Asymptotic Estimates

Yan Tao

Advanced 1 - Spring 2024

1 Comparing Rates of Growth

Problem 1 *A king offers a mathematician 1000 gold coins for their mathematica abilities.*

- *Alternatively, after some bargaining, the king offers instead to pay him one coin for every hour for one day. Should the mathematician take the deal?*

- *After some more haggling, the king improves his offer to one coin per minute. Is this now a good deal?*

- *Would the deal still be good if it were per second? Per millisecond? Per any smaller unit of time?*

Problem 2 *Now that the mathematician is hired, they can begin dealing with some mathematical issues. A neighboring kingdom has agreed to ship 100 bags of rice every day for the next month. The king wants to send the following counteroffer. They will send one bag of rice on the first day, two on the second day, four on the third day, and so on, each day sending double what was send the previous day. Would this counteroffer be better than the current offer of 100 bags per day?*

Problem 3 *Suppose in this kingdom, a month is instead exactly 40 days. The king sends the same counteroffer. Is the counteroffer better? What if it were 50 days? 100 days?*

Problem 4 *The neighboring kingdom sends a counter-counter offer. Instead of doubling the rice every day, they will send 10 bags on the first day, 20 on the second, 30 on the third, and so on, increasing by 10 on each day. Is this better than the original offer? How about the counteroffer?*

Problem 5 *Again suppose a month is 40 days (or 50, or 100). Does this change which offers are better than which?*

2 Asymptotic Notation

In the previous example, suppose a month were 300 or 1000 days long. Our calculators quickly stop being able to compute how many bags of rice our kingdom would receive. (Of course, the world would presumably run out of rice long before that point, but we could imagine that we're instead modelling something like the growth of a bacterial colony.) Instead, we can estimate how large certain numbers are, in terms of certain other large numbers.

Definition 1 *(Big O notation) Let f, g be functions of positive integers.*

If there is a constant C such that $f(n) \leq Cg(n)$ for all large enough positive integers n, we write $f(n) = O(g(n))$ as $n \to \infty$, or $f = O(g)$ for short. [1]

Problem 6 *Show the following:*

- $3n^3 = O(n^4)$

- $14n^2 = O(n^2)$

- $n^2 + 500n = O(n^2)$

Problem 7 *Consider all the scenarios from Problems 1-5, and let N equal the amount of times gold or rice is paid. Determine each offer's amount in terms of N, and use your previous work to write these as big O of each other. Which ones, in general, are the "better" offers?*

[1] Technically, the big O is a capital greek letter omicron. But since there is so little difference when typed or handwritten, we still call it big O.

Big O notation is helpful for seeing that one function is asymptotically smaller than (or equal to) another. But we don't yet have any way of describing whether functions are asymptotically the same. So we now introduce the analogue of $\geq$ for asymptotics:

Definition 2 *(Big Omega and Big Theta)*

- *If there is a constant C such that $f(n) \geq Cg(n)$ for all large enough positive integers n, we write $f(n) = \Omega(g(n))$ as $n \to \infty$, or $f = \Omega(g)$ for short.* [2]
- *If $f = O(g)$ and $f = \Omega(g)$, we say that $f = \Theta(g)$ (as $n \to \infty$).*

Problem 8 *Show the following:*

- $n^2 = \Omega(20n)$

- $n^2 + 500n = \Theta(n^2)$. *(Note: It will help to use what you showed as part of Problem 6.)*

- $n^5 + 50n^4 + 1000n^3 + 10000n^2 + 50000n + 100000 = \Theta(n^5)$

Problem 9 *Show that $f = \Omega(g)$ if and only if $g = O(f)$.*

Last time, we encountered a couple different definitions of the number e. To recap, we showed that

$$e^n = \frac{n^0}{0!} + \frac{n^1}{1!} + \frac{n^2}{2!} + \frac{n^3}{3!} + \frac{n^4}{4!} + \dots$$

Problem 10 *Show that any polynomial in n is $O(e^n)$.*

[2] This version of big omega, due to Knuth, is the version used in combinatorics and computer science. One may encounter a different definition of big omega in number theory, which is not the same.

We also defined logarithms last week in order to deal with especially large numbers. This page serves as a reminder of the definitions and properties, and the problems should be done if you did not do them last week.

Definition 3 *For $b > 1$, the **base b logarithm** is defined by*

$$\log_b(b^x) = b^{\log_b(x)} = x$$

*for all $x > 0$. (In other words, $\log_b$ is the inverse function of b^x.) We call the base e logarithm the **natural logarithm**, denoted* ln.

