CS5800 Introduction

CS5800-07 Algorithms

Fall 2020, Silicon Valley Campus

Introduction

- Goals: learn the basic concepts for designing and analyzing algorithms
- Instructional Staff:
  - Instructor: Anurag Bhardwaj
  - TA: Vishal Annamaneni, Zijun Wan, Gongzhan Xie
- Evaluation:
  - Midterm exam: 30%
  - Final exam: 30%
  - Individual assignments: 40%
- Suggested book: Introduction to Algorithms, Third Edition, by Cormen, Leiserson, Rivest, Stein from MIT Press

CS5800 Module 1

What is an Algorithm?

A Sequence of Instructions to Perform a Task

Formal Definition

- A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer: *a basic **algorithm for** division*
- What is important in an algorithm?
  1. Its correctness: it does what is suppose to do
  2. How fast it is: its time performance
  3. How many resources it uses:
     - Main memory
     - Secondary memory
     - Communication
     - Others ….
- Example: [What is an algorithm?]
- There can be many algorithms for the same problem, but ….

Etymology

- The word algorithm comes from a famous mathematician
  - [Muḥammad ibn Mūsā **al'Khwārizmī**]() (~780-840 AC)
  - He worked in the House of Wisdom of the Caliphate of Baghdad
- Main contributions:
  - *The Compendious Book on Calculation by Completion and Balancing* (~820)
    (Arab: *al-Kitāb al-mukhtaṣar fī ḥisāb* ***al-jabr*** *wal-muqābala*)
  - *The Book of Addition and Subtraction According to the Hindu Calculation*
  - Astronomical tables of *Siddhanta* (includes trigonometric tables, 820)
  - *Book of the Description of the Earth* (833)

Historical Context: Caliphate of Baghdad

Example: Euclid's Algorithm

- This algorithm finds the greatest common divisor (GCD) of two numbers
  - Replace the largest number by the difference of the largest with the smallest
  - Stop when the two numbers are equal
  - That number is the GCD
- This is probably the oldest attributed algorithm (300 BC)
- Can be done faster?
  - Yes. Replace the largest by the reminder of dividing the largest with the smallest
  - Steps are proportional to 5 times the number of digits of the smallest number
  - That is $5 \times \lceil \log_{10}(\text{smallest number}) \rceil$ (Gabriel Lamé, 1844)
- Can be done faster?

Comparing Functions

- Compare the two running time functions in ms of algorithms F & G:
  - $f(n) = n^2 + 1$
  - $g(n) = n \log_{10} n + 1000\, n + 9999$
- Value of the functions for $n = 100$:
  - $f(100) = 10{,}001 \cong 10$ sec. vs $g(100) = 110{,}199 \cong 110$ sec.
  - So can we say that F is better than G?
- What about $n = 10{,}000$?
  - $f(10{,}000) = 100{,}100{,}101 \cong 100{,}000$ sec. $\cong 27.7$ hours
  - $g(10{,}000) = 10{,}049{,}999 \cong 10{,}000$ sec. $\cong 2.7$ hours
- We need to focus in the dominant term (rate of growth)

Asymptotic Notations

Mathematical Tools To Compare the Efficiency of Different Algorithms

Motivation

- A mathematical tool (framework) used for describing algorithm's efficiency, considering the two abstractions we saw:
  - Constant factors are not important.
  - The most dominating term matters (growth rate)
- Fairly theoretical, mostly for understanding/gaining insights in analysis of algorithms
  - Formal definitions (big-Oh, big-Omega, ...)
- Note, in real world, constant factors and non-dominating terms could matter (sometimes seriously)
- Let's begin the theoretical journey!

O-Notation (Big-Oh)

- We say (define):
  - A function $f(n)$ is in big-Oh of $g(n)$ (denoted $O(g(n))$) iff (if and only if) there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all n that's greater than or equal to n_0.
- In other words,
  - $O(g(n))$ is a set of all functions (call any such function $x(n)$, to avoid confusion with $f(n)$) where there exist positive constants c and n_0 such that $0 \leq x(n) \leq cg(n)$ for all n that's greater than or equal to n_0.
  - And $f(n)$ is a member of the set $O(g(n))$ (is in the set).
  - Proper notation would be: $f(n) \in O(g(n))$, but we abuse $=$, and write $f(n) = O(g(n))$ most of the time.
- What does all this mean?
  - Study the worked examples in the following slides

Big-Oh Notation Proof Examples

Time To Prove Big-Oh Notations Formally

Is $\frac{1}{2}n^2 - 3\text{n} = O(n^2)$?

