13.4 Asymptotic Behavior

Be on your best behavior!

In most applications, we are most concerned about what happens for very large values of N. This is known as the *asymptotic behavior*. We want to learn what types of algorithms are able to handle large amounts of data. Some examples of applications that require highly efficient algorithms are:

- Simulating the interactions of billions of particles
- Maintaining a social network with billions of users
- Encoding billions of bytes of video data

Algorithms that scale well have better *asymptotic* runtime behavior than algorithms that scale poorly.

Let us return to the original problem of characterizing the runtimes of our `dup` functions. Recall that we desire characterizations that have the following features:

- Simple and mathematically rigorous.
- Clearly demonstrate the superiority of dup2 over dup1.

We've accomplished the second task! We are able to clearly see that `dup2` performed better than `dup1`. However, we didn't do it in a very simple or mathematically rigorous way. Luckily, we can apply a series of simplifications to solve these issues.

Simplification 1: Consider Only the Worst Case

When comparing algorithms, we often only care about the worst case. The worst case is often where we see the most interesting effects, so we can usually ignore all other cases but the worst case.

Example:

Consider the operation counts of some algorithm below. What do you expect will be the order of growth of the runtime for the algorithm?

- N (linear)
- N^2 (quadratic)
- N^3 (cubic)
- N^6 (sextic)

Operation	**Count**
less than (<)	$100N^2 + 3N$
greater than (>)	$N^3 + 1$
and (&&)	5000

Answer: N^3 (cubic)

Intuitively, N^3 grows faster than N^2, so it would "dominate." To help further convince you that this is the case, consider the following argument:

-

Suppose the < operator takes α nanoseconds, the > operator takes β nanoseconds, and the && takes γ nanoseconds.

- The total time is $\alpha(100N^2 + 3) + \beta(2N^3 + 1) + 5000\gamma$ nanoseconds.
- For very large N, the $2\beta N^3$ term is much larger than others.
- It can help to think of it in terms of calculus. What happens as N approaches infinity? Which term ends up dominating?

Simplification 2: Restrict Attention to One Operation

Pick some representative operation to act as a proxy for overall runtime. From our `dup` example:

- Good choice: `increment`, or **less than** or **equals** or **array accesses.**
- Bad choice: **assignment of** `j = i + 1`, or `i = 0.`

The operation we choose is called the “**cost model**.”

Simplification 3: Eliminate Low Order Terms

Ignore lower order terms.

Sanity check: Why does this make sense? (Related to the checkpoint above!)

Simplification 4: Eliminate Multiplicative Constants

Ignore multiplicative constants.

- Why? No real meaning!
- By choosing a single representative operation, we already “threw away” some information.
- Some operations had counts of $3N^2$, $N^2/2$, etc. In general, they are all in the family/shape of N^2.

Checkpoint Exercise:

Apply our four simplification rules to the `dup2` table.

Operation	Symbolic Count
i = 0	1
j = i+1	0 to N
<	0 to $N-1$
==	1 to $N-1$
array accesses	2 to $2N-2$

Example answer: array accesses with order of growth N.

<, ==, and j=i+1 would be fine answers as well.

Simplification Summary

- Only consider the worst case.
- Pick a representative operation (aka: cost model)
- Ignore lower order terms
- Ignore multiplicative constants.

Previous
13.3 Checkpoint: An Exercise

Next
13.6 Simplified Analysis Process