CS 305
Design and Analysis of Algorithms

09 / 09 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions / Comments?
- More about big O, big Omega, big Theta
- Loop invariants to prove correctness
- Little-o, little-omega
- Arithmetic Series

Algorithm / Problem

- What's an algorithm?
  - Finite sequence of computational steps that takes input and produces an output
- What's a computational problem?
  - Specified input / output relation (i.e. given certain input, produces certain output)
- What's an instance of a problem?
  - Some particular input
- What kinds of things do we want to analyze about an algorithm?
  - Efficiency (speed, space)
  - Correctness

FindMax problem and algorithm

- Given a list of numbers, find the maximum number in the list.
- Let's write pseudocode on the board for an algorithm that solves this.
  - Note indentation to indicate blocks
  - Left arrow assignment
  - Among other pseudocode conventions

FindMax problem and algorithm

- **Loop invariants** for determining correctness
- Must show three things about a loop invariant
  - Initialization: some property is true before loop
  - Maintenance: that property is true before each loop iteration
  - Termination: algorithm is correct if loop terminates and the reason the loop terminated and the loop invariant allows us to show that the algorithm is correct
- What's the loop invariant for findMax?

FindMax problem and algorithm

- What is true before the loop and before each subsequent iteration?
- That is the loop invariant for findMax.

FindMax problem and algorithm

- We use **n** to denote the size of the list.
- What is true before the loop and before each subsequent iteration?
- What's the loop invariant for findMax?
  - maxSoFar is largest value in sub list[1 .. i-1]
- Termination: when i reaches n+1 the loop terminates. So the maxSoFar is the largest value in the sub list[1 .. n]
- Together these imply that the maxSoFar is the largest value in the entire list

FindMax problem and algorithm

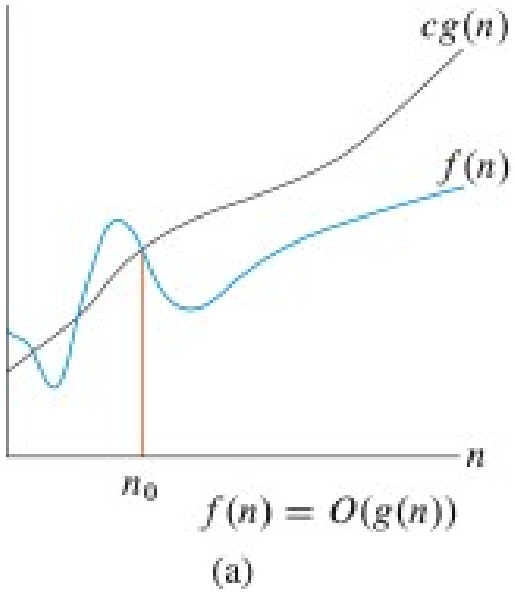
- Last time we analyzed it for efficiency (speed).
  - We considered
  - best case (best instance)
  - worst case (worst instance)
  - Could consider average case – though we often need to know probability distribution of the inputs
  - **FindMax was determined to run in big Theta(n) time**
- Abstract away from what machine it's being run on.
  - Assume (different) constant time for each line

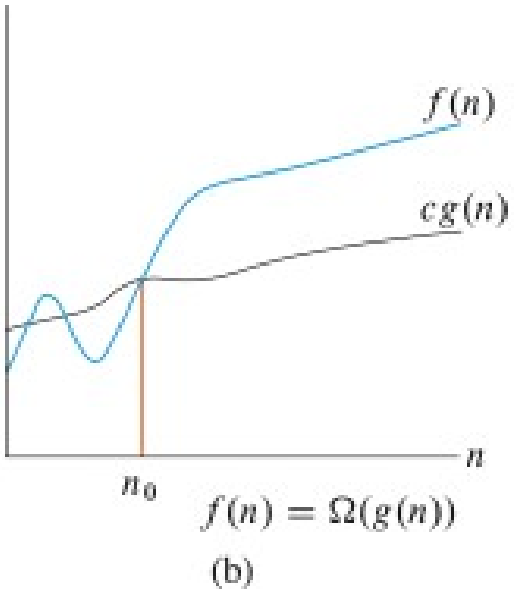
FindMax problem and algorithm

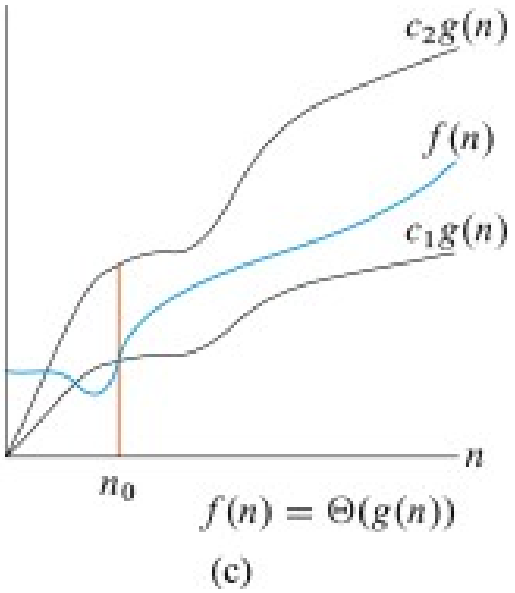
- Let's analyze it for efficiency (speed).
  - we'll focus mostly on best case and worst case running time in this course because
    - Gives an upper bound on running time
    - Worst case can happen often for some algorithms
    - Average is often just as bad
    - Worst case is easier to determine than average

