. What is the time for the base case? **Constant**
2. What are $D(n)$ and $C(n)$?
3. What is the growth order of $T(n)$?

Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1; \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- **Note:** $\Theta(n)$ stands for an arbitrary *anonymous* function belonging to $\Theta(n)$
- The base case is often not mentioned when $T(n) = \Theta(1)$ for values of n that are bounded by a constant

- What function $T(n)$ satisfies this equation?
 - This actually depends on the anonymous $\Theta(n)$ function
- And what is its asymptotic complexity?

16 April 2009

Winsborough CS 2233 Lecture 20

7

The Substitution Method for Solving Recurrences

- Guess the form of the solution
- Use mathematical induction to find the constants and show that the solution works
- Example:
 $T(n) = 2T(\lfloor n/2 \rfloor) + n$
 Guess $T(n) = O(n \lg n)$
 Requires us to prove there exists a constant c such that for sufficiently large n , $T(n) \leq cn \lg n$

16 April 2009

Winsborough CS 2233 Lecture 20

8

Proof of Induction Step

- Induction assumption: $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$
- Proof of step:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n, \text{ by def. of } T(n) \\ &\leq 2(c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n, \text{ ind. assump.} \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \end{aligned}$$
 provided $c \geq 1$

16 April 2009

Winsborough CS 2233 Lecture 20

9

Proof of Induction Basis

- When $n = 1$, we don't actually have $T(n) \leq cn \lg n$ because $T(n) = 1$ and $\lg 1 = 0$
- But if we take $n_0 = 2$ and any constant $c \geq 2$, then we have for all $n \geq n_0$, $T(n) \leq cn \lg n$, as required
- The base case of our induction shows:
 $T(2) = 2T(1) + 2 \leq 2c \lg 2$ and
 $T(3) = 2T(1) + 3 \leq 3c \lg 3$
 - Note that the basis of our induction is different from the basis of the recurrence

16 April 2009

Winsborough CS 2233 Lecture 20

10

Is Divide and Conquer Always Best?

- Consider programs for the Fibonacci sequence:
 - $f_0 = 0, f_1 = 1$
 - $f_n = f_{n-1} + f_{n-2}$, for $n \geq 2$
- Iterative (*Dynamic Programming*)
 - fib(n)


```

                    if n = 0 return 0; if n = 1 return 1;
                    fib_last = 0; fib = 1;
                    for i = 2 to n do
                        temp = fib;
                        fib = fib + fib_last;
                        fib_last = temp;
                    return fib
                    
```
- Time complexity is clearly $O(n)$ since for each value between 2 and n , a constant amount of work is done

16 April 2009

Winsborough CS 2233 Lecture 20

11

Is Divide and Conquer Always Best?

- Consider programs for the Fibonacci sequence:
 - $f_0 = 0, f_1 = 1$
 - $f_n = f_{n-1} + f_{n-2}$, for $n \geq 2$
- Recursive (*Divide and Conquer*)
 - fib(n)


```

                    if n = 0 return 0; if n = 1 return 1;
                    return fib(n-1) + fib(n-2)
                    
```
- The cost of computing fib(n)
 - $T(n) = T(n-1) + T(n-2) + 1$
 - This has solution $T(n) = O(2^n)$
- Divide and conquer does not work well when the solutions to subproblems require solving further subproblems that they share
 - In cases like this, dynamic programming is often much faster

16 April 2009

Winsborough CS 2233 Lecture 20

12