Today’s Outline

- Recurrence Relations

---

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
  \( 0 \leq f(n) < cg(n) \) for all \( n \geq n_0 \).
- In other words, we want to show that there is \( n_0 > 0 \) such that
  \[ 0 \leq f(n) < cf(n) \log n \]
Dividing by \( f(n) \log n \), we get:

\[ 0 \leq \frac{f(n)}{f(n) \log n} < c \]

- 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 \( n \geq n_0 \).

---

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 \( n_0 > 0 \) such that
  \( 0 \leq f(n) < cg(n) \) for all \( n \geq n_0 \).
- In other words, we want to show that there is \( n_0 > 0 \) such that
  \[ 0 \leq f(n) < cf(n) \log n \]
Dividing by \( f(n) \log n \), we get:

\[ \frac{1}{\log n} < c \]
\[ \frac{1}{c} < \log n \]
\[ 2^{1/c} < n \]

- Thus, for any \( c \), if we choose \( n_0 > 2^{1/c} \), for all \( n \geq n_0 \),
  \( f(n) < cg(n) \)
Asymptotic Analysis - Take Away

- 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)

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 $T(1) = 1$, $T(n) = T(n/2) + 1$

Alg: Binary Search

```java
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)}
}
```

What?

- $T(n)$ is a function giving the run time of Binary Search on an array of size $n$
- $T(n) = T(n/2) + 1$ is 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
Use of Recurrences

- 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

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

Example1

- 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 $f(1) = 1$

A Side Note

- 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 $T(n) = 2 \times T(n/2) + \Theta(n)$”, we just say “$T(n) = 2 \times T(n/2) + \Theta(n)$”
What now?

- 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.

Example

- Let’s guess that the solution to $T(n) = 2 \cdot T(n/2) + n$ 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.

Substitution Method

- One way to solve recurrences is the substitution method aka “guess and check”.
- What we do is make a good guess for the solution to $T(n)$, and then try to prove this is the solution by induction.

Proof

- Base Case: $T(2) \leq c \cdot 2$
- Inductive Hypothesis: For all $j < n$, $T(j) \leq cj \log j$.
- Inductive Step:

$$
T(n) = 2T(n/2) + n
\leq 2(cn/2 \log(n/2)) + n
\leq cn \log(n/2) + n
= cn(\log n - \log 2) + n
= cn \log n - cn + n
\leq cn \log n
$$

The last step holds if $c \geq 1$. 
Recurrences and Induction are closely related:

- 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!

Some Examples

- 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?

Sum Problem

- $f(n)$ is the sum of the integers $1, \ldots, n$

Tree Problem

- $f(n)$ is the maximum number of leaf nodes in a binary tree of height $n$

Recall:

- In a binary tree, each node has at most two children
- A leaf node is a node with no children
- 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 $n$.

Dominoes Problem

- $f(n)$ is the number of ways to tile a 2 by $n$ rectangle with dominoes (a domino is a 2 by 1 rectangle)

Simpler Subproblems

- 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. $f(n-1), f(n-2)$)

Recurrences

- Sum Problem: $f(n) = f(n-1) + n$, $f(1) = 1$
- Tree Problem: $f(n) = 2 * f(n-1)$, $f(0) = 1$
- Binary Search Problem: $f(n) = f(n/2) + 1$, $f(2) = 1$
- Dominoes problem: $f(n) = f(n-1) + f(n-2)$, $f(1) = 1$, $f(2) = 1$
Guesses

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

Inductive Proofs

“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

Sum Problem

- 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:

\[
\begin{align*}
    f(n) &= f(n-1) + n \\
    &= n(n-1)/2 + n \\
    &= (n + 1)n/2
\end{align*}
\]

Tree Problem

- 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:

\[
\begin{align*}
    f(n) &= 2 * f(n-1) \\
    &= 2 * (2^{n-1}) \\
    &= 2^n
\end{align*}
\]
Binary Search Problem

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

\[
\begin{align*}
  f(n) &= f(n/2) + 1 \quad (13) \\
   &= \log n/2 + 1 \quad (14) \\
   &= \log n - \log 2 + 1 \quad (15) \\
   &= \log n \quad (16)
\end{align*}
\]

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.

Todo

- Read Chapter 4 (Recurrences) in text