Algorithm Design and Analysis I

Prologue

Guoqiang Li
School of Computer Science

SHANGHAI JIAO TONG UNIVERSITY

Instructor

Instructor and Teaching Assistants

Guoqiang LI

Instructor and Teaching Assistants

Shanghai Jiao Tong University

Guoqiang LI

- Homepage: https://basics.sjtu.edu.cn/%7Eliguoqiang
- Course page: https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/AI1401/
- Canvas: https://oc.sjtu.edu.cn/courses/69730
- Email: li.g (AT) outlook (DOT) com
- Office: Rm. 1212, Building of Software
- Phone: 3420-4167

Instructor and Teaching Assistants

Guoqiang LI

- Homepage: https://basics.sjtu.edu.cn/%7Eliguoqiang
- Course page: https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/AI1401/
- Canvas: https://oc.sjtu.edu.cn/courses/69730
- Email: li.g (AT) outlook (DOT) com
- Office: Rm. 1212, Building of Software
- Phone: 3420-4167

TA:

Instructor and Teaching Assistants

Guoqiang LI

- Homepage: https://basics.sjtu.edu.cn/%7Eliguoqiang
- Course page: https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/AI1401/
- Canvas: https://oc.sjtu.edu.cn/courses/69730
- Email: li.g (AT) outlook (DOT) com
- Office: Rm. 1212, Building of Software
- Phone: 3420-4167

TA:

- Shuhan FENG: count_von (AT) sjtu (DOT) edu (DOT) cn
- Lianyi WU: edithwuly (AT) 163 (DOT) com

Instructor and Teaching Assistants

SHANGHAI JIAO TONG UNIVERSITY

Guoqiang LI

- Homepage: `https://basics.sjtu.edu.cn/%7Eliguoqiang`
- Course page: `https://basics.sjtu.edu.cn/%7Eliguoqiang/teaching/AI1401/`
- Canvas: https://oc.sjtu.edu.cn/courses/69730
- Email: li.g (AT) outlook (DOT) com
- Office: Rm. 1212, Building of Software
- Phone: 3420-4167

TA:

- Shuhan FENG: count_von (AT) sjtu (DOT) edu (DOT) cn
- Lianyi WU: edithwuly (AT) 163 (DOT) com

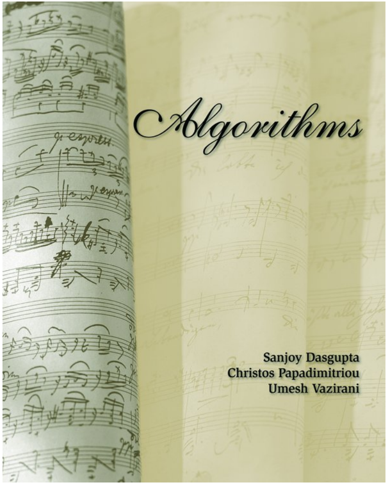
Office hour: Wed. 14:00-17:00 @ SEIEE 3-325

Reference Book

Textbook

Algorithms

- Sanjoy Dasgupta
- San Diego Christos Papadimitriou
- Umesh Vazirani
- McGraw-Hill, 2007.

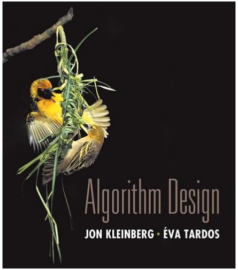


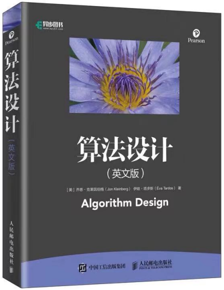
Reference Book

SHANGHAI JIAO TONG UNIVERSITY

Algorithm Design

- Jon Kleinberg, Éva Tardos
- Addison-Wesley, 2005.



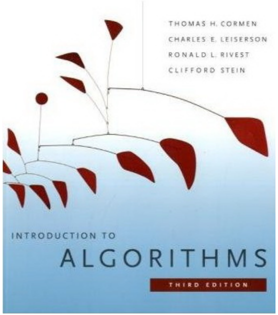


Reference Book

Shanghai Jiao Tong University

Introduction to Algorithms

- Thomas H. Cormen
- Charles E. Leiserson
- Ronald L. Rivest
- Clifford Stein
- The MIT Press (3rd edition), 2009.



