Asymptotic Analysis

Module 2: Algorithm Analysis

1 Overview

The running time of an algorithm is analyzed by evaluating the number of instructions in the RAM model of computation. We use asymptotic analysis to find the rate of growth of the running time and express it using $O(), \Omega(), \Theta()$ notation.

Goals:

- Understand the relevance of analysis in practice, as well as its assumptions and limitations
- Understand the definitions of $O(), \Omega(), \Theta()$
- Understand the rate of growth of common functions
- Find the rate of growth of a function
- Compare the order of growth of two arbitrary functions $f(n), g(n)$
- Analyze basic algorithm running times using $O(), \Omega(), \Theta()$

2 Analyzing algorithms

- Why analysis? We want to predict how the algorithm will behave (e.g. running time) on arbitrary inputs, and how it will compare to other algorithms. We want to understand its bottlenecks and the patterns behind its behaviour.
- Theoretical analysis: we break down the algorithm in instructions, and count the number of instructions. Note that the result depends crucially on what exactly we consider to be an *instruction*.
- What is an *instruction* ? An instruction should be primitive (in the sense that it corresponds to a machine instruction on a real computer; e.g. cannot count "sorting" as one operation) and at the same time high-level enough (so that it's independent of the specific processor and platform).
- What exactly is an instruction depends on the computer. We need to define a model of a generic computer. The standard model used in algorithms analysis is:

Random-access machine (RAM) model of computation:

- Contains all the standard instructions available on any computer:
  - Load/Store, Arithmetics (e.g. $+, -, *, /$), Logic (e.g. $>$), jumps
- All instructions take the same amount of time
- Instructions executed sequentially one at a time

- To analyze an algorithm, we assume the RAM model of computation and we count the number of instructions.
- The **running time** of an algorithm is **the number of instructions it executes in the RAM model of computation**.
- RAM model not completely realistic, e.g.
  - main memory not infinite (even though we often imagine it is when we program)
  - not all memory accesses take same time (cache, main memory, disk)
  - not all arithmetic operations take same time (e.g. multiplications expensive)
  - instruction pipelining, etc
- But RAM model gives realistic results in most cases.
- The running time is expressed as a function of the *input size*
  - E.g: running time of a sorting algorithm is expressed as a function of n, the number of elements in the input array.
- Not always possible to come up with a precise expression of running time for a specific input.
  - E.g. Running time of insertion sort on a specific input depends on that input; we can't nail it down without knowing what the specific input is.

We are interested in the smallest and the largest running time for an input of size n:

- **Best-case running time:** The shortest possible running time for *any* input of size n.
- **Worst-case running time:** The longest possible running time for *any* input of size n.

Sometimes we might also be interested in:

- Average-case running time: The average running time over a set of inputs of size n.
  - Must be careful: Average may be different depending on what set we take the average on. Often avarage case analysis assumes any input permutation is equally likely (which is not necessarily realistic).
- Unless otherwise specified, we are interested in the worst-case running time. It gives us a guarantee, the algorithm cannot take longer than this.

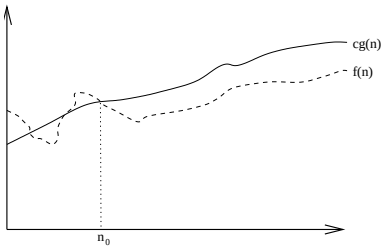
- For some algorithms, worst-case occur fairly often, and average case is often as bad as worst case (but not always!). Some other times worst-case running times occur infrequently, on inputs that are not realistic in practice; in these cases worst-case analysis is not a good indication of the algorithms behavior (some different analysis are possible, such as smoothed analysis).

3 Asymptotic analysis

- Assume we have done a careful analysis counting all instructions and found that:
  Algorithm X best case is $3n + 5$ (linear), and worst case is $2n^2 + 5n + 7$ (quadratic).
- The effort to compute all terms and the constants in front of the terms is not worth it for two reasons:
  1. These constants are not accurate because the RAM model is not completely realistic
  2. For large input the running time is dominated by the higher order term
- For these reasons, when analysing algorithms, we are not interested in all the constants and all the lower order terms, but in the overall *rate of growth* of the running time. This is called *aymptotic analysis*, because we think of $n \to \infty$.
- The rate of growth is expressed using O-notation Ω-notation and Θ-notation. You have seen big-Oh intuitively in Data Structures; now we will now define it more carefully.

