Analysis of Algorithms

Neerja Mhaskar

Dept. of Computing and Software, McMaster University, Canada

Acknowledgments: Material mainly based on the textbook Algorithms by Robert Sedgewick and Kevin Wayne (Chapters 1.4) Prof. Janicki's course slides

Measuring the running time of a Program - I

- The running time of a program depends on factors such as:
  - the input to the program,
  - the quality of code generated by the compiler used to create the object program,
  - the nature and speed of the instructions on the machine used to execute the program, and
  - the time complexity of the algorithm underlying the program.
- The fact that running time depends on the input tells us that the running time of a program should be defined as a function of the input.
- Often, the running time depends not on the exact input but only on the size of the input.

Measuring the running time of a Program - II

- It is customary, then, to talk of $T(n)$, the running time of a program, on inputs of size n. For example, some program may have a running time $T(n) = cn^2$, where c is a constant.
- The units of $T(n)$ will be left unspecified, but we can think of $T(n)$ as being the number of instructions executed on an idealized computer.
- For many programs, the running time is really a function of the particular input, and not just of the input size.
- In that case we define $T(n)$ to be the worst case running time, that is, the maximum, over all inputs of size n, of the running time on that input.

Measuring the running time of a Program - III

- We also consider $T_{avg}(n)$, the average, over all inputs of size n, of the running time on that input.
- While $T_{avg}(n)$ appears a fairer measure, it is often fallacious to assume that all inputs are equally likely.
- In practice, the average running time is often much harder to determine than the worst-case running time, both because the analysis becomes mathematically intractable and because the notion of average input frequently has no obvious meaning.
- Thus, we shall use worst-case running time as the principal measure of time complexity, although we shall mention average-case complexity wherever we can do so meaningfully.

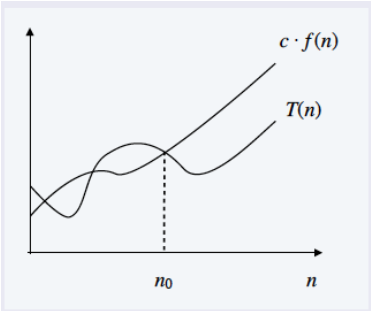
Cost of Basic Operations

Most primitive operations take constant time. Below is the table giving the running time of some of the basic operations used in programs.

operation	example	nanoseconds †
variable declaration	`int a`	c_1
assignment statement	`a = b`	c_2
integer compare	`a < b`	c_3
array element access	`a[i]`	c_4
array length	`a.length`	c_5
1D array allocation	`new int[N]`	$c_6 N$
2D array allocation	`new int[N][N]`	$c_7 N^2$

Big-Oh Notation

Big-Oh Notation: $T(n) \in O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq T(n) \leq c \cdot f(n)$ for all $n \geq n_0$. Note that $O(f(n))$ is a set of functions.



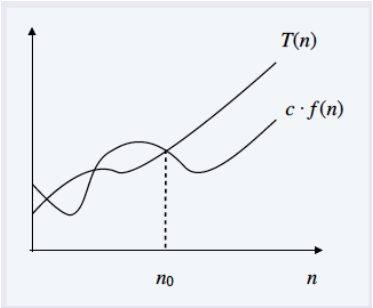
Example

$T(n) = 32n^2 + 17n + 1$, then

- $T(n) \in O(n^2) \longrightarrow$ chose $c = 50, n_0 = 1$
- $T(n) \in O(n^3)$
- $T(n) \notin O(n)$ and $T(n) \notin O(n \log n)$

Big-Omega Notation

Big-Omega Definition: $T(n) \in \Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$. Note that $\Omega(f(n))$ is a set of functions.



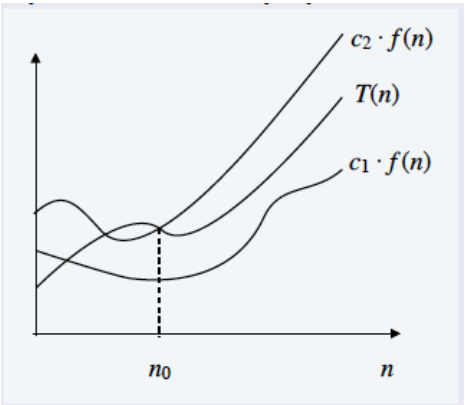
Example

$T(n) = 32n^2 + 17n + 1$, then

- $T(n) \in \Omega(n^2) \longrightarrow$ chose $c = 32, n_0 = 1$
- $T(n) \in \Omega(n)$
- $T(n) \notin \Omega(n^3)$ and $T(n) \notin \Omega(n^3 \log n)$

Big-Theta Notation

Big-Theta Notation: $T(n) \in \Theta(f(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$ and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$. Note that $\Theta(f(n))$ is a set of functions.



