CS 209
Data Structures and Mathematical Foundations

02 / 21 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions/Comments?
- Reminder of log and lg
- Recap of all our algorithm analysis so far
- Write findmax and carefully count up work to determine runtime
- List of functions in increasing order
- Graphs of some of those functions and charts with actual calculations of times

log

- Log function and relation to exponential function
- Notation: lg = log base 2
- lg of a number means = what power of 2 produces that number

- Example: If we know $2^{10} = 1024$, then that means lg(1024) = 10
- lg(1024) is the power of 2 that results in 1024

- Occurs in runtime algorithm analysis when we continually cut the size of the list in half (e.g. like in binary search).

Algorithm Analysis

- Some takeaways
  - Log function grows very slowly, that is, as n increases lg n increases slowly
  - Exponential grows very fast, that is, as n increases 2 to the n increases very quickly
  - The slower growing functions are more desireable runtimes for algorithms, for large values of n (large inputs)

Algorithm Analysis

- Recap on runtime analysis we have done so far
- Linear search: best case is constant time, worst case is linear time (aka n time)
- Binary search: best case is constant time, worst case is log(n) time
- SelectionSort: there was no difference between best and worst cases --- all cases take quadratic time (aka n^2 time)
- InsertionSort: Best Case was linear (aka n time), and worst case was quadratic (aka n^2 time)
- Space analysis:
  - We noticed that MergeSort had n extra space required. All the other algorithms we looked at had only constant extra space required.

FindMax

- Let’s write the code to do it
- Let’s more carefully count up the work than we have been doing

Algorithm Analysis

- Some common functions (in increasing order) used in analysis are
  - constant functions (e.g. $f(n) = 10$)
  - logarithmic functions (e.g. $f(n) = \log(20n)$)
  - log squared (e.g. $f(n) = \log^2(7n)$)
  - linear functions (e.g. $f(n) = 3n - 9$)
  - N log N (e.g. $f(n) = 2n \log n$)
  - quadratic functions (e.g. $f(n) = 5n^2 + 3n$)
  - cubic functions (e.g. $f(n) = 3n^3 - 17n^2 + (4/7)n$)
  - exponential functions (e.g. $f(n) = 5^n$)
  - factorial functions (e.g. $f(n) = n!$)

Algorithm Analysis

- Let's look at the tables with examples of actual times for certain running times given large inputs
- The time complexity of an algorithm is much more important than processor speed (for large enough inputs) even though processor speeds get faster year after year

Algorithm Analysis

- Growth rates of functions are different than being able to say one function is less than another
  - e.g. $200*n^2 + 100$ is greater than $0.002*n^3$ for many low n values but as n increases above some value, $0.002*n^3$ will always be bigger
  - For low values of n, we would actually prefer the cubic runtime algorithm over the quadratic
  - **But for large enough values of n, the quadratic runtime algorithm will be more efficient (run in less time)**
- The constant being multiplied by the dominant term is ignored when describing the runtime, but for low enough values of n, we may prefer a less efficient runtime algorithm

Algorithm Analysis

- Suppose we have an algorithm A that runs in 20*n milliseconds and and algorithm B that runs in n^2 milliseconds.
- Which algorithm will you prefer for LARGE n?