3.1 $O()$: Big-O

- Notation: Let $f(n), g(n)$ be non-negative functions (think of them as representing running times of two algorithms)
- We think of "f is $O(g)$" as corresponding to "$f(n) \leq g(n)$"
- We say that f is $O(g(n))$ if, for sufficiently large n, the function $f(n)$ is bounded above by a constant multiple of $g(n)$. That is,



f is $O(g(n))$ if $\exists\ c > 0, n_0 > 0$ such that $f(n) \leq cg(n)\ \forall n \geq n_0$

- Put differently, f is $O(g(n))$ if f is upper bounded by g in the asymptotic sense, i.e. as n goes to ∞. That is, the limit
$$lim_{n\longrightarrow\infty}\frac{f(n)}{g(n)}$$
is either 0 (strict upper bound), or a constant $c > 0$.

- Examples:
  - $\frac{1}{3}n^2+3n$ is $O(n^2)$ because $\frac{1}{3}n^2+3n \leq \frac{1}{3}n^2+3n^2 = (\frac{1}{3}+3)n^2$; holds for $c = 1/3+3 = 3.3$ and $n > 1$.
  - $100n^2$ is $O(n^2)$ because $100n^2 < c \cdot n^2$, pick for e.g. $c = 101$
  - n is $O(n^2)$ because $n < n^2$ so pick for e.g. $c = 1$
  - $20n^2 + 10n \lg n + 5$ is $O(n^3)$ because $20n^2 + 10n \lg n + 5 < 20n^2 + 10n^2 + 5n^2$, so pick for e.g. $c > 20 + 10 + 5$, $c = 40$

- $O(\cdot)$ expresses an upper bound, but not necessarilly tight:
  n is $O(n)$, n is $O(n^2)$, n is $O(n^3)$, ..., n is $O(n^{100})$

- So what? How does a tight bound matter? We can prove that an algorithm has running time $O(n^3)$, and then we analyze more carefully and manage to show that the running time is in fact $O(n^2)$. The first bound is correct, but the second one is better. We want the "tightest" possible bound for the running time. Big-Oh is not sufficient.

Self Quiz 1

Let $f(n) = 3n^2 + 2n + 1$. Mark True or False

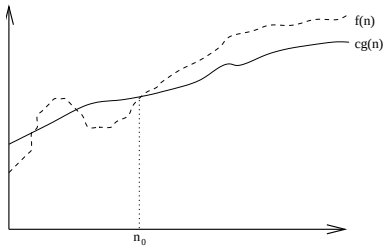
1. $f(n)$ is $O(n^2)$
2. $f(n)$ is $O(n^3)$
3. $f(n)$ is $O(n)$
4. $f(n)$ is $O(n \lg n)$

3.2 $\Omega()$: big-Omega

- Again, let $f(n), g(n)$ be non-negative functions (think of them as representing running times of two algorithms)

- Intuitively we think of "f is $\Omega(g)$ " as corresponding to "$f(n) \geq g(n)$"

- We say that f is $\Omega(g(n))$ if, for sufficiently large n, the function $f(n)$ is bounded below by a constant multiple of $g(n)$. That is,



$f(n)$ is $\Omega(g(n))$ if $\exists\ c > 0, n_0 > 0$ such that $cg(n) \leq f(n)\ \forall n \geq n_0$

- Put differently, f is $\Omega(g(n))$ if f is lower bounded by g in the asymptotic sense, i.e. as n goes to ∞. That is, the limit
$$lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)}$$
is either ∞ (strict lower bound), or a constant $c > 0$.
- Examples:
  - $\frac{1}{3}n^2 + 3n \in \Omega(n^2)$ because $\frac{1}{3}n^2 + 3n \geq cn^2$; true if $c = 1/3$ and $n > 0$.
  - $f(n) = an^2 + bn + c$, $g(n) = n^2$: $f(n)$ is $\Omega(g(n))$ because $an^2 + bn + c > an^2$, true if we pick $c = a$ and $n > 0$
  - $f(n) = 100n^2$, $g(n) = n^2$: $f(n)$ is $\Omega(g(n))$ because $100n^2 > c.n^2$, pick $c = 1$ and $n > 0$
- $\Omega(\cdot)$ gives a lower bound, but not necessarilly tight:
  n is $\Omega(n)$, n^2 is $\Omega(n)$, n^3 is $\Omega(n)$, n^{100} is $\Omega(n)$

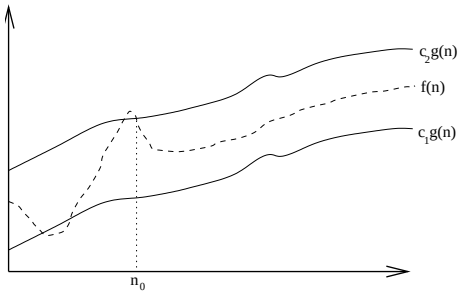
Self Quiz 2

Let $f(n) = 7n^2 + 4n + 3$. Mark with True or False.

