CIS 575. Introduction to Algorithm Analysis
Material for January 29, 2024

A Recipe for Estimating Sums

1 A Recipe For Analyzing Loops

From the previous note, we get a key result, expressed as

Theorem 3.28 in Howell's textbook Assume that

$$T(n) = \sum_{i=1}^{g(n)} f(i)$$

If f is non-decreasing then

$$\mathbf{T(n) \in O(g(n) \cdot f(g(n)))}.$$

If f is non-decreasing and **smooth** (and g eventually becomes greater than any constant) then

$$\mathbf{T(n) \in \Theta(g(n) \cdot f(g(n)))}.$$

Applying Howell's Thm 3.28 to Our Motivating Example Recall that the running time $T(n)$ of the program

```
for i ← 1 to n²
    for j ← 1 to i³
        k ← k + i + j
```

is given by

$$T(n) = \sum_{i=1}^{n^2} i^3$$

Since i^3 is a smooth function (as is any polynomial with positive coefficients), we can apply *Howell*'s Thm 3.28 to get the known result:

$$T(n) \in \Theta(n^2 \cdot (n^2)^3) = \Theta(n^8)$$

Applying Thm 3.28 to Double Nesting Now consider the program

```
m ← 0
for i ← 0 to n
    for j ← i to n
        sum ← 0
        for k ← i to j − 1
            sum ← sum + A[k]
        m ← Max(m, sum)
return m
```

It may not be clear what this program is supposed to accomplish (but the curious reader may study Section 1.6 in *Howell*'s textbook). Fortunately, our newly developed recipe for analyzing the *running time* of a program does *not* require that we understand the *meaning* of the program.

Let us first observe that the outer, middle and inner loop each iterates at most $n+1$ times, and since the body of the inner loop executes in constant time, the total running time must be in $O(n^3)$. But since the inner loop often doesn't iterate many times (when i and j are close to each other), one *may* suppose that it is possible to get a tighter bound, say $\Theta(n^2)$. To investigate whether this is the case, let us embark on a systematic analysis.

The **inner loop** (for given values of i and j) iterates $j-i$ times, with each iteration taking constant time; hence its running time is (proportional to) $\mathbf{j-i}$.

The **middle loop** (for given value of i) iterates $n-i+1$ times. In the qth iteration, $j = q+i-1$; hence the running time of the qth iteration is (proportional to) $q-1$. The running time is thus given by the sum

$$\sum_{q=1}^{n-i+1} (q-1)$$

which fits the pattern of *Howell*'s Theorem 3.28: as $q-1$ is a smooth function, we infer that the running time of the **middle loop** (for given value of i) is in $\Theta((n-i+1)(n-i))$ which equals $\boldsymbol{\Theta}((\mathbf{n}-\mathbf{i})^{\mathbf{2}})$.

The running time of the **outer loop** can thus be expressed as (is proportional to) the sum

$$\sum_{i=0}^{n} (n-i)^2$$

where we need to be careful: since $(n-i)^2$ is *not* a non-decreasing function of i, *Howell*'s Theorem 3.28 is not immediately applicable. But clearly, the above sum is equivalent to

$$\sum_{i=0}^{n} i^2 = \sum_{i=1}^{n} i^2$$

which since i^2 is a smooth function allows us to infer that the **total running time** is in $\Theta(n \cdot n^2) = \boldsymbol{\Theta}(\mathbf{n^3})$ (our initial rough approximation thus happened to be a tight bound).

Analyzing While Loops For a **while** loop, it may take a little effort to apply *Howell*'s Theorem 3.28. For instance, consider the program

$x, k \leftarrow 0, 1$
while $k \leq n$
    $k \leftarrow k + k$
    $x \leftarrow x + 1$

where it may not be obvious how to express its running time. But observe that in the i'th iteration, k will increase from 2^{i-1} to 2^i. The loop will terminate when $k > n$, which will happen when $2^i > n$, that is roughly after $\lg(n)$ iterations. The running time is thus proportional to

$$\sum_{i=1}^{\lg(n)} 1$$

which is obviously in $\mathbf{\Theta}(\lg(\mathbf{n}))$.