A function $f(n)$ is in big-Oh of $g(n)$ (denoted $O(g(n))$) iff (if and only if) there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

- If you think it is, and need to prove it, you must find c and n_0 such that
$$0 \le \frac{1}{2}n^2 - 3\text{n} \le cn^2$$
for all $n \ge n_0$
- This is an easy case, after trying a few possible c and n_0 with some intuitions:
  - As long as c is at least $\frac{1}{2}$, the inequality holds for any non-negative n!
  - Thus, we can simply let $c = \frac{1}{2}, n_0 = 6$, which satisfies the definition (the inequality above) of big-Oh.
  - In fact, there are infinitely many valid c and n_0 that can be used for the proof.

Is $100n^2 + 123\text{n} = O(n^2)$?

A function $f(n)$ is in big-Oh of $g(n)$ (denoted $O(g(n))$) iff (if and only if) there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

- If you think it is, and need to prove it, you must find c and n_0 such that
$$0 \le 100n^2 + 123n \le cn^2$$
for all $n \ge n_0$
- This is another easy case, after trying a few possible c and n_0 with some intuitions:
  - You soon realize that c must be greater than 100. Let $c = 101$.
  - Then for what values of n does $100n^2 + 123n \le 101n^2$?
    - Just solve the inequality, and you get $n \ge 123$, which gives us 123 for n_0.

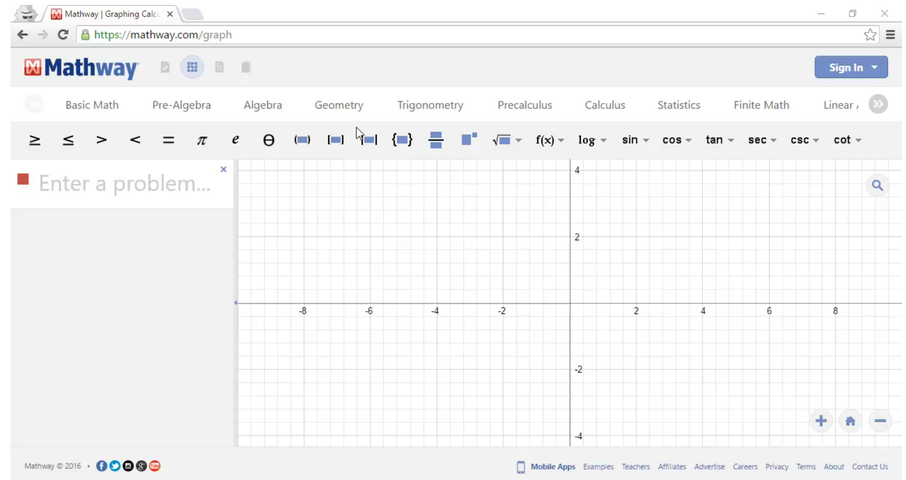
Is $100n^2 + 123\text{n} = O(n^3)$?

- If you think it is, and need to prove it, you must find c and n_0 such that
$$0 \le 100n^2 + 123n \le cn^3$$
for all $n \ge n_0$
- This is another easy case, after trying a few possible c and n_0 with some intuitions:
  - c doesn't have to be very big here, because we have n^3 that grows fast. Let $c = 1$.
  - Then for what values of n is $100n^2 + 123n \le n^3$?
    - No need to solve the inequality precisely. Just divide both sides by n^2, which gives us $n \ge 100 + \frac{123}{n}$, where $100 + \frac{123}{n} \le 223$, and this (223) is our n_0.

Is $\frac{1}{200}n^2 = O(n)$?

- If you think it is, and need to prove it, you must find c and n_0 such that
$$0 \le \frac{1}{200}n^2 \le cn$$
for all $n \ge n_0$
- However, you soon realize that no matter what values of c and n_0 you choose, there'll be always some n that makes $\frac{1}{200}n^2 > cn$
  - Easy, just pick $n = 200c + 1$ for any c. Doesn't matter what n_0 is (in this case). This is called counterexample.
- Therefore, $\frac{1}{200}n^2 = O(n)$ is not true!

Intuition of Big-Oh: Why $\frac{1}{200}n^2 \neq O(n)$?



Lesson From Visualization

Rate of growth of a higher order term (n^2) can't be overcome by a constant factor c multiplied to a lower order term (n), no matter how big c can get.

