Use constructive induction...

Use constructive induction to set values for $a$ and $b$ in the following equation:

$$\sum_{i=1}^{n} i = an^2 + bn$$
The n\textsuperscript{th} Fibonacci number

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

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

The n\textsuperscript{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);
}

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:
\[ T(0)=T(1)=1 \]
\[ T(i)=T(i-1)+T(i-2) \]
If we assume that \( \exists x \in \mathbb{R}^+ \text{ s.t. } T(n) \leq x^n \)
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:

```java
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?

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

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

(proof of this left to homework)

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