CS472 Module 2 Part C - Math for Analysis of Algorithms, Part 3: Examples

Athens State University

Outline

Contents

1 Working with Big-Oh

Rate of Increase

For each of the following functions, indicate how much the function's value will change if its argument is increased fourfold:

1. $\log_2(n)$
2. $\sqrt(n)$
3. n

1. n^2
2. n^3
3. 2^n

Answer:

1. $\log_2(4n) - \log_2(n) - \log_2(n) = 2$
2. $\dfrac{\sqrt(4n)}{\sqrt(n)} = 4$
3. $\dfrac{4n}{\sqrt(n)} = 4$

1. $\dfrac{4n^2}{\sqrt(n^2)} = 4^2$

2. $\frac{4n^3}{\sqrt(n^3)} = 4^3$

3. $\frac{2^{4n}}{\sqrt(2^n)} = 2^{3*n} = (2^n)^3$

Rate of increase

Indicate whether the first function of each of the following pairs of functions has a smaller, same, or larger order of growth(to within a constant multiple) than the other:

1. $n(n+1)$ and $2000 * n^2$
2. $\log_2(n)$ and $\ln(n)$
3. 2^{n-1} and 2^n

1. $100 * n^2$ and $0.01 * n^3$
2. $\log_2^2(n)$ and $\log_2(n^2)$
3. $(n-1)!$ and $n!$

Rate of increase

Answer:

1. $n(n+1) = n^2 + n$
2. Recall $\log_a(n) = \log_a(b) * \log_b(n)$
3. $2^{n-1} = \frac{1}{2} * 2^n$

1. Quadratic has lower order of growth than cubic
2. $$\log_2^2(n) = \log_2(n) * \log_2(n)$$ $$\log_2(n^2) = 2\log_2(n))$$
3. $n! = n * (n-1)!$

2 Algorithm Complexity

Looping

For the following algorithm, what is the output when $n = 2$, $n = 4$ and $n = 16$? and what is the time complexity of algorithm assuming that n is divisible by 2?

Algorithm 1: Nested loops

```
for i ← 2 ... n do
    for j ← 0 ... n do
        Output i and j;
        j ← j + floor (n/4);
```

n = 4: 20122232430313233340414243444
Assume that n is divisible by 4:

$$T_n = (n-1)(1+n/4) \in O(n^2) \tag{1}$$

Another looping example

A student in CS317 includes this function as a submission to an assignment in that course. What is the time complexity of this code and suggest how the student can improve the efficiency of their code?

```
int add_them(int n, int A[])
{
int i,j,k; j = 0;
for (i = 1; i <= n ; i++)
{
 j = j + A[i];
}
k = 1;
for (i = 1; i <= n; i++)
{
 k = k + k; }
return j+k;
}
```

Listing 1: A Noob's Code

Another looping example

A student in CS317 includes this function as a submission to an assignment in that course. What is the time complexity of this code and suggest how the student can improve the efficiency of their code?

- Both loops run from 1 to n, so the algorithm is $O(n)$.
- Here's where you apply some of the knowledge of sums you're taught in discrete math. This just computing 2^n, so why not do that directly

Design and Analysis

Design and analyze the time complexity of an algorithm to solve the following problem: Given a list of n distinct positive integers, partition the list into two sublists, each of size $n/2$, such that the difference between the sums of the integers in the two sublists is minimized.

Design and Analysis

Algorithm 2: Min List Partition

Input: A list of nodes A
Output: Two lists A and $A - S$
mindiff ← some large number;
for *each subset* S *of size* $n/2 of A$ **do**
  diff ← `abs` (`sum` (s)-`sum` (A-S));
  if *diff* $<$ *mindiff* **then**
    mindiff ← diff;
    TempSet ← S;
Output mindff and TempSet;

Design and Analysis

- Each iteration of the /for/ loop has a constant worst-case cost (one compare, an int assignment, and a set assignment)
- The loop executes $\binom{n}{(n/2)}$ times
  - More discrete math: This is $O\left(\frac{n!}{((n/2)!)^2)}\right)$

3 Induction, Recursion, and Recurrence Relations

Principle of Induction

- The *Principle of Induction* says that, for any logical predicate P, one may specify some basis $P(0)$ and an inductive step s.t. $P(k) \implies P(k+1)$ implies that, for any natural number $n \in N$, $P(n)$ is true

A More Pragmatic View of Induction Proofs

- One starts with an *induction base* that says the predicate is true for some initial value taken from the natural numbers.
- You then state the *induction hypothesis* that assumes the predicate is true for an arbitrary number greater than or equal to the initial value.
- And then you prove that if the predicate is true for some value n, then it will be true for $n+1$

Example 1. Prove that for all positive integers n, that

$$1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Proof. Induction base: For $n = 1$,

$$1 = \frac{1(1+1)}{2}$$

.

Induction hypothesis: Assume, for an arbitrary positive integer n, that

$$1 + 2 + \ldots + n = \frac{(n(n+1)}{2}$$

.

