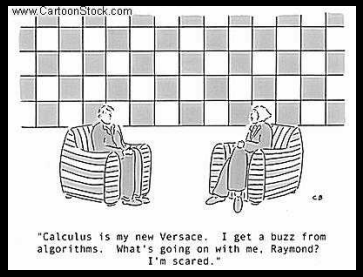


CS202 Fall 2012 Lecture 11 – 10/4

Runtime Analysis of Algorithms

Prof. Tanya Berger-Wolf
http://www.cs.uic.edu/bin/view/CS202/WebHome

Two example algorithms.

Suppose we wish to find the maximum number in a sequence of n numbers.

How long should we spend doing this?

Suppose it takes 1 time unit to make a comparison between two numbers.

In this case our algorithm takes about n time units.

We really have no clue how long the algorithm takes to run, but we have an inkling that it depends linearly on n.

We say the algorithm has "order n" running time.

Two example algorithms.

I have a number between 0 and 63. You ask a question, I'll tell you yes or no.

How long will it take you to find my secret number?

Suppose it takes 1 time unit to answer a query about my number.

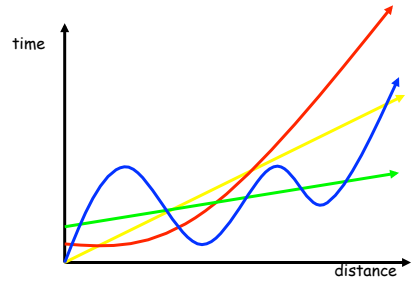
In this case the algorithm takes about lg n time units.

We really have no clue how long the algorithm takes to run, but we have an inkling that it depends logarithmically on n.

We say the algorithm has "order log n" running time.

Who wins the race?

The following graph gives times for completing races of length x, for 4 different competitors.



At each distance, who wins?

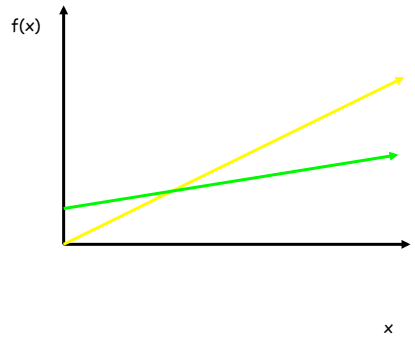
Who is the tortoise?

Who is the hare?

How would you describe blue's performance?

Quiz time…

Describe these functions:

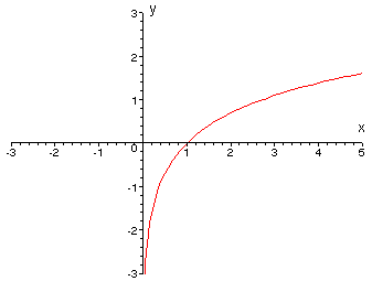


Linear functions

Every unit increase in x results in the same increase in f(x).

Quiz time…

Describe this function:

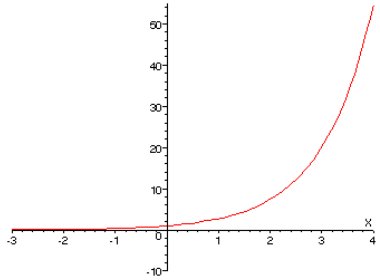


Logarithmic function

Very slow growing.

Quiz time…

Describe this function:

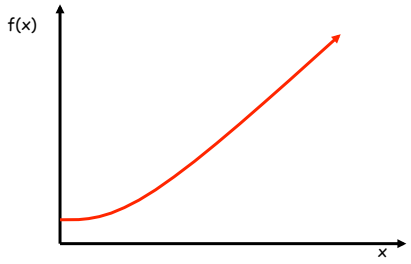


Exponential function

Very fast growing.

Quiz time…

Describe this function:



Quadratic function

Polynomial

Growth of functions

Algorithm analysis is concerned with:

- *Type* of function that describes run time (we ignore constant factors since different machines have different speed/cycle)
- Large values of x

Growth of functions (review?)

Important definition:

For functions f and g we write

$f(x) = O(g(x))$

to denote

$\exists\, c,k$ so that $\forall\, x>k$, $f(x) \le c \cdot g(x)$

We say "f(x) is big O of g(x)"

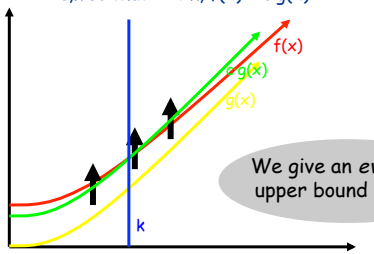
Recipe for proving $f(x) = O(g(x))$: find a **c** and **k** so that the inequality holds.

