CM0889 Analysis of Algorithms
Algorithm Analysis

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2020-2

Preliminaries

Conventions

- The number assigned to chapters, examples, exercises, figures, sections, or theorems on these slides correspond to the numbers assigned in the textbook [Skiena 2012].
- The source code examples are in course's repository.

Introduction

Definition

The **computational complexity** of an algorithm is the amount of resources (e.g. time and space) required to execute it.

Definition

The **analysis of algorithms**—term coined by Donald Knuth—is the study of the computational complexity of algorithms.

Introduction

Definition

The **computational complexity** of an algorithm is the amount of resources (e.g. time and space) required to execute it.

Definition

The **analysis of algorithms**—term coined by Donald Knuth—is the study of the computational complexity of algorithms.

Convention

For us 'the complexity of an algorithm' means the time computational complexity of the algorithm.

Introduction

Two abstractions

For the analysis of algorithms we required two abstractions:

(i) Where do the algorithms run? In a theoretical computer, i.e., we are interested in machine-independent algorithms.

Introduction

Two abstractions

For the analysis of algorithms we required two abstractions:

(i) Where do the algorithms run? In a theoretical computer, i.e., we are interested in machine-independent algorithms.

(ii) Which complexity are we interested? We are interested in **asymptotic complexity**, i.e., we are interested in the behaviour of the algorithm for large values of the input.

The RAM Model of Computation

See Skiena's lecture slides: Asymptotic Notation

Best, Worst and Average-Case Complexity

The running time function

If the running time of an algorithm depends of the input then it usually means it depends of the size of the input.

So, we shall use a function

$$T(n) : \mathbb{N} \to \mathbb{R}^{\geq 0}$$

which will denote the running time of an algorithm on inputs of size n.

Best, Worst and Average-Case Complexity

Example

For a sorting algorithm the size of the input is the number of elements to sort.

Best, Worst and Average-Case Complexity

There complexity functions

Given an input of size n we can think in three complexity functions: best-case complexity, worst-case complexity and average-case complexity.

See Skiena's lecture slides: Asymptotic Notation

Asymptotic Notations: Big O

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{O}$ of $\boldsymbol{g(n)}$**, denoted by $O(g(n))$, by

$$O(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notations: Big O

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{O}$ of $\boldsymbol{g(n)}$**, denoted by $O(g(n))$, by

$$O(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \,\}.$$

Notation

Both '$f(n) = O(g(n))$' and '$f(n)$ is $O(g(n))$' mean that $f(n) \in O(g(n))$.

Asymptotic Notations: Big O

Definition (continuation)

If $f(n) \in O(g(n))$ then function $g(n)$ is an upper bound on the growth rate of the function $f(n)$.*

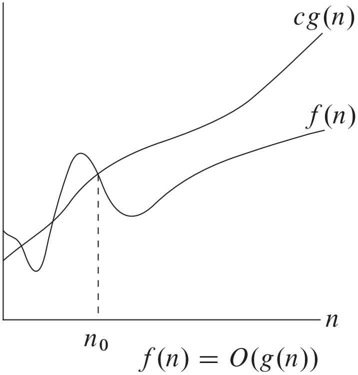


*Figure source: Cormen, Leiserson, Rivest and Stein [2009, Fig. 3.1b].

Asymptotic Notations: Big O

Example

Let $T(n) = 3n^2 - 100n + 6$. The function $T(n)$ is $O(n^2)$ because choosing $n_0 = 1$ and $c = 3$ we have that

$$3n^2 - 100n + 6 \leq cn^2, \quad \text{for all n} \geq \text{n}_0,$$

that is,

$$3n^2 - 100n + 6 \leq 3n^2, \quad \text{for all n} \geq 1.$$

Asymptotic Notations: Big O

Exercise

Let $T(n) = (n+1)^2$. To prove that $T(n) \in O(n^2)$. Hint: Choose $n_0 = 1$ and $c = 4$.

