Data Structures & Programming

Complexity Analysis part2

Golnar Sheikhshab

Analysis of Algorithms

- Experimentally (empirically)
  - Limited to a set of inputs
  - Environmental factors
  - Comparing algorithms is hard
  - Need for full implementation first
  -
- What we'd like
  - Account for all possible inputs
  - Compare algorithms independent of hardware
  - Analyze high-level descriptions of algorihtms

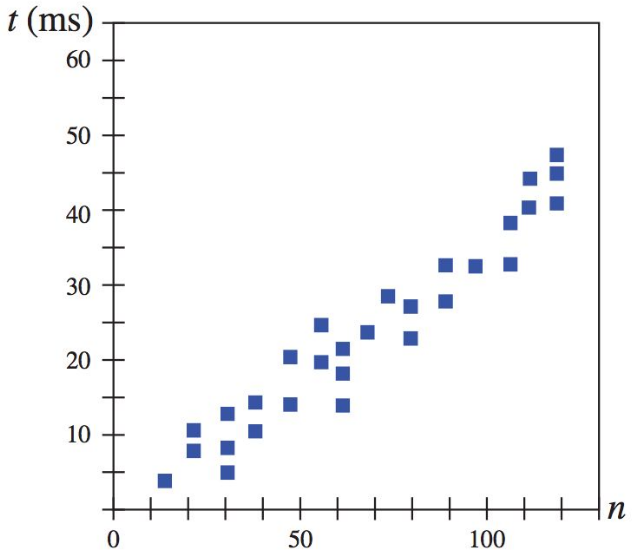


Figure 4.3: Results of an experimental study on the running time of an algorithm. A dot with coordinates (n, t) indicates that on an input of size n, the running time of the algorithm is t milliseconds (ms).

Measurement of goodness

How much time will the algorithm take to complete the job?

How much space would it need?

We care about how the time/space grow with the size of input

Counting primitive operations

Assumption : they take almost the same time to complete

It correlates with the actual running time

primitive operations:

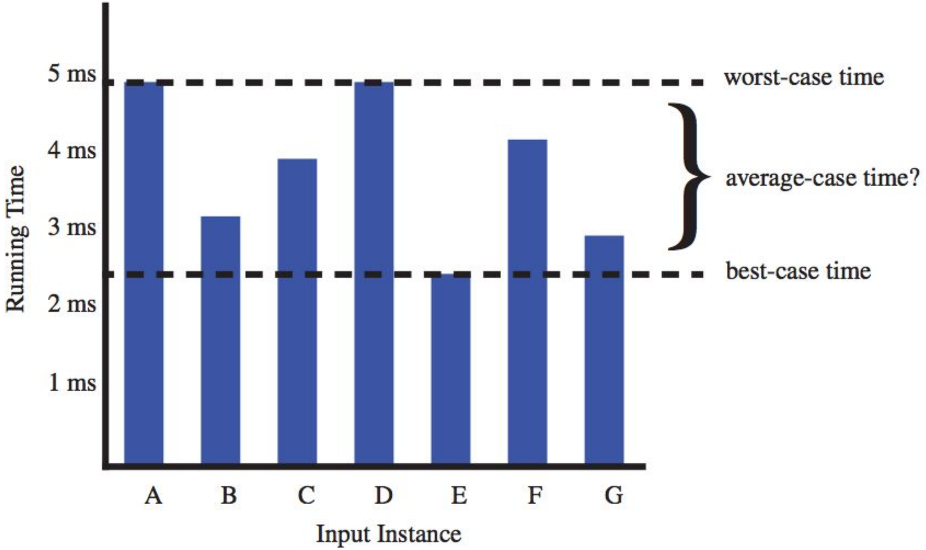
- Assigning a value to a variable
- Calling a function
- Performing an arithmetic operation
- Comparing two numbers
- Indexing into an array
- Following an object reference
- Returning from a function

Best case, Average case, Worst case scenarios

Best case: immaterial

Average case: challenging to analyze

Worst case: simple, important, default



Asymptotic notation

We care about growth of time/space consumption

The big-Oh notation

$f(n) \in O(g(n))$ if there exist c and n0 such that

$$f(n) \leq cg(n), \quad \text{for} \quad n \geq n_0$$

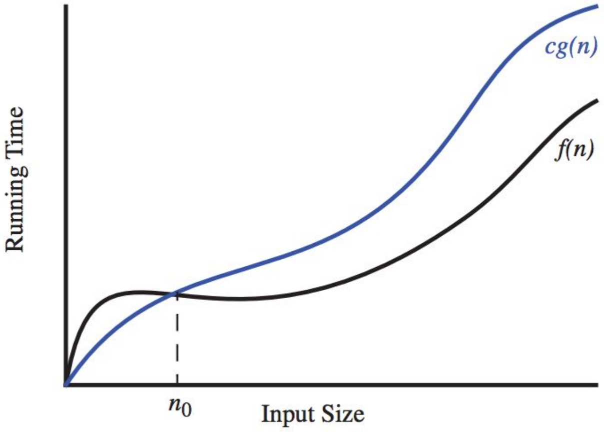


Figure 4.5: The "big-Oh" notation. The function $f(n)$ is $O(g(n))$, since $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

Examples of big-oh

$5n^4 + 3n^3 + 2n^2 + 4n + 1 \text{ is } O(n^4)$

$5n^2 + 3n\log n + 2n + 5 \text{ is } O(n^2)$

$3\log n + 2 \text{ is } O(\log n)$

$2^{n+2} \text{ is } O(2^n)$

$2n + 100\log n \text{ is } O(n)$

Big-Omega and Big-Theta

$$f(n) \text{ is } \Omega(g(n)) \text{ if } g(n) \text{ is } O(f(n))$$

That is if for some real c >0 and integer n0 > 0 we have

$$f(n) \geq cg(n), \quad \text{for} \quad n \geq n_0$$

$$f(n) \text{ is } \Theta(g(n)) \text{ if } f(n) \text{ is } O(g(n)) \text{ and } f(n) \text{ is } \Omega(g(n))$$

That is if for some real c' >0 and c'' >0 and some integer n0 > 0 we have

$$c'g(n) \leq f(n) \leq c''g(n), \quad \text{for} \quad n \geq n_0$$

Asymptotically better means

Bigger examples are tractable

Running Time (μs)	***Maximum Problem Size (n)***		
	1 second	***1 minute***	***1 hour***
$400n$	2,500	150,000	9,000,000
$2n^2$	707	5,477	42,426
2^n	19	25	31

Table 4.3: Maximum size of a problem that can be solved in 1 second, 1 minute, and 1 hour, for various running times measured in microseconds.

Asymptotically better means

Bigger examples are tractable

Running Time	*New Maximum Problem Size*
$400n$	$256m$
$2n^2$	$16m$
2^n	$m+8$

Table 4.4: Increase in the maximum size of a problem that can be solved in a fixed amount of time by using a computer that is 256 times faster than the previous one. Each entry is a function of m, the previous maximum problem size.

Some words of caution

In asymptotic analysis we are hiding constants and slower-growing terms.

For example $10^{100}n$ is $O(n)$ but its constant (one googole) ” is believed by many astronomers to be an upper bound on the number of atoms in the observable universe. So we are unlikely to ever have a real-world problem that has this number as its input size.

Polynomial time complexity could be acceptable given the context and the worst case size of n.

Exponential time complexity is never considered efficient.

Reading material

Section 4.2