Problem 11 *Show that*

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

Use this fact to prove that

$$\log_b(x^n) = n \log_b(x)$$

Problem 12 *(Base change formula) Show that*

$$a^{\log_a(b)x} = b^x$$

for any x and any $a, b > 1$. Use this to show that

$$\log_a(y) = \log_a(b) \log_b(y)$$

for any $y > 0$ and any $a, b > 1$.

Problem 13 *Show that* $\log_a(n) = \Theta(\log_b(n))$ *for all* $a, b > 1$.

Problem 14 *Explain why, when using a big O related approximation, the base of a logarithm doesn't matter. (So we'll write, for example,* $O(\log n)$ *if something is* $O(\ln n)$ *or if it's* $O(\log_2(n))$*.)*

Problem 15 *Using Problem* 10*, show that* $\log(n) = O(n)$.

Problem 16 *Using the previous problem, show that a polynomial in* n *is* $O(b^n)$ *for any* $b > 1$.

3 Using Big O to Count Lots of Things

In both combinatorics and computer science, the preferred logarithm base is 2 (i.e. binary). From now on, all logarithms, unless indicated otherwise, are base 2. Also, the following approximation is very helpful.

Problem 17 *Show that for fixed k, $\binom{n}{k} = \Theta(n^k)$. (Hint: See Problem 12 from last week's worksheet, or ask your instructors about the result if you don't have it.)*

Problem 18 *Let F_n be the n^{th} Fibonacci number, which is defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$. Show that $F_n = O(2^n)$.*

Problem 19 *Let G_n be the number of **simple graphs** (that is, a graph with no edge from a vertex to itself, and at most one edge between any pair of distinct vertices) with n vertices. Find $\log(G_n)$.*

Problem 20 *We say two graphs are **isomorphic** if there is a way to relabel their vertices to make them the same. Let H_n be the number of simple graphs with n vertices up to isomorphism. Find an asymptotic upper bound for $\log(H_n)$ (that is, some $f(n)$ such that $\log(H_n) = O(f(n))$).*

Problem 21 *Find an asymptotic lower bound for $\log(H_n)$ (that is, some $f(n)$ such that $\log(H_n) = \Omega(f(n))$).*

One application of big O is comparing different algorithms, when run on big inputs.

Definition 4 *Given an algorithm, its* ***(worst-case) runtime*** *is the smallest N such that it will always be done in at most N steps.*

Problem 22 *Given a large integer n, and another large integer $m \approx n$, written in decimal form, consider the following two algorithms for multiplying them:*

- *Algorithm A: Start with 0, add n, add n again, repeat m times.*
- *Algorithm B: Multiply the last digit of m by the last digit of n, then by the second-to-last digit (carrying over any tens), then the next digit, and so on. Repeat with each digit of m, and add the results. (This is likely the multiplication algorithm you've learned from elementary school).*

Give asymptotic upper bounds for the runtimes of Algorithms A and B. Which one is better?

Problem 23 *The* ***Euclidean algorithm*** *finds the greatest common divisor of two positive integers n and m as follows. Suppose without loss of generality that $m > n$. We divide m by n. If m is divisible by n, then we stop the algorithm and the answer is n. Otherwise, we replace m by the remainder of this division and repeat. Show that the Euclidean algorithm's runtime is $O(\log n)$.*

Problem 24 *The* ***bubblesort*** *algorithm sorts a list as follows. Suppose we have a list of numbers $\{1, \ldots, n\}$ appearing in any order. We run through the list, and if any element is larger than the immediately following element, we switch them. If we run through the entire list and make no switches, the list is sorted and we stop. Give an asymptotic upper bound for bubblesort's runtime.*

Problem 25 *The* ***mergesort*** *algorithm sorts a list as follows. We split the list in half, then split those halves in half, and keep going until the pieces are sorted. Then we put the pieces back together by comparing their elements, to put them in order. Give an asymptotic upper bound for mergesort's runtime.*

Problem 26 *(Bonus) Show that sorting a list is $\Omega(n \log n)$, so that this is the "best" we can do.*[3] *(Hint: How much information is needed to sort a list of n values? It will also help to recall Stirling's Formula from last week.)*

[3]Though mergesort is the best sorting algorithm asymptotically, the constant that comes with the big O is not the most efficient for most real-life data sets. The quicksort algorithm, which has a worse worst-case runtime, sorts "most" lists a bit faster, and is what your computer uses to sort lists.