Asymptotic Notations: Big O

Exercise

Let $T(n) = (n+1)^2$. To prove that $T(n) \in O(n^2)$. Hint: Choose $n_0 = 1$ and $c = 4$.

Question

If $T(n) \in O(n^2)$ then $T(n) \in O(n^3)$? What about $O(n^4)$?

Asymptotic Notations: Big O

Example

Let $T(n) = 6n^2$. The function $T(n)$ is not $O(n)$ because

$$6n^2 > cn, \text{when } n > c.$$

Asymptotic Notations: Big O

Theorem

Let d be a natural number and $T(n)$ a polynomial function of degree d, that is,

$$T : \mathbb{N} \to \mathbb{R}$$

$$T(n) = \sum_{i=0}^{d} c_i n^i, \quad \text{with } c_i \in \mathbb{R} \text{ and } c_d \neq 0.$$

If $c_d > 0$ then $T(n) \in O(n^d)$.*

*See, e.g. [Cormen, Leiserson, Rivest and Stein 2009].

Asymptotic Notations: Big O

Theorem

Let d be a natural number and $T(n)$ a polynomial function of degree d, that is,

$$T : \mathbb{N} \to \mathbb{R}$$

$$T(n) = \sum_{i=0}^{d} c_i n^i, \quad \text{with } c_i \in \mathbb{R} \text{ and } c_d \neq 0.$$

If $c_d > 0$ then $T(n) \in O(n^d)$.*

Example

$T(n) = 42n^3 + 1523n^2 + 45728n$ is $O(n^3)$.

*See, e.g. [Cormen, Leiserson, Rivest and Stein 2009].

Asymptotic Notations: Big O

Example

Since any constant is a polynomial of degree 0, any constant function is $O(n^0)$, i.e. $O(1)$.

Remark

Note the missing variable in $O(1)$.*

*We could use the λ-calculus notation, i.e. $O(\lambda n.1)$.

Asymptotic Notations: Big O

Example

Let $T(n) = \lg(7n^2 + 4n)$. To prove that:

(i) $T(n)$ is $O(\lg n)$.

(ii) $T(n)$ is $O(\log_b n)$, for any real number $b > 1$.

Adapted from [Vrajitoru and Knight 2014, Example 3.3.2.(c)].

Asymptotic Notations: Big O

Proof

i) Since

$$\begin{aligned}
\lg(7n^2 + 4n) &< \lg(7n^2 + 4n^2) \\
&= \lg(11n^2) \\
&= \lg 11 + 2\lg n \\
&< \lg n + 2\lg n, \quad \text{for } n \geq 12 \\
&= 3\lg n
\end{aligned}$$

then $T(n)$ is $O(\lg n)$ by choosing $n_0 = 12$ and $c = 3$.

Asymptotic Notations: Big O

Proof (continuation)

(ii) Case $b < 2$

Since $\lg n < \log_b n$ then $T(n)$ is $O(\log_b n)$ because it is $O(\lg n)$.

Asymptotic Notations: Big O

Proof (continuation)

(ii) Case $b > 2$

Because $\log_b n < \lg n$ we can not use the fact that $T(n)$ is $O(\lg n)$ like in the case $b < 2$.

Now, since for $n \geq 12$,

$$\lg(7n^2 + 4n) \leq 3 \lg n \quad \text{and} \quad \lg n = \lg b \cdot \log_b n,$$

then $T(n)$ is $O(\log_b n)$ by choosing $n_0 = 12$ and $c = 3 \cdot \lceil \lg b \rceil$.

