CS161 - Big O

David Kauchak

We need a way to talk about that computational cost of an algorithm that focuses on the essential parts and ignores details that are not relavent and that is somewhat agnostic to the underlying hardware.

How would you answer the question, what is the running time of algorithm x?

We saw some of this last time in our examination of Insertion-Sort and Merge-Sort.

- Asymptotic notation:
  - Precisely calculating the actual steps is tedious and not generally useful
  - Different operations take different amounts of time. Even from run to run, things such as caching, etc. will complicate things
  - Want to identify categories of algorithmic runtimes
  - Compare different algorithms

  $f_1(n)$ takes n^2 steps
  $f_2(n)$ takes $3n + 100$ steps
  $f_3(n)$ takes $2n + 1$ steps

  Which algorithm is better? Is the difference between f_2 and f_3 important/significant?

- Runtime examples

	n	$n \log n$	n^2	n^3	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 18 min	10^{25} years
$n = 100$	< 1 sec	< 1 sec	1 sec	1s	10^{17} years	very long
$n = 1000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long

(adapted from [2], Table 2.1, pg. 34)

- $O(g(n))$ is the set of functions:

  $O(g(n)) = \{f(n) :$ there exists positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$

- $\Omega(g(n))$ is the set of functions:

  $\Omega(g(n)) = \{f(n) :$ there exists positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$

- Θ-notation

  $\Theta(g(n))$ is the set of functions:

  $\Theta(g(n)) = \{f(n) :$ there exists positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

  Note Θ implies both an upper and lower bound

- Picture of asymptotic bound (e.g. CLRS pg. 43)

- Proving bounds - Find constants c and n_0 that satisfy the inequalities

  Show that $5n^2 - 15n + 100$ is $\theta(n^2)$

  First, show $O(n^2)$, i.e. that there exists positive constants c and n_0 such that $5n^2 - 15n + 100 \leq cn^2$ for all $n > n_0$

$$\begin{aligned} cn^2 &\geq 5n^2 - 15n + 100 \\ c &\geq 5 - 15/n + 100/n^2 \end{aligned}$$

  We can ignore the $-15/n$ term, since it is always negative. If we let $n_0 = 1$ and $c = 5 + 100 = 105$ (or anything greater than 105), then

for all $n \geq n_0$ the above inequality is satisfied since $100/n^2$ only get smaller as n increases.

Second, show $\Omega(n^2)$, i.e. that there exists positive constance c and n_0 such that $5n^2 - 15n + 100 \geq cn^2$ of all $n > n_0$

$$c \leq 5 - 15/n + 100/n^2$$

Like we did above, we can ignore the $100/n^2$ term since it is always positive. If we let $n_0 = 4$ and $c = 5 - 15/4 = 1.25$ (or anything less than 1.25), then the above inequality is satisfied since $15/n$ is always decreasing as n increases.

- Proving that bounds don't hold

  $5n^2 \neq \theta(n)$

  Only have to show that one of the conditions doesn't hold (i.e. Ω or O).

  Suppose some c and n_0 exists such that

  $$cn \geq 5n^2$$

  for all $n \geq n_0$, but then $n \leq c/5$ which cannot hold for all n since for any constant c we can always pick an n large enough such that the above is not satisfied.

- Some rules of thumb (adapted from [1], pg. 8):
  - Multiplicative constants can be omitted: $14n^2$ becomes n^2
  - Lower order functions can be omitted: $n+5$ become n and n^2+n become n
  - n^a dominates n^b if $a > b$: for instance, n^2 dominates n
  - a^n dominates b^n if $a > b$: for instance 3^n dominates 2^n
  - Any exponential dominates any polynomial: 3^n dominates n^5
  - Any polynomial dominates any logorithm: n dominates $(\log n)^3$. This also means n^2 dominates $n \log n$.
  - DO NOT omit lower order terms of different variables: $n^2 + m$ does NOT become n^2

- Some examples:
  - $O(1)$ - constant time
    Regardless of the size of the input, there is a fixed amount of work
    * add two 32 bit numbers
    * determine if a number is even or odd
    * sum the first 20 elements of an array
    * delete an element from a doubly linked list
  - $O(\log n)$ - logarithmic
    At each iteration of the algorithm, discard some **proportion** of the input (often half)
  - $O(n)$ - linear
    Do some constant amount of work on each element of the input
    * find an item in a linked list
    * determine the largest element in the array
  - $O(n \log n)$
    Divide and conquer algorithms with a linear amount of work to recombine
    * sort a list of numbers with Merge-Sort
    * FFT
  - $O(n^2)$
    Double nested loops that iterate over the data
    * Insertion-Sort
  - $O(2^n)$
    * Enumerate all possible subsets
    * Traveling salesman using dynamic programming
  - $O(n!)$
    * Enumerate all permutation
    * determinant of a matrix with exansion by minors

Why is it hard to find examples of algorithms in these latter categories?

These notes are adapted from material found in chapter 3 of [4].

References

[1] Sanjoy Dasgupta, Christos Papadimitiou and Umesh Vazirani. 2008. Algorithms. McGraw-Hill Companies, Inc.

[2] Jon Kleinberg and Eva Tardos. 2006. Algorithm Design. Pearson Education, Inc.

[3] http://en.wikipedia.org/wiki/Big_O_notation

[4] Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest and Clifford Stein. 2007. Introduction to Algorithms, 2nd ed. MIT Press.