1. $f(n)$ is $\Omega(n^3)$
2. $f(n)$ is $\Omega(n)$
3. $f(n)$ is $\Omega(1)$
4. $f(n)$ is $\Omega(n^2)$

3.3 $\Theta()$: Big-Theta

- $\Theta()$ is used to express asymptotically *tight* bounds
- We think of "f is $\Theta(g)$" as corresponding to "$f(n) = g(n)$"
- We say that $g(n)$ is a tight bound for $f(n)$, or $f(n)$ is $\Theta(g(n))$, if $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$. That is, $f(n)$ grows like $g(n)$ to within a constant factor.



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

- Put differently, f is $\Theta(g(n))$ if f and g are equal as n goes to ∞. That is, the limit

$$lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)}$$

  is a constant $c > 0$.

- Examples:
  - $6n \log n + \sqrt{n} \log^2 n$ is $\Theta(n \log n)$
  - n is not $\Theta(n^2)$
  - $6n \lg n + \sqrt{n} \lg^n$ is $\Theta(n \lg n)$
- When analysing the worst or best case running time of an algorithm we aim for asymptotic tight bounds because they characterize the running time of an algorithm precisely up to constant factors.

Self Quiz 3

Let $f(n) = 5n^2 + 3n + 13$. Mark with True or False.

1. $f(n)$ is $\Theta(n^3)$
2. $f(n)$ is $\Theta(n)$
3. $f(n)$ is $\Theta(1)$
4. $f(n)$ is $\Theta(n^2)$

4 Asymptotic analysis and limits

- Another way to think of upper bounds/lower bounds/tight bounds is to calculate the limit:

$$lim_{n\longrightarrow\infty}\frac{f(n)}{g(n)}$$

- Using the definition of O, Ω, Θ it can be shown that :
  - if $lim_{n\longrightarrow\infty}\frac{f(n)}{g(n)} = 0$: then $f = O(g)$ and $f \neq \Theta(g)$ (intuitively $f < g$)
  - if $lim_{n\longrightarrow\infty}\frac{f(n)}{g(n)} = \infty$: then $f = \Omega(g)$ and $f \neq \Theta(g)$ (intuitively $f > g$)
  - if $lim_{n\longrightarrow\infty}\frac{f(n)}{g(n)} = c, c > 0$: then intuitively $f = c \cdot g$ or $f = \Theta(g)$
- Thus comparing f and g in terms of their growth can be summarized as:

  - f is below $g \Leftrightarrow f \in O(g) \Leftrightarrow f \leq g$
  - f is above $g \Leftrightarrow f \in \Omega(g) \Leftrightarrow f \geq g$
  - f is both above and below $g \Leftrightarrow f \in \Theta(g) \Leftrightarrow f = g$

5 Asymptotic analysis, continued

- (It can be shown that) Upper and lower bounds are **symmetrical**: If f is upper-bounded by g then g is lower-bounded by f and the other way around.

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

  Example: n is $O(n^2)$ and n^2 is $\Omega(n)$

- Similarly, it can be shown that if f is a tight bound for g then g is a tight bound for f

$$f \text{ is } \Theta(g) \Longleftrightarrow g \text{ is } \Theta(f)$$

- The correct way to say is that $f(n) \in O(g(n))$ or $f(n)$ is $O(g(n))$. Abusing notation, people often write $f(n) = O(g(n))$.

$$3n^2 + 2n + 10 = O(n^2), n = O(n^2), n^2 = \Omega(n), n\log n = \Omega(n), 2n^2 + 3n = \Theta(n^2)$$

- We use $O(1)$ to denote constant time

- Ignoring constants and concentrating on the rate of growth makes it much easier to analyze algorithms; for e.g. we can easily prove the $O(n^2)$ insertion-sort time bound by saying that both loops run in $O(n)$ time.
- We often use $O(n)$ in equations and recurrences: e.g. $2n^2 + 3n + 1 = 2n^2 + O(n)$, meaning that $2n^2 + 3n + 1 = 2n^2 + f(n)$ where $f(n)$ is some function in $O(n)$.
- Not all functions are asymptotically comparable! There exist functions f, g such that f is not $O(g)$, f is not $\Omega(g)$ and f is not $\Theta(g)$.

6 Asymptotic bounds for some common functions

- **Polynomials:** Let $f(n) = a_0+a_1n+a_2n^2+a_3n^3+...+a_dn^d$, where $a_1, a_2, \ldots, a_d$ are constants and $a_d > 0$. Then $f(n)$ is $\Theta(n^d)$.