Induction step: We need to show that

$$1 + 2 + \ldots + (n+1) = \frac{(n+1)[(n+1)+1]}{2}$$

.

$$
\begin{aligned}
1+2+\ldots+(n+1) &= 1+2+\ldots+n+(n+1) \\
&= \frac{n(n+1)}{2}+n+1 \\
&= \frac{n(n+1)+2(n+1)}{2} \\
&= \frac{(n+1)(n+2)}{2} \\
&= \frac{(n+1)[(n+1)+1]}{2}
\end{aligned}
$$

$\square$

Recurrence Relations

- A *recurrence relation* is a relation that expresses a value at n in terms of smaller values of n.
- A recurrence relation **must** provide an *initial condition* that defines the starting point of the relation.
- The *closed form* (or *solution*) of a recurrence relation is an explicit expression for the values of the relation.

Recurrence Relations and Induction

Find a closed form expression for the recurrence relation, assuming that n is a power of 2:

$$
\begin{aligned}
&t_n = 7 * t_{n/2}, n > 1 \\
&t_1 = 1
\end{aligned}
$$

Recurrence relations and Induction

The first few values are:

$$
\begin{aligned}
t_2 &= 7 * t_{2/2} = 7 * t_1 = 7 \\
t_4 &= 7 * t_{4/2} = 7 * t_2 = 7^2 \\
t_8 &= 7 * t_{8/2} = 7 * t_1 = 7^3 \\
t_{16} &= 7 * t_{16/2} = 7 * t_8 = 7^4
\end{aligned}
$$

We can guess that $t_n = 7^{\log(n)}$.
Assume:$t_n = 7^{\log(n)}$.
Induction: $t_{2n} = 7^{\log(2n)}$.
Now insert $2n$ into the occurrence, we get (note where we use induction):

$$
\begin{aligned}
t_{2n} &= 7 * t_{(2n)/2} = 7 * t_n \\
&= 7 * 7^{\log(n)} \\
&= 7^{1+\log(1+\log(n)} \\
&= 7^{\log(2)+\log(n)} \\
&= 7^{\log(2n)}
\end{aligned}
$$

Note: $7^{\log(n)} = n^{\log(7)} \approx n^{2.81}$

What is the closed form expression for the recurrence relation:

$$t_n - 5 * t_{n-1} + 6_{n-2} = 0, n > 1$$
$$t_0 = 0$$
$$t_1 = 1$$

Note this is a homogeneous linear recurrence, so we can make a transformation and get the general form using some algebra.

Let's set $t_n = r^n$.

Result: $t_n - 5 * t_{n-1} + 6_{n-2} = r^n - 5 * r^{n-1} + 6 * r^{n-2}$

This equation in r^n is the **characteristic equation** of the recurrence.

For $t_n = r^n$ is solution to the recurrence, we know from discrete math that r must be a root of

$$r^n - 5 * r^{n-1} + 6 * r^{n-2} = 0 \tag{2}$$

$$r^n - 5 * r^{n-1} + 6 * r^{n-2} =$$
$$r^{n-2}(r^2 - 5r + 6)$$

Thus, the roots are 0 and the roots of $r^2 - 5r + 6 = 0$

$$r^2 - 5r + 6 = 0$$
$$(r - 3)(r - 2) = 0$$

So, we now know the roots of the char. equation are 0, 2, 3. Substituting back into the orig. assumption, the closed forms of equation are $t_n = 0$, $t_n = 3^n$, and $t_n = 2^n$

More on characteristic equations

Definition 2. The **characteristic equation** for the linear homogeneous linear equation with constant coefficients

$$a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0 \tag{3}$$

is defined as

$$a_0 r^k + a_1 r^{k-1} + \ldots + a_k r^0 = 0 \tag{4}$$

The Fibonacci Sequence, again

The Fibanocci sequence $< 1, 1, 2, 3, 5, 8, 13, \ldots >$ is generated by the recurrence relation:

$$f(0) = 1$$
$$f(1) = 1$$
$$f(n) = f(n-1) + f(n-2)$$

The Fibonacci Sequence, again

Suppose we use the recurrence relation for the Fibonacci sequence to build an algorithm that generates the sequence

Algorithm 3: Recursive Algorithm for generating the Fibonacci sequence

Input: An integer n
Output: The Fib-sequence up to n
if *n is 0 or 1* **then**
  return 1;
else
  return `FibRec` (n-1) + `FibRec` (n-2);

The characteristic equation for the Fib. recurrence equation:

$$t_n - t_{n-1} - t_{n-2} = 0 \tag{5}$$

or

$$r^2 - r - 1 = 0 \tag{6}$$

The roots of this equation are:

$$\begin{aligned} r_1 &= \frac{1+\sqrt(5)}{2} \\ r_2 &= \frac{1-\sqrt(5)}{2} \end{aligned} \tag{7}$$

So, for the Fib. sequence, we can express the closed form as

$$t_n = c_1\left(\frac{1+\sqrt(5)}{2}\right)^n + c_2\left(\frac{1-\sqrt(5)}{2}\right)^n \tag{8}$$

So, for the Fib. sequence, use the initial conditions to find values of c_0 and c_1

$$\begin{aligned} t_0 &= c_1\left(\frac{1+\sqrt(5)}{2}\right)^0 + c_2\left(\frac{1-\sqrt(5)}{2}\right)^0 = 0 \\ t_1 &= c_1\left(\frac{1+\sqrt(5)}{2}\right)^1 + c_2\left(\frac{1-\sqrt(5)}{2}\right)^1 = 1 \end{aligned} \tag{9}$$

Solving this system of linear equations yields $c_1 = \frac{1}{\sqrt(5)}$ and $c_2 = \frac{-1}{\sqrt(5)}$