Analysis of recursive algorithms

Recursion allows to state algorithms succinctly, and to prove their correctness by induction. We can also use it to analyze them.

Example A:
```c
int foo(int n)
{
    if (n < 0)
        return 0;
    else {
        a = foo(n-1);
        a *= n; // or some other small task
        return a;
    }
}
```

We can measure the running time with the number of recursive calls. By induction on \( n \): \( T(n) = T(0) + n \).
Example B, Tower of Hanoi: we start with a stack of disc on peg A, and we have to move all of them to peg B.

Rules:
(a) every disc has a different size;
(b) move one disc at the time;
(c) never put a larger disc on top of a smaller one.

Observations:
(a) to move the largest disc, we must have a peg with this disc only,
a peg with no discs and a peg with all other discs;
(b) preparing to move the largest disc is a similar task,
but one that involves one less disc—all but the largest one;
(b) finishing after we moved the largest disc is also a similar task.

```c
void Hanoi(peg A, peg B, peg C, int n)
{
    if (n == 0)
        return;
    Hanoi(A, C, B, n-1);
    move a disc from A to B;
    Hanoi(C, B, A, n-1);
}
```
We can measure the running time with the number of times we move a disc. 
\[ T(n) = 2T(n - 1) + 1 \]
\[ T(0) = 0 \]
\[ T(1) = 1 \]
\[ T(2) = 3 \]
\[ T(3) = 7 \]
\[ T(n) = 2^n - 1, \text{ guess and verify method} \]

Using induction, we can prove that the following algorithm is almost equivalent (all discs moved to another peg, but which one?):

```java
for (i = 1; i < 2^n; i++)
    if (i&1)
        move the smallest disc clockwise: A → B → C → A
    else
        make the only other legal move;
```

This gives another recursive approach: those other legal moves solve the problem for \( n - 1 \) discs, all but the smallest. This approach allows to solve problem iteratively.
\[ T(n) = cn + T(bn), \text{ where } b < 1 \]

Assume/guess that \( T(n) \leq dn \)

\[ T(n) = cn + T(bn) \leq cn + dbn \leq dn \]

The last inequality is what we need to complete the proof, but is it true?

We choose \( d \), so we will make it true!

Needed and sufficient: \( c + db = d \equiv d(1 - b) = c \equiv d = \frac{c}{1 - b} \)

This is a special case of the Master Theorem.
Example C, Fibonacci recurrence:
\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_{i+2} = f_{i+1} + f_i \]
\[ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... \]

Solution with assumption: \( f_i = x^i \)
\[ x^{i+2} = x^{i+1} + x^i \]
\[ x^2 = x^1 + x^0 \]
\[ x^2 - x - 1 = 0 \]
\[ x = \frac{1 \pm \sqrt{5}}{2}, \text{ so } x_0 = \frac{1 + \sqrt{5}}{2}, x_1 = \frac{1 - \sqrt{5}}{2} \]

Linear combination of solutions satisfies the same recurrence relation.
We can find the correct combination, one that fits our \( f_0 \) and \( f_1 \):
\[ a(x_0^0) + b(x_1^0) = 0, \text{ thus } b = -a \]
\[ a(x_0^1) + b(x_1^1) = 1 \]
\[ ax_0 - ax_1 = 1 \]
\[ a(x_0 - x_1) = 1 \]
\[ a\sqrt{5} = 1 \]
\[ a = \frac{1}{\sqrt{5}} \]

\[ f_i = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^i - \left( \frac{1 - \sqrt{5}}{2} \right)^i \]
Example D, Master Theorem recurrence:

\[ T(n) = aT(n/b) + n^c \]

This gives a tree of recursive calls
We have \( \log_b n \) levels in the tree
Level 0: this is root level, one call of size \( n \)
Level \( i \): \( m_i \) calls of size \( s_i \), they make
\[ am_i \text{ calls of size } s_i b^{-1} \]
Thus:
\[
\begin{align*}
m_0 &= 1, \quad m_i = am_{i-1} \\
s_0 &= n, \quad s_i = s_{i-1}/b \\
\end{align*}
\]

work of level \( i \) is
\[
m_is_i^c = a^i(nb^{-i})^c = (ab^{-c})^i n^c
\]

Cases: root dominates, leaves dominate, levels are equal

Root dominates: \( A = ab^{-c} < 1 \),
the total work is not larger than
\[
\left( \sum_{i=0}^{\infty} A^i \right) n^c = \Theta(n^c)
\]

Leaves dominate: \( A = ab^{-c} > 1 \),
the level \( L \) of leaves satisfies
\[
\begin{align*}
nb^{-L} &\approx 1 \\
\Rightarrow &\quad b^L \approx n \\n\Rightarrow &\quad L \approx \log_b n
\end{align*}
\]
the work of a call on the leaf level is \( \Theta(1) \)
so the total work is \( \Theta(m_L) \)
\[
m_L = a^{\log_b n} = n^{\log_b a}
\]

Balanced: \( A = ab^{-c} = 1 \),
work on levels is roughly equal, so we multiply work of level 0
with the number of levels
\[
\Theta(n^c \log n)
\]