In short, $T(n) \in \Theta(f(n))$, if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$.

Example

$T(n) = 32n^2 + 17n + 1$, then

- $T(n) \in \Theta(n^2) \longrightarrow$ chose $c_1 = 32, c_2 = 50, n_0 = 1$
- $T(n) \notin \Theta(n)$ and $T(n) \notin \Theta(n^3)$

Notation Abuses

- **Equals sign:** $O(f(n))$ is a set of functions, but computer scientists often write $T(n) = O(f(n))$ instead of $T(n) \in O(f(n))$. Same is the case with Big-Omega and Big-Theta notations.
- **Domain** The domain of $f(n)$ is typically the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- **Nonnegative functions:** When using big-Oh (big-Omega and big-Theta) notation, we assume that the functions involved are (asymptotically) nonnegative.

Useful facts

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty$, then $f(n) \in O(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$, then $f(n) \in \Omega(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \infty$, then $f(n) \in \Theta(g(n))$.

Tilde ($\sim$) Notation (textbook)

In the textbook, if $f(n) \in \Theta(g(n))$ then $f(n) \sim g(n)$.

Formally, if $f(n) \sim g(n)$, if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Big-Oh Notation with multiple variables

Big-Oh Notation with multiple variables:
$T(m,n) \in O(f(m,n))$ if there exist constants $c > 0$ $m_0 > 0$and $n_0 \geq 0$ such that $0 \leq T(m,n) \leq c \cdot f(m,n)$ for all $m \geq m_0$ and $n \geq n_0$.

Example

$T(n) = 32mn^2 + 17mn + 32n^3$, then

- $T(m,n)$ is both in $O(n^3 + mn^2)$ and $O(mn^3)$
- $T(m,n) \notin O(mn^2)$ and $T(m,n) \notin O(n^3)$

Why it matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Commonly encountered Order growth functions

order of growth

description	function
constant	1
logarithmic	$\log N$
linear	N
linearithmic	$N \log N$
quadratic	N^2
cubic	N^3
exponential	2^N

Commonly encountered order-of-growth functions

Order of the growth functions: $constant < logarithmic < linear < linearithmic < quadratic < cubic < \ldots < exponential$

Some helpful results

- $O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(max(f(n), g(n))$
- $O(f(n))O(g(n)) = O(f(n)g(n))$
- **Polynomials:** Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$. with $a_d > 0$, Then $T(n) = \Theta(n^d)$. This is due to the fact that

