Recitation 1, CIS-2166

David Dobor

January 15, 2016

Section 3.2 of Rosen's textbook contains an in-depth discussion of the asymptotic notation. This note is intended to complement the discussion in the book and may be useful to ease your understanding of that section. Some example problems and their solutions, similar to the ones you'll see on your homework, can be found at the end of this note. We will see more examples in following recitations.

Why use Big-O?

Suppose you're writing a program that manages inventory for some nation-wide chain of stores and that every night your job is to process the changes that occured in the inventory during that day. Suppose it takes about $10,000$ *ms* for your program to read in the data from the disk and about 10 *ms* to process each item in the inventory. If you have n items in the inventory then the running time of your program may be represented by:

$$T(n) = 10,000 + 10n$$

For large n, the run-time of your program will be "dominated" by the second term: the time to read in the data (10 000 *ms*) will become negligible compared to the time it takes to process a large amount of inventory items.

In asymptotic analysis we want to abstract away from $T(n)$ the details that do not matter in the long run, like the time it takes to read in the data in this example, or the time it takes to process an individual item. These will vary depending on whether you are running your program on a modern super-computer or on a desktop from the 80-s that is still stashed away in your parents' garage. We just want to draw some general conclusions about the running time of the program you wrote.

To help reason about $T(n)$, we introduce another function, $f(n)$, that is simpler than $T(n)$. We use $f(n)$ for an upper bound on $T(n)$. This $f(n)$ will enable us to draw conclusions about how fast the inventory processing algorithm runs, without letting the details get in the way.

Here's a definition of Big-O that is slightly less formal than what the book has:

DEFINITION: We say that $T(n)$ is Big-O of $f(n)$ and write $T(n) \in O(f(n))$ if and only if

$$T(n) \leq Cf(n) \text{ whenever } n \text{ is big for some large constant } C$$

Well, but what is "big n"? Answer: big enough to have the first function, $T(n)$, "fit below" the second one, $f(n)$. We'll look at an example in a second. We'll give this "big enough n" a name - we'll call it k, as the textbook does. And what is "large enough C"? The answer is the same. *And* we get to choose C.

That is, we get to choose the constants C and k such that

$$T(n) \leq Cf(n) \quad \text{whenever } n > k$$

If we can identify such C and k, then we say that $T(n) \in O(f(n))$; Otherwise, if we can prove that no such C and k exist, then we say that $T(n) \notin O(f(n))$

Whenever you see $T(n) \in O(f(n))$, it may be helpful to think: "$T(n)$ is smaller than $f(n)$" or that "$f(n)$ dominates $T(n)$".

(*As an aside*, note that we do not usually write that $T(n) = O(f(n))$. We instead think of $f(n)$ as being a member - or being "like" other members - of a *set* of functions denoted by $O(f(n))$. However, people often use the "=" sign, and that is something to be aware of. We'll stick with the "$\in$" sign in our discussions.)

Examples

EXAMPLE 0: Consider comparing $T(n) = 10 + 2n$ to $f(n) = n$. We want to see if $f(n)$ will dominate $T(n)$ for large enough n. That is, will the plot of $f(n)$ be above the plot of $T(n)$ if we were to plot the two functions? In fact, let's plot them.

Let's set the constant C in the above definition to 1 and see what the plots look like. If you are inclined to use Matlab (not required for this course), here's how to do it.

```
n = linspace(0,20,100);
T = 10 + 2*n;
f = 1*n;

figure(1)
```

```
plot(n,T, 'linewidth', 2, 'color', 'g')
xlim([0 20])
ylim([0 50])
hold on
plot(n,f, 'linewidth', 2, 'color', 'b')
legend('T(n) = 10 + 2 n', 'f(n) = n', 'location', 'northwest')
```

