_ In-Class Soln 1 _____

Let f(n) be an always positive function and let $g(n) = f(n) \log n$. Show that f(n) = o(g(n))

- For any positive constant c, we want to show there is a $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0$.
- \bullet In other words, we want to show that there is $n_0>0$ such that

$$0 \le f(n) < cf(n) \log n$$

Dividing by $f(n) \log n$, we get:

$$0 \le \frac{f(n)}{f(n)\log n} < c$$

• We know that $\lim_{n\to\infty}\frac{f(n)}{f(n)\log n} = \lim_{n\to\infty}\frac{1}{\log n} = 0$. Thus, for any constant c, there must be a n_0 such that the above inequality is satisfied for all $n \ge n_0$.

2

___ In-Class Soln 2 ____

Let f(n) be an always positive function and let $g(n) = f(n) \log n$. Show that f(n) = o(g(n))

- For any positive constant c, we want to show there is a n₀ > 0 such that 0 ≤ f(n) < cg(n) for all n ≥ n₀.
- \bullet In other words, we want to show that there is $n_0>0$ such that

$$0 \le f(n) < cf(n) \log n$$

Dividing by $f(n) \log n$, we get:

$$\begin{array}{rcl} \displaystyle \frac{1}{\log n} & < & c \\ \displaystyle (1/c) & < & \log n \\ \displaystyle 2^{1/c} & < & n \end{array}$$

• Thus, for any c, if we choose $n_0 > 2^{1/c}$, for all $n \ge n_0$, f(n) < cg(n)

CS 361, Lecture 4

Jared Saia University of New Mexico

Today's Outline _____

• Recurrence Relations

_ Alg: Binary Search ____

What?

• In studying behavior of algorithms, we'll be more concerned with *rate* of growth than with constants

- $O, \Theta, \Omega, o, \omega$ give us a way to talk about rates of growth
- Asymptotic analysis is an extremely useful way to compare run times of algorithms
- However, empirical analysis is also important (you'll be studying this in your project)

bool BinarySearch (int arr[], int s, int e, int key){
 if (e-s<=0) return false;
 int mid = (e-s)/2;
 if (arr[key]==arr[mid]){
 return true;
 }else if (key < arr[mid]){
 return BinarySearch (arr,s,mid,key);}
 else{
 return BinarySearch (arr,mid,e,key)}
}</pre>

4

Recurrence Relations _____

• Getting the run times of recursive algorithms can be challenging

- Recall our algorithm for binary search
- Let T(n) be the run time of this algorithm on an array of size n
- Then we can write T(1) = 1, T(n) = T(n/2) + 1

- T(n) is a function giving the run time of Binary Search on an array of size n
- T(n) = T(n/2) + 1 is a an example of a *recurrence* relation
- A Recurrence Relation is any equation for a function T(n), where T(n) appears on both the left and right sides of the equation.
- We always want to "solve" these recurrence relation by getting an equation for T(n), where T appears on just the left side of the equation

____ Alg1 ____

- We can use recurrence relations to analyze many properties of recursive algorithms e.g. run time, value returned, etc.
- To do this we need to: 1) write down the correct recurrence relation 2) solve the recurrence relation
- Step 1 is usually easier than step 2

```
Alg1 (int n){
    if (n<=1) return 1;
    else
        return Alg1(n/2) + Alg1(n/2) + n;
}</pre>
```



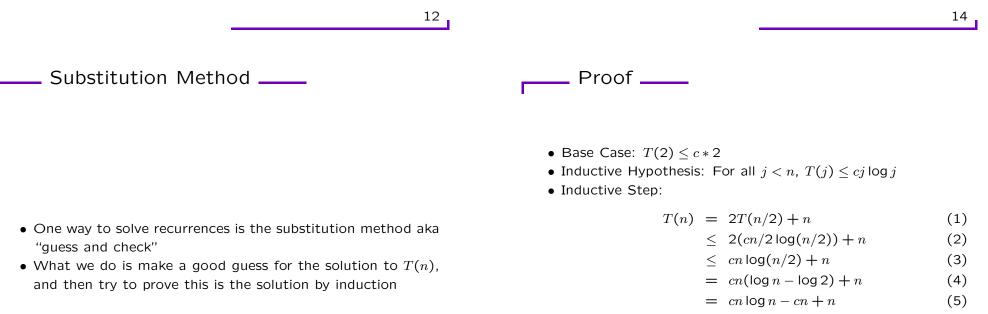
- The running time of an algorithm on a constant size input is always $\Theta(1)$
- Thus for convenience, we frequently omit statements of the boundary conditions and just assume T(n) is constant when n is a constant.
- Example: Instead of saying "If n = 1, $T(n) = \theta(1)$, and if $T(n) = 2 * T(n/2) + \Theta(n)$ ", we just say " $T(n) = 2 * T(n/2) + \Theta(n)$ "

- Let T(n) be the run time of Alg1 on input n
- Then we can write T(n) = 2T(n/2) + 1
- Let f(n) be the value returned by Alg1 on input n
- Then we can write f(n) = 2f(n/2) + n and f(1) = 1

___ Example ____

- To get the "real" run time or value returned, we need to *solve* the recurrence relation
- This means that no function appear on the right hand side
- We will review several techniques for solving recurrences including: the substitution method, recursion trees, the Master method, and annihilators

- Let's guess that the solution to T(n) = 2 * T(n/2) + n is $T(n) = O(n \log n)$
- We want to show that $T(n) \leq cn \log n$ for appropriate choice of constant c
- We can prove this by induction.



