Analyzing the Fibonacci Sequence, Transitive Closure, and a Bubblish Sort

The n^{th} Fibonacci number

The 0^{th} number of the Fibonacci sequence is 0.

The 1^{st} number of the Fibonacci sequence is 1.

The n^{th} number of the Fibonacci sequence is defined as the sum of the previous two numbers in the sequence.

This is a recursive definition, and appears to be an excellent candidate for a recursive solution…

Recursive Algorithm

```
long fib(int n) {
   if (n<2) return n;
   return fib(n-1)+fib(n-2);
}
```

Note: I mix languages in pseudocode…

Let's assume that a comparison has a cost of 1 in terms of run-time, and that this is the only cost we care about.

We want to know the run-time of this algorithm on input n. We will call this $T(n)$.

Computing the Run-Time

Given the following recurrence:

$T(0)=T(1)=1$

$T(i)=1+T(i-1)+T(i-2)$

If we assume that $\exists x \in \mathbf{R}^{+}$ s.t. $T(n) \leq x^{n}$ then we can solve for x.

Can we do better?

Is there a way to improve the recursive algorithm if we are allowed to allocate an array? Consider the following example using memoization:

```
long fib(int n) {
static long Marr[1000]={0,1};
static int Mlast=1;
  if (n>Mlast) {
        long x=fib(n-1)+fib(n-2);
        Mlast=n;
        Marr[Mlast]=x;
  }
  return Marr[n];
}
```

Does this work?

What is it's run-time?

What about plain iteration?

```
long fib(int n) {
long first=0, second=1, tmp;
   for (int i=0; i<n; i++) {
         tmp = first+second;
         first = second;
         second = tmp;
   }
   return first;
}
```

Does it work?

What is it’s run-time?

Can we do better?

How about just a formula?

let $Phi = \frac{1+\sqrt{5}}{2}$

let $phi = \frac{1-\sqrt{5}}{2}$

$$Fib(n) = \frac{Phi^n - phi^n}{\sqrt{5}}$$

(proof of this left to HW)

Is this a faster way to compute the n^{th} Fibonacci number?

Transitive Closure

```
for outer = 1 to n
    for i = 1 to n
        for j = 1 to n
            for k = 1 to n
                if (R(i,j) ^ R(j,k))
                then R(i,k) = true;
```

This is an “overkill” implementation of transitive closure.

What is its runtime in terms of if statements?

Better Transitive Closure?

Is there a better algorithm? How do we define "better" when talking about algorithms?

Could we shave some iterations off the i, j, or k loops? Would doing so limit the types of graphs on which the algorithm would work? What would such a change (if valid) actually save?

Let's assume we could shorten each loop by one iteration. How do we calculate the runtime when the loops differ in starting and ending values?

Loops with Dependencies

As we explore more, we sometimes have loops in an algorithm that are not independent.

Example: BubblishSort

for $i = 1$ to $n-1$

for $j = i+1$ to n

if $(a_i > a_j)$ then swap(a_i, a_j);

What is the runtime in terms of if statements?