$$\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d \neq 0, \infty.$$

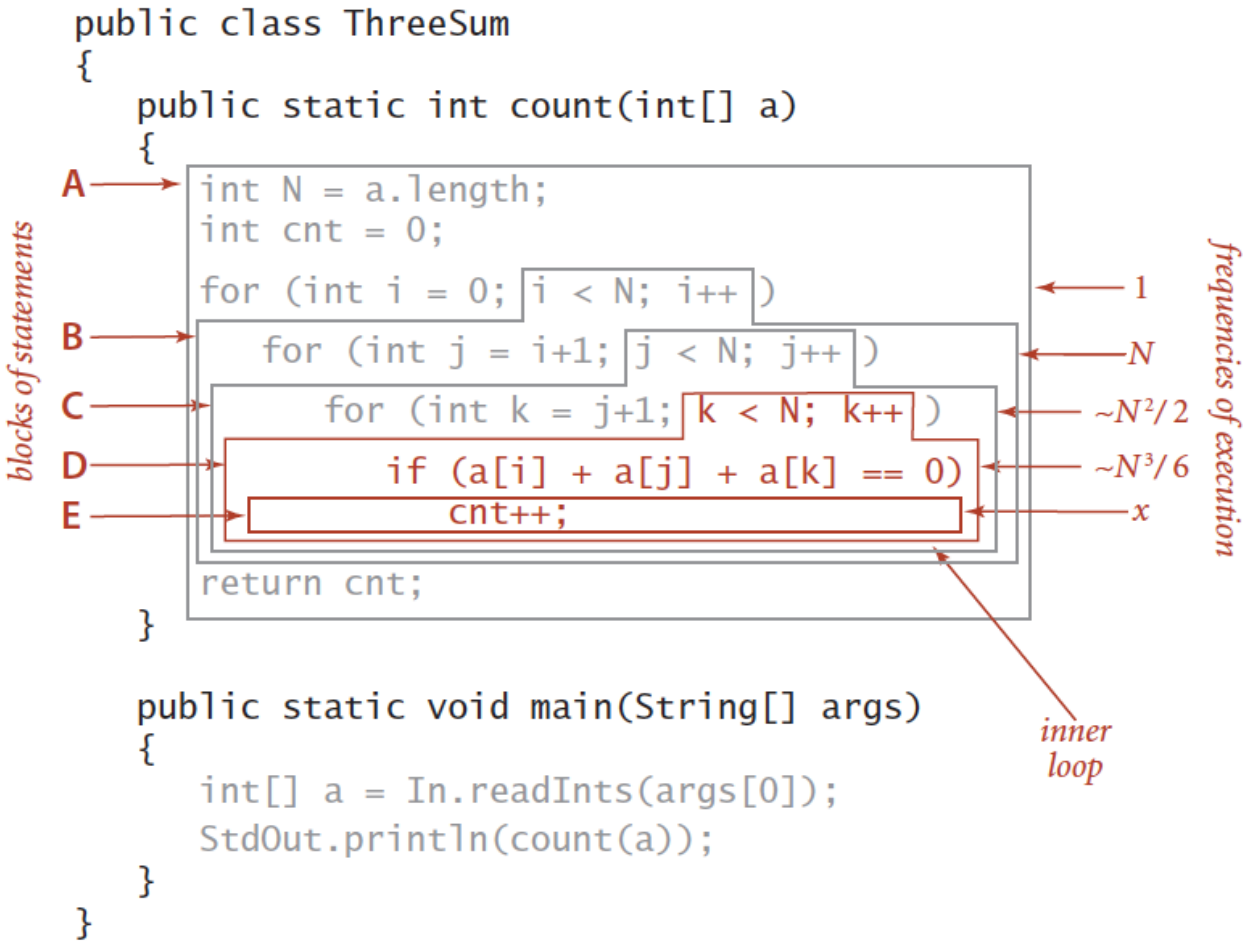
- **Exponentials and polynomials:** For every $r > 1$, and every $d > 0$, $n^d \in O(r^n)$. Since

$$\lim_{n \to \infty} \frac{n^d}{r^n} = 0.$$

Questions

- $O(2^n + n^{10000000000}) =?$
- $O(n^{10000000000}) =?$
- $O(2^n)$? $O(n^k)$, for any k

How to Compute $T(n)$ of an algorithm - I



Anatomy of a program's statement execution frequencies

How to Compute $T(n)$ of an algorithm - II

statement block	time in seconds	frequency	total time
E	t_0	x *(depends on input)*	$t_0 x$
D	t_1	$N^3/6 - N^2/2 + N/3$	$t_1 (N^3/6 - N^2/2 + N/3)$
C	t_2	$N^2/2 - N/2$	$t_2 (N^2/2 - N/2)$
B	t_3	N	$t_3 N$
A	t_4	1	t_4

grand total

$$(t_1/6)\, N^3 + (t_2/2 - t_1/2)\, N^2 + (t_1/3 - t_2/2 + t_3)\, N + t_4 + t_0 x$$

tilde approximation

$$\sim (t_1 / 6)\, N^3 \text{ (assuming } x \text{ is small)}$$

order of growth

$$N^3$$

Analyzing the running time of a program (example)

Useful approximations of functions for analysis of algorithms

description	approximation
harmonic sum	$H_N = 1 + 1/2 + 1/3 + 1/4 + \ldots + 1/N \sim \ln N$
triangular sum	$1 + 2 + 3 + 4 + \ldots + N \sim N^2/2$
geometric sum	$1 + 2 + 4 + 8 + \ldots + N = 2N - 1 \sim 2N$ when $N = 2^n$
Stirling's approximation	$\lg N! = \lg 1 + \lg 2 + \lg 3 + \lg 4 + \ldots + \lg N \sim N \lg N$
binomial coefficients	$\binom{N}{k} \sim N^k/k!$ when k is a small constant
exponential	$(1 - 1/x)^x \sim 1/e$

Useful approximations for the analysis of algorithms

Types of Analyses

- Best case. Lower bound on cost.
  - Determined by "easiest" input.
  - Provides a goal for all inputs.
- Worst case. Upper bound on cost.
  - Determined by "most difficult" input.
  - Provides a guarantee for all inputs.
- Average case. Expected cost for random input.
  - Need a model for "random" input.
  - Provides a way to predict performance.

Caveats

When trying to analyze program performance in detail, many a times you might get inconsistent or misleading results, possibly due to the following:

- Large Constants
- Non-dominant inner loop
- Instruction Time
- System Consideration
- Too close to call
- Strong dependence on inputs
- Multiple problem parameters