Data Structures with C++ : CS189

Lecture 4-2: Big O

Big O Purpose

- A programmer has to get a task done
- A computer scientist needs to find the best solution and prove it is the best
- The idea of "Big O" is we need a scientific way to talk about speed so we can prove which solution is best

Big O Definition

- Picture a graph for an algorithm with the y axis being time and the x axis being input size
  - To count n pencils, I touch n pencils
  - To count n+1 pencils, I touch n+1
  - Linear graph
- That is "Big O of n" or "O(n)"
  - "This algorithm finishes in no more than linear time"
  - "Omega of n" changes that to "no less than"
  - "Theta of n" changes that to "exactly"
  - The reason O is the most used over Theta is we don't actually care about the precise speed. Just the worst case.

O(n^2)

- Everyone in class has to shake everyone's hand
  - Note: example not recommended during plague
- One more person appears. How many more handshakes is that?
  - For pencils one more pencil was one more action
  - Here the new person must shake n peoples' hands
  - "For each person, shake each person"
- Technically "(n^2 - n) / 2" or whatever
- Don't want the exact curve, just the shape.
  - n^2 is the worst there, so it dominates at n = 10,000,000

O(log n)

- This curve decelerates, just like how n-squared accelerated
- It takes me a minute to eat half the cookies
  - 3: Nom, 1, Nom
  - 4: Nom, 2, Nom, 1, Nom
  - 5: Nom, 2, Nom, 1, Nom
  - 6: Nom, 3, Nom, 1, Nom
  - 7: Nom, 3, Nom, 1, Nom
  - 8: Nom, 4, Nom, 2, Nom, 1, Nom
- Time doesn't go up until the input doubles
  - In this example, from 4 to 8.

Big O Guidelines

- If any part of an algorithm is n^2, it is O(n^2)
- A loop is O(n)
  - Do something to everything
- A loop in a loop is O(n^2)
  - For each thing, do something to everything
- Recursion can be exponential O(2^n) if you aren't careful
  - For each thing… run the program on each thing
- The cost of anything not related to n is irrelevant
  - Branches, function calls, variables

Vector's Big O's

- If you don't have all the methods *and* at the right speed, you don't count as a vector
- At: O(1) - No matter how much data is there, jumping to an index is free
- PushBack: O(1) - Always know the last position
- PopFront: O(n) - To take away the first element you'd have to shift n things left. This is so much slower than List's that STL doesn't even offer it

Examples

1. I tell all of you to go buy me a pizza.
    a. Pizzas bought: O(n)
2. I tell you all to buy a pizza for each student in the class.
    a. Pizzas bought: O(n^2)
3. I tell you to find the heaviest pizza.
    a. Weigh a pizza: O(n)
4. I tell you to find the two closest pizzas.
    a. Measure distance of each pizza to each pizza: O(n^2) (We can beat this in week 13)

Best / Worst Cases

- We have O, Omega, and Theta for "no worse", "no better", and "exactly"
  - Pure computer science level
- In real life we have concrete concerns
  - An algorithm is O(1) 99% of the time and O(n^2) 1 % of time. Pick that or a 100% O(n)?
    - Medical equipment, A. Webpage, B.
- This means we may actually care about constants
  - O(n) vs O(10n). Same shape, but in real life, very different

Time and Big O's

- If I am running an algo on 1,000 records twice a week, Big O matters less
- If I am running 1,000 simulations a second on a chip meant for spacecraft, Big O isn't enough
- "Benchmarking" is almost the opposite of Big O
  - Measuring actual speed of algos on actual set of data
- "Amortized" (a-more-tized) is taking Big O over time
  - After running 10 billion times, which is faster?

Go to Canvas Quiz

Make sure to focus on what the action is in each. Just like how when looking at code you ignore extra statements, in Big O you focus on what is being done.

P vs NP

- The following is pure computer science
  - Intro to high level algo class
  - May Goldstein forgive me if I misspeak here
- If I have a problem I can solve with a Big O that's a polynomial (n, n^2, 1, log), it's "easy"
- If I can't, but if I'm given an answer I can check if it is right in polynomial time, it is "hard"
- Polynomial vs Non-Polynomial

NP Complete

- All NP problems can be transformed in to any other NP problem
  - Example next page
- It follows that if you can solve any NP problem, you will solve them all
  - Meaning that P == NP
  - There is a million dollar prize for doing this
  - It would change computing
- Not only can we not prove that, we can't prove that we can't prove that

NP Complete Example from CSULB 528

- Start with a set of points and connecting lines
- Hamiltonian Path: Is there a path that touches every point exactly once?
- Max Clique: Are all the points connected?
- If there is a Hamiltonian Path, then there is a Max Clique
- NP Because every point has to check if every point is connected to every point that is connected to every point which is connected to every point that is

End

If you go to CSULB, take any class with Goldstein, Foss, or Monge teaching.