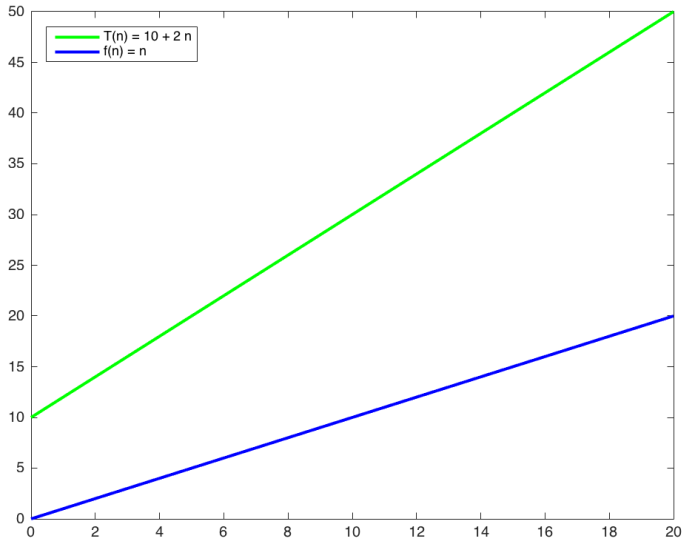

Figure 1: Here $T(n)$ dominates $f(n)$ everywhere, meaning that the plot of $f(n)$ fits entirely under the plot of $T(n)$.

At first it may appear to you that since $f(n)$ is below $T(n)$, we can't really claim that $T(n) \in O(f(n))$.

But recall from the definition that we get to choose C. In the above figure, C is 1. Now consider setting C to something like 5. We then can have $f(n)$ dominate $T(n)$.

```
f = 5*n;
plot(n,T, 'linewidth', 2, 'color', 'g')
xlim([0 20])
ylim([0 50])
hold on
plot(n,f, 'linewidth', 2, 'color', 'b')
legend('T(n) = 10 + 2 n', 'f(n) = 5 n', 'location', 'northwest')
```

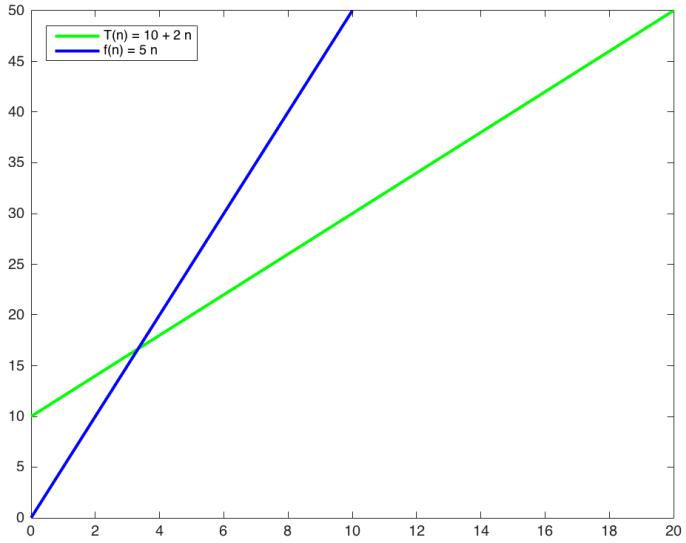

Figure 2: Here $f(n)$ begins to dominate $T(n)$ after a certain point.

So now we see that after a certain point f begins to dominate T instead (what is that point?). Now it should be clearer how f and T in this example fit the Big-O definition.

Now consider other examples similar to what you will find in your homework. The trick to solving these problems is to explicitly identify the constants C and k that fit the above definition. We rely on the mathematical fact that $\log(n) \leq n$ for $n \geq 1$.

EXAMPLE 1:

$5n^2 + 3n \log n + 2n + 5$ is $O(n^2)$

JUSTIFICATION: $5n^2 + 3n\log n + 2n + 5 \leq (5+3+2+5)n^2 = Cn^2$ for $C = 15$ when $n \geq k = 1$

EXAMPLE 2:
$20n^3 + 10n\log n + 5$ is $O(n^3)$

JUSTIFICATION: $5n^2 + 3n\log n + 2n + 5 \leq 35n^2$ for $n \geq k = 1$

EXAMPLE 3:
$3n\log n + 5$ is $O(\log n)$

JUSTIFICATION: $3n\log n + 5 \leq 8\log n$ for $n \geq n_0 = 2$. Note that $\log n$ is zero for $n = 1$. That's why we use $k = 2$ here.

EXAMPLE 4:
2^{n+2} is $O(2^n)$

JUSTIFICATION: $2^{n+2} = 2^n \cdot 2^2 = 4 \cdot 2^n$, so we can take $C = 4$ and $k = 1$ in this case.

EXAMPLE 5:
$2n + 100\log n$ is $O(n)$

JUSTIFICATION: $2n + 100\log n \leq 102n$ for $n \geq 1$, so we can take $C = 102$ and $k = 1$ in this case.