More Intuitions of Big-Oh

- Lower order term (e.g., $123n$ in $100n^2 + 123n$) can be always ignored by setting n_0 sufficiently large.
  - Highest order term is the "dominating" term.
- Rule of thumb: Pick the highest order term only, drop the constant factor, and that's your best big-Oh notation for a given function. E.g.,
  - $\frac{1}{2}n^2 - 3n$: Pick the highest order term only ($\frac{1}{2}n^2$) and drop the constant factor ($\frac{1}{2}$), which gives $O(n^2)$.
  - $100n^2 + 123n$: Same, pick $100n^2$, drop 100, giving $O(n^2)$.

Bounding Properties of Asymptotic Notations

Useful Intuitions On Asymptotic Notations

Big-Oh Is Upper Bound

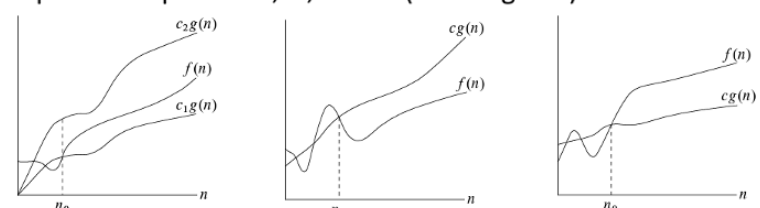
- If $f(n)$ is in $O(n^2)$, then $f(n)$ is also in $O(n^3)$, $O(n^4)$, … .
  - E.g., $f(n) = \frac{1}{200}n^2 + 123n$.
- But, $f(n)$ is not in $O(n)$ for the above example!
- The function in $O(\,)$ for a given function $f(n)$ is an "asymptotic" upper bound.
- There are many upper bounds, and we'd prefer to find the "tight" upper bound.
- In the above example, it's $O(n^2)$.
  - Because it's tighter than all other $O(n^k)$ for any $k > 2$.
  - And also because $f(n)$ cannot be in $O(n)$.

Big-Omega Notation: Asymptotic Lower Bound

- Not as frequently mentioned as big-Oh, but still important notation for asymptotic lower bound
  - E.g., algorithm A takes at least $\Omega(n)$ time for input size n
- Exactly the same patterned definition, just the different inequality:
  - A function $f(n)$ is in $\Omega(g(n))$ iff there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all n that's greater than or equal to n_0.
- Exercise: Prove that $n^4 + 12n^3 - 34n^2 + 56n + 78$
  - Is in $\Omega(n^4)$, in $\Omega(n^3)$, in $\Omega(n^2)$, in $\Omega(n^1)$, and in $\Omega(1)$.
  - But not in $\Omega(n^5)$.

Theta Notation: Asymptotic Tight Bound

- If $f(n) = O(g(n))$ and also $f(n) = \Omega(g(n))$, we say $f(n) = \Theta(g(n))$. In other words,
  - A function $f(n)$ is in $\Theta(g(n))$ iff there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all n that's greater than or equal to n_0.
- Graphic examples of Θ, O, and Ω (CLRS Fig. 3.1)



$f(n) = \Theta(g(n))$ $f(n) = O(g(n))$ $f(n) = \Omega(g(n))$

Small-oh/omega Notations: Non-tight Bounds

- $2n^2 = O(n^2)$ is asymptotically tight, but $2n = O(n^2)$ is not tight.
- Use o-notation to denote an upper bound that is not asymptotically tight. Define:
  - A function $f(n)$ is in $o(g(n))$ iff for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq f(\mathrm{n}) < \mathrm{c}g(n)$ for all n that's greater than or equal to n_0.
- E.g.: $2n = o(n^2)$, but $2n^2 \neq o(n^2)$.
- Similar definition for $\mathrm{f(n)} = \omega(g(n))$.

Useful Relationships of Asymptotic Notations

Different Asymptotic Notations Are Related

Properties of Asymptotic Notations

- $f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.
- Transitivity
  - $f(n) = X(g(n))$ and $g(n) = X(h(n))$ implies $f(n) = X(h(n))$. (X can be any of $O, \Omega, \Theta, o, \omega$.
- Reflexivity: $f(n) = X(f(n))$ for $X = O, \Omega, \Theta$.
- Symmetry: $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$.
- Transpose symmetry:
  - $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$.
  - $f(n) = o(g(n))$ iff $g(n) = \omega(f(n))$.