  Proof: $lim_{n\longrightarrow\infty}\frac{a_1n+a_2n^2+a_3n^3+...+a_dn^d}{n^d} = a_d > 0$

  Examples:

  - $4n^5 + 3n^2 + 33 = \Theta(n^5)$
- **Logarithms:** $log_an = \Theta(log_bn)$ for every $a, b > 0$ constants. In other words, no need to specify base given it's constant.

  Proof: $lim_{n\longrightarrow\infty}\frac{log_an}{\log_b n} = \frac{1}{\log_b a} > 0$

  Examples:

  - $\log_2 n = \Theta(\log_7 n) = \Theta(\ln n) = \Theta(\log_3 n)$
- **Polynomials and logarithms:**

Any polylog grows slower than any polynomial.
$$\log^a n = O(n^b), \forall a > 0, b > 0$$

  Proof: $lim_{n\longrightarrow\infty}\frac{log^an}{n^b} = 0$

  Examples:

  - $\lg n = O(n)$
  - $\lg^2 n = O(n)$
  - $\lg^3 n = O(n)$
  - $\lg n = O(\sqrt{n})$
  - $\lg^{10} n = O(n^{.01})$

- **Polynomials and exponentials:**

Any polynomial grows slower than any exponential with base $c > 1$.
$$n^b = O(c^n), \forall b > 0, c > 1$$

  Proof: $lim_{n \longrightarrow \infty} \frac{n^b}{c^n} = 0$

  Examples:

  - $n = O(2^n)$
  - $n = O(3^n)$
  - $n^2 = O(2^n)$
  - $n^{100} = O(2^n)$

- **Factorials:** Stirling formula: $\lg n! = \Theta(n \lg n)$. This may come handy in some exercises.

Self Quiz 4

Mark with True or False.

1. n is $O(\lg n)$
2. n is $O(\sqrt{n})$
3. $\lg n$ is $O(\sqrt{n})$
4. $\sqrt{n}$ is $O(n)$
5. $\lg n$ is $O(n)$
6. n^3 is $O(3^n)$

7 Example: Using asymptotic notation to analyze algorithms

Let's analyze Insertion Sort:

- We can say Insertion sort runs in at least constant time: $\Omega(1)$. Duh. That's correct, but it's not a tight bound for the best case, because there is no input of size n on which Insertion sort actually runs in constant time.
- Similarly, we could say that Insertion sort runs in at most $O(n^4)$. That's correct, but it's not a tight bound for the worst case because there is no input of size n on which Insertion sort actually runs in $\Theta(n^4)$ time.

- We can say that Insertion sort **best case** is $\Theta(n)$ time; it's a tight bound for the best case, because there exists an input on which the running time is $\Theta(n)$.
- We'll show that the **worst case** of insertion sort is $\Theta(n^2)$; it's a tight bound because there exists an input on which the running time is $\Theta(n^2)$.
- We can say that the running time of Insertion sort is anywhere in $[\Omega(n), O(n^2)]$.
- Note that we cannot say that the running time of insertion sort is $\Theta(n^2)$

8 Algorithms matter!

Sort 10 million integers on

- 1 GHZ computer (1000 million instructions per second) using $2n^2$ algorithm: This takes $\frac{2\cdot(10^7)^2 \ inst.}{10^9 \ inst. \ per \ second} = 200000$ seconds ≈ 55 hours.
- 100 MHz computer (100 million instructions per second) using $50n \log n$ algorithm: This takes $\frac{50\cdot 10^7 \cdot \log 10^7 \ inst.}{10^8 \ inst. \ per \ second} < \frac{50\cdot 10^7 \cdot 7 \cdot 3}{10^8} = 5 \cdot 7 \cdot 3 = 105$ seconds.

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

Appendix: Review of Log

- $\log_2 n$ is defined as the number of times we can divide n by 2 to get to 1 or less.
- Base 2 logarithm comes up all the time in computer science
- We'll use $\lg n$ to mean $\log_2 n$
- It's important to remember that $\lg n << \sqrt{n} << n$
- Properties:
  - $\lg^k n = (\lg n)^k$
  - $\lg \lg n = \lg(\lg n)$
  - $a^{\log_b c} = c^{\log_b a}$
  - $a^{\log_a b} = b$
  - $\log_a n = \frac{\log_b n}{\log_b a}$
  - $\lg b^n = n \lg b$
  - $\lg xy = \lg x + \lg y$
  - $\log_a b = \frac{1}{\log_b a}$

Appendix: Quiz 1 answers

1. True 2. True 3. False 4. False

Appendix: Quiz 2 answers

1. False 2. True 3. True 4. True

Appendix: Quiz 3 answers

1. False 2. False 3. False 4. True

Appendix: Quiz 4 answers

1. False 2. False 3. True 4. True 5. True 6. True