Asymptotic Notations: Big Ω

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{\Omega}$ of $\boldsymbol{g(n)}$**, denoted by $\Omega(g(n))$, by

$$\Omega(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \geq cg(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notations: Big Ω

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{\Omega}$ of $\boldsymbol{g(n)}$**, denoted by $\Omega(g(n))$, by

$$\Omega(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \geq cg(n) \text{ for all } n \geq n_0 \,\}.$$

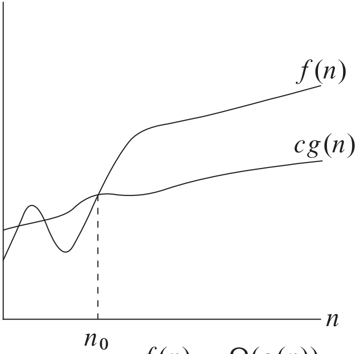
Notation

Both '$f(n) = \Omega(g(n))$' and '$f(n)$ is $\Omega(g(n))$' mean that $f(n) \in \Omega(g(n))$.

Asymptotic Notations: Big Ω

Definition (continuation)

If $f(n) \in \Omega(g(n))$ then function $g(n)$ is a lower bound on the growth rate of the function $f(n)$.*



$f(n) = \Omega(g(n))$

*Figure source: Cormen, Leiserson, Rivest and Stein [2009, Fig. 3.1c].

Asymptotic Notations: Big Θ

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{\Theta}$ of $\boldsymbol{g(n)}$**, denoted by $\Theta(g(n))$, by

$$\Theta(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c_1, c_2 \in \mathbb{R}^+ \text{ and } n_0 \in \mathbb{Z}^+ \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notations: Big Θ

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big $\boldsymbol{\Theta}$ of $\boldsymbol{g(n)}$**, denoted by $\Theta(g(n))$, by

$$\Theta(g(n)) := \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c_1, c_2 \in \mathbb{R}^+ \text{ and } n_0 \in \mathbb{Z}^+ \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \,\}.$$

Notation

Both '$f(n) = \Theta(g(n))$' and '$f(n)$ is $\Theta(g(n))$' mean that $f(n) \in \Theta(g(n))$.

Asymptotic Notations: Big Θ

Definition (continuation)

If $f(n) \in \Theta(g(n))$ then function $g(n)$ is a lower bound and an upper bound on the growth rate of the function $f(n)$.*

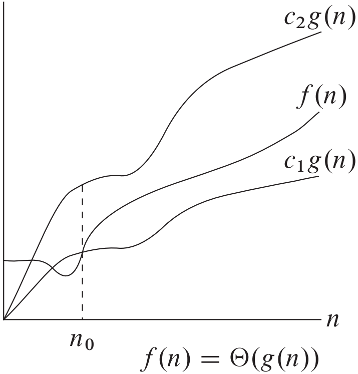


*Figure source: Cormen, Leiserson, Rivest and Stein [2009, Fig. 3.1a].

The Tyranny of Growth Rate

Growing rates of some functions

Each operation takes one nanosecond (10^9 seconds). Figure 2.4 in the textbook.

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

The Tyranny of Growth Rate

Supercomputers

Machines from: www.top500.org (last updated: September 2020)
PetaFLOP (PFLOP): 10^{15} floating-point operations per second

Date	Machine	PFLOPs
2020-06	Fugaku	415.53
2019-06	Summit	148.60
2018-11	Summit	143.50
2018-06	Summit	122.30
2016-06	Sunway TaihuLight	93.01
2013-06	Tianhe-2	33.86
2012-06	Blue Gene/Q	16.32
2011-06	K computer	8.16

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

The problem was proposed in Karp's 21 NP-complete problems [Karp 1972].

The Tyranny of Growth Rate

Improvements on the time complexity of 3-SAT deterministic algorithmic *

$O(1.32793^n)$	Liu [2018]
$O(1.3303^n)$	Makino, Tamaki and Yamamoto [2011, 2013]
$O(1.3334^n)$	Moser and Scheder [2011]
$O(1.439^n)$	Kutzkov and Scheder [2010]
$O(1.465^n)$	Scheder [2008]
$O(1.473^n)$	Brueggemann and Kern [2004]
$O(1.481^n)$	Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning [2002]

(continued on next slide)

*Main sources: Hertli [2011, 2015]. Last updated: July 2020.

