Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2025

Topic Notes: Analysis with Recurrences

In our earlier work on analysis fundamentals, we examined several algorithms with loop structrues that could be analyzed by setting up summations.

We next consider a very different example, an algorithm to determine how many bits are needed to represent a positive integer in binary.

```
ALGORITHM BinDigits($n$)
    $count \leftarrow 1$
    while $n > 1$ do
        $count \leftarrow count + 1$
        $n \leftarrow \lfloor \frac{n}{2} \rfloor$
    return $count$
```

Our summation techniques will not work here – while this is not a recursive algorithm, the approach here will involve recurrence relations, which are usually applied to recursive algorithm analysis.

Analyzing Recursive Algorithms

Our approach to the analysis of recursive algorithms differs somewhat. The first three steps are the same: determining the input size parameter, identifying the basic operation, and separating best, average, and worst case behavior.

Setting up a summation is replaced by setting up and solving a *recurrence relation*.

Example: Computing a Factorial

We start with a simple recursive algorithm to find $n!$:

```
ALGORITHM Factorial($n$)
    if $n = 0$ then
        return $1$
    else
        return $n \cdot$ Factorial($n - 1$)
```

You are hopefully familiar with recurrence relations from your math experience, but don't worry, we'll talk about how to set them up and to solve them.

The recurrence for this problem is quite simple:

$$M(n) = M(n-1) + 1$$

Why? The total number of multiplications, $M(n)$, to compute $n!$ is the number of multiplications to compute $(n-1)!$, which we can denote as $M(n-1)$, plus the 1 to get from $(n-1)!$ to $n!$.

We do need a *stopping condition* or *base case* for this recurrence, just as we have a stopping condition for the recursive algorithm. For $n = 0$, we do not need to do any multiplications, so we can add the initial condition $M(0) = 0$.

We can easily determine that $M(n) = n$ just by thinking about this for a few minutes. But instead, we will worth through this by using back substitution.

$$\begin{aligned} M(n) &= M(n-1) + 1 \\ &= [M(n-2) + 1] + 1 = M(n-2) + 2 \\ &= [M(n-3) + 1] + 2 = M(n-3) + 3 \end{aligned}$$

Our goal is to find the pattern – what would our recurrence look like in terms of some $M(n-i)$?

$$M(n) = M(n-i) + i$$

In order to complete the process, we need to be able to apply the base case. If we choose $i = n$, that will allow us to do so:

$$M(n) = M(n-n) + n = M(0) + n = n \in \Theta(n).$$

Example 2: Towers of Hanoi

Many of you are all likely to be familiar with the Towers of Hanoi.

Recall that solving an instance of this problem for n disks involves solving an instance of the problem of size $n-1$, moving a single disk, then again solving an instance of the problem of size $n-1$. We denote the number of moves to solve the probem for n disks as $M(n)$.

We will analyze this algorithm as part of our handout.

Example 3: Number of Binary Digits Needed for a Number

We return now to the problem of determining how many bits are needed to represent a positive integer in binary.

To find the recurrence more readily, we recast the problem recursively:

```
ALGORITHM BinDigits($n$)
    if $n = 1$ then
        return $1$
    else
        return BinDigits($\lfloor \frac{n}{2} \rfloor$) $+ 1$
```

Again, we proceed on our handout.

The problem becomes a bit complicated by the presence of a floor function. We can be precise and apply backward substitution only if we assume that n is a power of 2. Fortunately, we can do this and still get the correct order of growth (by the *smoothness rule*, which we will just take as fact – interested students can find a proof).

Example 4: Another recurrence example

Suppose in the analysis of some algorithm, we find the following recurrence:

$$C(n) = 2C(n/2) + 2$$

when $n > 0$, and a base case of $C(1) = 0$.

We proceed on the handout to solve this recurrence.