Growth of functions

iff

$f(x) = O(g(x))$

$\exists\, c,k$ so that $\forall\, x>k$, $f(x) \le c \cdot g(x)$



We give an *eventual* upper bound on f(x)

Growth of functions (examples)

iff

$f(x) = O(g(x))$

$\exists\, c,k$ so that $\forall\, x>k$, $f(x) \le c \cdot g(x)$

$3n = O(15n)$ since $\forall\, n>0$ $3n \le 1 \cdot 15n$

There's k

There's c

Growth of functions (examples)

iff

$f(x) = O(g(x))$

$\exists\ c,k$ so that $\forall\ x>k$, $f(x) \le c \cdot g(x)$

$15n = O(3n)$ since $\forall\ n \ge$ 0 $15n \le$ 5 $\cdot 3n$

Growth of functions (examples)

iff

$f(x) = O(g(x))$

$\exists\ c,k$ so that $\forall\ x>k$, $f(x) \le c \cdot g(x)$

$x^2 = O(x^3)$ since $\forall\ x \ge$ 1 $x^2 \le x^3$

Growth of functions (examples)

iff

$f(x) = O(g(x))$

$\exists\ c,k$ so that $\forall\ x>k$, $f(x) \le c \cdot g(x)$

$1000x^2 = O(x^2)$ since $\forall\ x \ge$ 0 $1000x^2 \le$ 1000 $\cdot x^2$

Growth of functions (examples)

iff

$f(x) = O(g(x))$

$\exists\ c,k$ so that $\forall\ x>k$, $f(x) \le c \cdot g(x)$

Prove that $x^2 + 100x + 100 = O((1/100)x^2)$

$x^2 + 100x + 100 \le 201x^2$ when $x > 1$

$\le 20100 \cdot (1/100)x^2$

$100x \le 100x^2$

$100 \le 100x^2$

k = 1,
c = 20100

Growth of functions (examples)

Prove that $5x + 100 = O(x/2)$

Similar problem, different technique.

Need $\forall x >$ ___, $5x + 100 \le$ ___ $\cdot x/2$

Try c = 10

$\forall x >$ ___, $5x + 100 \le 10 \cdot x/2$

Nothing works for k

Try c = 11

$\forall x >$ ___, $5x + 100 \le 5x + x/2$

$\forall x >$ 200 , $100 \le x/2$

k = 200, c = 11

Growth of functions

Guidelines:

In general, only the largest term in a sum matters.

$a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_nx^0 = O(x^n)$

n dominates lg n.

$n^5 \lg n = O(n^6)$

List of common functions in increasing O() order:

$1 \quad n \quad (n \lg n) \quad n^2 \quad n^3 \quad \ldots \quad 2^n \quad n!$

Growth of functions (more examples)

$$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 \le n \cdot n \cdot \ldots \cdot n \cdot n \cdot n = n^n$$

$n! = O(n^n)$ So

$$\log n! \le \log n^n = n \log n, \qquad \log n! = O(n \log n)$$

Questions?

Back to Runtime Analysis

```
Public static int sumCubes( int n )
{
        int partialSum =0;

        for( int i=1; i <= n; i++ )
        {
                partialSum += i * i * i;
        }
        return partialSum;
}
```

compute sum of cubes

Declarations with initialization and return counts for constant time.
The red line is executed once for each i between 1 and n. So n times.
Total running time: linear in n.

20

General Rules

- ***for* loops**: (running time of statements inside) * (number of iterations)
- **Nested loops:** analyze inside out, runtime of inside statement * product of loop sizes

```
for  (i = 0; i < n; i++ )
        for ( j = 0; j < m; j++ )
                k++;
```

- **Consecutive statements**: add up
- ***if/else*:** runtime of the test + runtime of max of the inside statements

```
if ( conditions )
        statement1;
else
        statement2;
```

21

Remember This?

```
Public static int findMax(int [] arr)
{
        int max = arr[0];
        for( int i=1; i<arr.length; i++ )
        {
                if ( arr[i] > max ) max = arr[i];
        }
        return max;
}
```

How many times is the statement max = arr[i] executed?

Best case: never

What about worst case? Once for every comparison – O(n)

On average? In expectation if all inputs of $x_1..x_n$ are equally likely?

(How many possible inputs are there of $x_1..x_n$? n!)

Remember This?

```
Public static int findMax(int [] arr)
{
        int max = arr[0];
        for( int i=1; i<arr.length; i++ )
        {
                if ( arr[i] > max ) max = arr[i];
        }
        return max;
}
```

Example: n = 3

Input ordering	$x_1<x_2<x_3$	$x_1<x_3<x_2$	$x_2<x_1<x_3$	$x_2<x_3<x_1$	$x_3<x_1<x_2$	$x_3<x_2<x_1$
Number of **max = arr[i];**	2	1	1	0	1	0

E[Number of **max = arr[i];**] = (2 + 1 + 1 + 0 + 1 + 0) * 1/6 = 5/6

Remember This?

```
Public static int findMax(int [] arr)
{
        int max = arr[0];
        for( int i=1; i<arr.length; i++ )
        {
                if ( arr[i] > max ) max = arr[i];
        }
        return max;
}
```

So how do we do this in general?

How many inputs with **0** executions of **max = arr[i];** ? (n-1)! since max = x[0]

How many inputs with **n-1** executions of **max = arr[i];** ? 1, since only on ordered input

How many inputs with **k** executions of **max = arr[i];** ? Lets call it $P_n(k)$

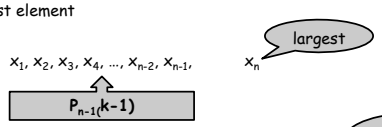
Average Runtime of Max

How many inputs with **k** executions of **max = arr[i];** ?

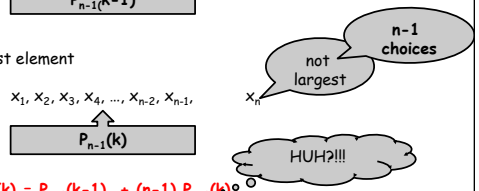
Lets call it $P_n(k)$

We have the following two possibilities:

1. either x_n is the largest element



2. or x_n is not the largest element



$P_n(k) = P_{n-1}(k-1) + (n-1) P_{n-1}(k)$

Must learn how to solve recurrences!

25