The Tyranny of Growth Rate

Improvements on the time complexity of 3-SAT deterministic algorithmic (continuation)

$O(1.497^n)$	Schiermeyer [1996]
$O(1.505^n)$	Kullmann [1999]
$O(1.6181^n)$	Monien and Speckenmeyer [1979, 1985]
$O(2^n)$	Brute-force search

The Tyranny of Growth Rate

3-SAT simulation

Running 3-SAT times on different supercomputers using the faster deterministic algorithm, i.e. $T(1.32793^n)$.

Date	Machine	PFLOPs	$n = 150$	$n = 200$	$n = 400$
2020-06	Fugaku	415.53	7.2 sec	120.2 days	1.4×10^{24} yrs
2019-06	Summit	148.60	20.1 sec	336.1 days	4.0×10^{24} yrs
2018-11	Summit	143.50	20.8 sec	348.1 days	4.1×10^{24} yrs
2018-06	Summit	122.30	24.5 sec	1.1 yrs	4.8×10^{24} yrs
2016-06	Sunway TaihuLight	93.01	32.2 sec	1.5 yrs	6.4×10^{24} yrs
2013-06	Tianhe-2	33.86	1.5 min	4.1 yrs	1.7×10^{25} yrs
2012-06	Blue Gene/Q	16.32	3.1 min	8.4 yrs	3.6×10^{25} yrs
2011-06	K computer	8.16	6.1 min	16.8 yrs	7.3×10^{25} yrs

The Tyranny of Growth Rate

3-SAT simulation

Running 3-SAT times for different deterministic algorithms using the faster supercomputer, i.e. 415.53 PFLOPs.

Complexity	$n = 150$	$n = 200$	$n = 400$
$T(1.32793^n)$	7.2 sec	120.2 days	1.4×10^{24} yrs
$T(1.3303^n)$	9.4 sec	172.0 days	2.9×10^{24} yrs
$T(1.3334^n)$	13.3 sec	273.5 days	7.3×10^{24} yrs
$T(1.439^n)$	14.2 days	3.1×10^{6} yrs	1.3×10^{38} yrs
$T(1.465^n)$	209.1 days	1.1×10^{8} yrs	1.7×10^{4} yrs
$T(2^n)$	1.1×10^{20} yrs	1.3×10^{35} yrs	2.0×10^{95} yrs

Dominance Relations

Example (informal)

See
http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/.

Dominance Relations

Definition

Let f and g two functions. The function f **dominates** the function g, denoted $f \gg g$, iff $g(n)$ becomes insignificant relative to $f(n)$ as n approaches infinity, that is, $\lim_{n\to\infty} g(n)/f(n) = 0$.

Dominance Relations

Definition

Let f and g two functions. The function f **dominates** the function g, denoted $f \gg g$, iff $g(n)$ becomes insignificant relative to $f(n)$ as n approaches infinity, that is, $\lim_{n\to\infty} g(n)/f(n) = 0$.

Example

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1.$$

References

Brueggemann, Tobias and Kern, Walter (2004). An Improved Deterministic Local Search Algorithm for 3-SAT. Theoretical Computer Science 329.1–3, pp. 303–313. DOI: 10.1016/j.tcs.2004.08.002 (cit. on p. 37).

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. and Stein, Clifford [1990] (2009). Introduction to Algorithms. 3rd ed. MIT Press (cit. on pp. 13, 18, 19, 27, 30).

Dantsin, Evgeny, Goerdt, Andreas, Hirsch, Edward A., Kannan, Ravi, Kleinberg, Jon, Papadimitriou, Christos, Raghavan, Prabhakar and Schöning, Uwe (2002). A Deterministic $(2 - 2/(k + 1))^n$ Algorithm for k-SAT Based on Local Search. Theoretical Computer Science 289.1, pp. 69–83. DOI: 10.1016/S0304-3975(01)00174-8 (cit. on p. 37).

Hertli, Timon (2011). 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011). IEEE, pp. 277–284. DOI: 10.1109/FOCS.2011.22 (cit. on p. 37).