Analogy with Numeric Inequalities

- $f(n) = O(g(n))$ is like $a \leq b$.
- $f(n) = \Omega(g(n))$ is like $a \geq b$.
- $f(n) = \Theta(g(n))$ is like $a = b$.
- $f(n) = o(g(n))$ is like $a < b$.
- $f(n) = \omega(g(n))$ is like $a > b$.

Asymptotic Notations and Algorithm Analysis

How Asymptotic Notations Are Used In Algorithm Analysis

Lesson Objectives

- Analyze simple algorithms for the asymptotic notations of their running time performance (time complexities)
- Prove formally that the derived asymptotic notations are correct.

Selection Sort Analysis

- Selection sort works in two steps:
  - Find the index of the minimum of array values
  - Place the minimum in the correct position
    - And repeat this process on the smaller array
- Analysis of selection sort:
  - Time for FIND_MIN_INDEX: $T_1(n) = \Theta(n)$
  - Time for INS_SORT: $T_2(n) = T_1(n) + c + T_2(n-1)$
    - This is a recurrence!
    - Solving recurrence, get $T_2(n) = \Theta(n^2)$
- The same is derived with more detail in the following slides

Selection Sort

- Visit http://visualgo.net/sorting
  - Click SEL, then Sort to see how selection sort algorithm works
- Note sub-problem decomposition:
  - For array A[i,n]:
    - Find index of minimum of array values in index i to n. Call it j.
    - Swap A[i] and A[j].
    - Repeat the above for subarray A[i+1,n].
    - Of course if i=n, nothing to do, so just stop.

FindMinIndex(A, s)

- Input: Array A[1,n], starting index s.
- Output: Index j (between s and n) of the minimum of A[s,n]
- E.g., if A={55,88,33,44,99} and s=2, return 3 (the index of 33, which is the minimum of A[2,5])
- Steps:
  - indexOfMinimumSoFar = s
  - for i = s to n do
    - if A[i] < A[indexOfMinimumSoFar]:
      - indexOfMinimumSoFar = i
  - return indexOfMinimumSoFar

SelSort(A)

- Let m=FindMinIndex(A,1). Then swap A[1] and A[m].
- Let m=FindMinIndex(A,2). Then swap A[2] and A[m].
- ...
- Steps:
  - for s = 1 to n:
    - m = FindMinIndex(A,s)
    - tmp = A[m]; A[m] = A[s]; A[s] = tmp;   // Swap A[s] and A[m]

SelSort(A) Running Time Analysis

- Count number of operations executed
  - for s = 1 to n: // n times of 1 comparison and 1 increment, plus:
    - m = FindMinIndex(A, s)   // # FindMin(A,1) + # FindMin(A,2) + ... (not n times)
    - tmp = A[m]; A[m] = A[s]; A[s] = tmp;   // 3
- Therefore, T_SelSort(n) = 5n + $\sum_{s=1}^{n} \# \, FindMinIndex(A, s)$
- # FindMinIndex(A,s):
  - indexOfMinimumSoFar = s   // 1
  - for i = s to n: // n-s+1 times of1 comparison and 1 increment, plus:
    - if A[i] < A[indexOfMinimumSoFar]:   // 1
      - indexOfMinimumSoFar = i   // 0 ~ 1 (1 if condition is true, 0 otherwise)
  - return indexOfMinimumSoFar   // 1
- # FindMinIndex(A,s) = 2(n-s+1)+2+(0~1)*(n-s+1)
  $\geq 2(n-s) + c_1, \leq 3(n-s) + c_2 \Rightarrow \Theta(n-s)$

Final T_SelSort(n): $\Theta(n^2)$

- T_SelSort(n) = 5n + $\sum_{s=1}^{n} \# \, FindMinIndex(A, s)$
  $\geq 5n + 2\{(n-1) + (n-2) + \cdots + 1\} + c_1 n$
  $= 2\frac{n(n-1)}{2} + 5\text{n} + \text{c}_1\text{n}$
  $= n^2 + c_2 n$
- T_SelSort(n) = 5n + $\sum_{s=1}^{n} \# \, FindMinIndex(A, s)$
  $\leq 5n + 3\{(n-1) + (n-2) + \cdots + 1\} + c_1 n$
  $= 3\frac{n(n-1)}{2} + 5\text{n} + \text{c}_1\text{n}$
  $= \frac{3}{2}n^2 + c_3 n$
- These 2 inequalities fit the definition of $\Theta(n^2)$!