 $\leq cn \log n$ (6)

The last step holds if $c \ge 1$

16

Recurrences and Induction _____

Recurrences and Induction are closely related:

Some Examples _____

- To find some solution to f(n), solve a recurrence
- To prove that a solution for f(n) is correct, use induction

For both recurrences and induction, we always solve a big problem by reducing it to smaller problems! • f(n) is the sum of the integers $1, \ldots, n$

___ Sum Problem _____

- f(n) is the maximum number of leaf nodes in a binary tree of height n
- The next several problems can be attacked by induction/recurrences
- For each problem, we'll need to reduce it to smaller problems
- Question: How can we reduce each problem to a smaller subproblem?

- Recall:
 - In a binary tree, each node has at most two children
 - A *leaf* node is a node with no children

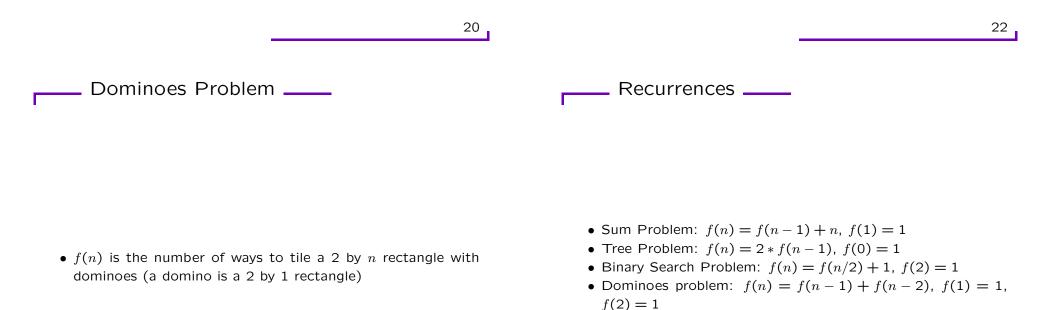
Tree Problem

• The height of a tree is the length of the longest path from the root to a leaf node.

Binary Search Problem _____

• f(n) is the maximum number of queries that need to be made for binary search on a sorted array of size n.

- Sum Problem: What is the sum of all numbers between 1 and n-1 (i.e. f(n-1))?
- Tree Problem: What is the maximum number of leaf nodes in a binary tree of height n 1? (i.e. f(n 1))
- Binary Search Problem: What is the maximum number of queries that need to be made for binary search on a sorted array of size n/2? (i.e. f(n/2))
- Dominoes problem: What is the number of ways to tile a 2 by n-1 rectangle with dominoes? What is the number of ways to tile a 2 by n-2 rectangle with dominoes? (i.e. f(n-1), f(n-2))



____ Sum Problem _____

- Sum Problem: f(n) = (n + 1)n/2
- Tree Problem: $f(n) = 2^n$
- Binary Search Problem: $f(n) = \log n$
- Dominoes problem: $f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n$

- Want to show that f(n) = (n+1)n/2.
- Prove by induction on n
- Base case: f(1) = 2 * 1/2 = 1
- Inductive hypothesis: for all j < n, f(j) = (j+1)j/2
- Inductive step:

$$f(n) = f(n-1) + n$$
 (7)

$$= n(n-1)/2 + n$$
 (8)

$$= (n+1)n/2$$
 (9)



"Trying is the first step to failure" - Homer Simpson

- Now that we've made these guesses, we can try using induction to prove they're correct (the substitution method)
- We'll give inductive proofs that these guesses are correct for the first three problems

- Want to show that $f(n) = 2^n$.
- Prove by induction on n
- Base case: $f(0) = 2^0 = 1$
- Inductive hypothesis: for all j < n, $f(j) = 2^j$
- Inductive step:

$$f(n) = 2 * f(n-1)$$
 (10)

$$= 2 * (2^{n-1}) \tag{11}$$

 $= 2^{n}$ (12)

Binary Search Problem _____

____ Todo ____

- Want to show that $f(n) = \log n$. (assume *n* is a power of 2)
- \bullet Prove by induction on \boldsymbol{n}
- Base case: $f(2) = \log 2 = 1$
- Inductive hypothesis: for all j < n, $f(j) = \log j$
- Inductive step:

$$f(n) = f(n/2) + 1$$
 (13)

$$= \log n/2 + 1$$
 (14)

$$= \log n - \log 2 + 1 \tag{15}$$

$$= \log n$$
 (16)

• Read Chapter 4 (Recurrences) in text

28

29

____ In Class Exercise _____

- Consider the recurrence f(n) = 2f(n/2) + 1, f(1) = 1
- Guess that $f(n) \leq cn 1$:
- Q1: Show the base case for what values of c does it hold?
- Q2: What is the inductive hypothesis?
- Q3: Show the inductive step.