— (2015). Improved Exponential Algorithms for SAT and CISP. PhD thesis. ETH Zurich. DOI: 10.3929/ethz-a-010512781 (cit. on p. 37).

References

Karp, Richard M. (1972). Reducibility Among Combinatorial Problems. In: Complexity of Computer Computations. Ed. by Miller, Raymond E. and Thatcher, James W. Plenum Press, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cit. on pp. 33–36).

Kullmann, O. (1999). New Methods for 3-SAT Decision and Worst-Case Analysis. Theoretical Computer Science 223.1–2, pp. 1–72. DOI: 10.1016/S0304-3975(98)00017-6 (cit. on p. 38).

Kutzkov, Konstantin and Scheder, Dominik (2010). Using CSP to Improve Deterministic 3-SAT. CoRR abs/1007.1166. URL: https://arxiv.org/abs/1007.1166 (cit. on p. 37).

Liu, Sixue (2018). Chain, Generalization of Covering Code, and Deterministic Algorithm for k-SAT. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). Ed. by Chatzigiannakis, Ioannis, Kaklamanis, Christos, Marx, Dániel and Sannella, Donald. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs), 88:1–88:13. DOI: 10.4230/LIPIcs.ICALP.2018.88 (cit. on p. 37).

Makino, Kazuhisa, Tamaki, Suguru and Yamamoto, Masaki (2011). Derandomizing HSSW Algorithm for 3-SAT. In: Computing and Combinatorics (COCOON 2011). Ed. by Fu, Bin and Du, Ding-Zhu. Vol. 6842. Lecture Notes in Computer Science. Springer, pp. 1–12. DOI: 10.1007/978-3-642-22685-4_1 (cit. on p. 37).

References

Makino, Kazuhisa, Tamaki, Suguru and Yamamoto, Masaki (2013). Derandomizing HSSW Algorithm for 3-SAT. Algorithmica 67.2, pp. 112–124. DOI: 10.1007/s00453-012-9741-4 (cit. on p. 37).

Monien, B. and Speckenmeyer, E. (1979). 3-Satisfiability is Testable in $O(1.62^r)$ Steps. Tech. rep. 3/1979. Reihe Theoretische Informatik, Universität Gesamthochschule Paderborn (cit. on p. 38).

— (1985). Solving Satisfiability in less than 2^n Steps. Discrete Applied Mathematics 10.3, pp. 287–295. DOI: 10.1016/0166-218X(85)90050-2 (cit. on p. 38).

Moser, Robin A. and Scheder, Dominik (2011). A Full Derandomization of Schöning's k-SAT Algorithm. In: Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing (STOC 2011), pp. 245–252. DOI: 10.1145/1993636.1993670 (cit. on p. 37).

Scheder, Dominik (2008). Guided Search and a Faster Deterministic Algorithm for 3-SAT. In: Proc. of the 8th Latin American Symposium on Theoretical Informatic (LATIN 2008). Ed. by Laber, Eduardo Sany, Bornstein, Claudson, Nogueira, Tito Loana and Faria, Luerbio. Vol. 4957. Lecture Notes in Computer Science. Springer, pp. 60–71. DOI: 10.1007/978-3-540-78773-0_6 (cit. on p. 37).

References

Schiermeyer, Ingo (1996). Pure Literal Look Ahead: An $O(1.497^n)$ 3-Satisfability Algorithm (Extended Abstract). Workshop on the Satisfability Problem, Siena 1996. URL: http://gauss.ececs.uc.edu/franco_files/SAT96/sat-workshop-abstracts.html (cit. on p. 38).

Skiena, Steven S. [1997] (2012). The Algorithm Design Manual. 2nd ed. Corrected printing. Springer. DOI: 10.1007/978-1-84800-070-4 (cit. on p. 2).

Vrajitoru, Dana and Knight, William (2014). Practical Analysis of Algorithms. Springer. DOI: 10.1007/978-3-319-09888-3 (cit. on p. 21).