Asymptotic notation

- big O
  - When we say f(n) is O(g(n))
  - g(n) is an upper bound on f(n)
  - Let's look at figure 3.2 in text







Asymptotic notation

- The running time of findMax is O(n) because n is an upper bound on a*n + b (where a and b are constants (not dependent on n)). Note: n is also a lower bound on a*n + b. Together those make n a tight bound on a*n + b.

- Note:
- a*n + b is also $O(n^2)$, O(n*log n), O(n!), $O(2^n)$
  - Because n^2, n*log n, n!, 2^n are all upper bounds on a*n + b
- a*n + b is NOT O(1), O(log n)

Asymptotic notation

- A reminder of the meaning of “for all” or “for any” vs. “there exists”
- Definitions of Big O, Big Omega and Big Theta on the board.
- Note that these define sets of functions, but we typically say “is” instead of “is in the set”
- Because we cannot do better than n for findMax, the overall (including best and worst cases) running time of findMax is Big Theta (n) – it is an asymptotically **tight bound**
  - Notice that we must compare each element to the maxSoFar and since there are n elements we cannot do better than n-1 compares

Asymptotic notation

- Why use asymptotic behavior for efficiency?
  - Focuses on large input sizes (large values of n)
  - Ignores small inputs (small values of n) because we may be able to create special purpose algorithms if the input is expected to be small
  - Ignore constants
    - Faster machines can combat constant factors
    - Faster machines cannot combat poor asymptotics

Asymptotic notation

- Analyze space of findMax
  - Inputs are the list and the size of the list
  - Additional space for maxSoFar and i
  - Only care about the additional space required beyond the inputs for space analysis
  - Space complexity of findMax is Big Theta (1)

Asymptotic notation

- Idea of tight bound vs. not tight bound
- e.g.
- $2*n^2 = O(n^2)$ is asymptotically tight so we can say $2*n^2 = \text{BigTheta}(n^2)$
- $2*n = O(n^2)$ is NOT asymptotically tight (but it is correct to say)
- One can use little o to denote an upper bound that is NOT asymptotically tight
- Definition on the board
- Note: $2*n = o(n^2)$ BUT $2*n^2 \; != \; o(n^2)$

Asymptotic notation

- Little omega is the corresponding not asymptotically tight bound for lower bounds
- Definition on the board

$f(n)$ is $o(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$f(n)$ is $O(g(n))$ iff $\exists\ c > 0,\ n_0 > 0\ \ni$
$0 \leq f(n) \leq c\, g(n)\ \not\forall\ n \geq n_0$

$f(n)$ is $\Theta(g(n))$ iff $f(n)$ is $O(g(n))$ AND
$f(n)$ is $\Omega(g(n))$.

$f(n)$ is $\Omega(g(n))$ iff $\exists\ c > 0,\ n_0 > 0\ \ni$
$0 \leq c g(n) \leq f(n)\ \not\forall\ n \geq n_0$

$f(n)$ is $\omega(g(n))$ iff $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Asymptotic notation

- Let's use that limit definition to show that 2n is little-o (n^2)

Arithmetic Series

- Let me introduce Arithmetic Series on the board.
- This is one of 3 sums you should memorize.
- The other two being Geometric and Harmonic.
- See Appendix A of our text (pgs. 1141, 1142)

Arithmetic Series

Let's prove the sum is big $O(n^2)$ on the board

I leave you to prove that it is also Big Omega of n^2

Which means that it is Big Theta of n^2

Sorting algorithms

- Reminder of what Selection Sort and Insertion Sort do (online animation).
- Let's discuss Selection Sort.
- Keep current "last".
- Continually find max element index in list and swap with the current "last".
  - How long does 1 find max take?
  - How many times do we do find max?
  - Are there any best or worst case situtations? All the same?

Sorting Algorithms

- Insertion Sort pseudocode on the board with run time analysis.
- Are there any best or worst case situtations? All the same?

Sorting Algorithms

- Let's look at an instance of Merge Sort and see how it works.
- Are there any best or worst case situtations? All the same?
- Let's see a recursion tree for n elements of MergeSort.
  - How many elements processed in each level?
  - How many levels?