Scoring Policy

0% Attendees.

Scoring Policy

0% Attendees.

40% Homework.

- Eight assignments.
- Each one is 5pts.
- Work out individually.
- Each assignment will be evaluated by A, B, C, D, F (Excellent(5), Good(5), Fair(4), Delay(3), Fail(0))

Scoring Policy

0% Attendees.

40% Homework.

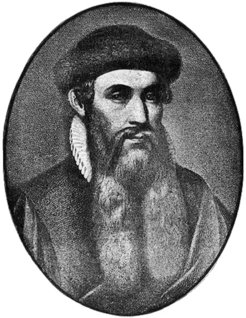
- Eight assignments.
- Each one is 5pts.
- Work out individually.
- Each assignment will be evaluated by A, B, C, D, F (Excellent(5), Good(5), Fair(4), Delay(3), Fail(0))

60% Final exam.

Any Questions?

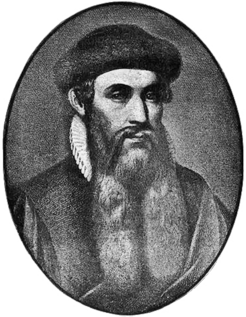
Two Things Change the World

Johann Gutenberg



Johann Gutenberg (1398 - 1468)

Johann Gutenberg



Johann Gutenberg (1398 - 1468)

In 1448 in the German city of Mainz a goldsmith named Johann Gutenberg discovered a way to print books by putting together movable metallic pieces.

Sheng BI

Bì Shēng (972-1051)

Bì Shēng was a Chinese artisan, engineer, and inventor of the world's first movable type technology, with printing being one of the Four Great Inventions of Ancient China.

Two Ideas Changed the World

Shanghai Jiao Tong University

Because of the typography, literacy spread, the Dark Ages ended, the human intellect was liberated, science and technology triumphed, the Industrial Revolution happened.

Many historians say we owe all this to typography.

Others insist that the key development was not typography, but algorithms.

Decimal System

Gutenberg would write the number 1448 as $MCDXLVIII$.

Decimal System

Gutenberg would write the number 1448 as $MCDXLVIII$.

How to add two Roman numerals? What is

$$MCDXLVIII + DCCCXII$$

Decimal System

Gutenberg would write the number 1448 as $MCDXLVIII$.

How to add two Roman numerals? What is

$$MCDXLVIII + DCCCXII$$

The decimal system was invented in India around AD 600. Using only 10 symbols, even very large numbers were written down compactly, and arithmetic is done efficiently by elementary steps.

Al Khwarizmi

Al Khwarizmi (780 - 850)

Al Khwarizmi

Al Khwarizmi (780 - 850)

In the 12th century, Latin translations of his work on the Indian numerals, introduced the decimal system to the Western world. (Source: Wikipedia)

Algorithms

SHANGHAI JIAO TONG UNIVERSITY

Al Khwarizmi laid out the basic methods for

- adding,
- multiplying,
- dividing numbers,
- extracting square roots,
- calculating digits of π.

Algorithms

Al Khwarizmi laid out the basic methods for

- adding,
- multiplying,
- dividing numbers,
- extracting square roots,
- calculating digits of π.

These procedures were precise, unambiguous, mechanical, efficient, correct.

Algorithms

Al Khwarizmi laid out the basic methods for

- adding,
- multiplying,
- dividing numbers,
- extracting square roots,
- calculating digits of π.

These procedures were precise, unambiguous, mechanical, efficient, correct.

They were algorithms, a term coined to honor the wise man after the decimal system was finally adopted in Europe, many centuries later.

Chongzhi ZU

Chongzhi ZU (429 – 500)

A Chinese astronomer, inventor, mathematician, politician, and writer during the Liu Song and Southern Qi dynasties. He was most notable for calculating π as between 3.1415926 and 3.1415927, a record in precision which would not be surpassed for nearly 900 years.

What Is An Algorithm

What Is An Algorithm

A step by step procedure for solving a problem or accomplishing some end.

What Is An Algorithm

A step by step procedure for solving a problem or accomplishing some end.

An abstract recipe, prescribing a process which may be carried out by a human, a computer or by other means.

What Is An Algorithm

A step by step procedure for solving a problem or accomplishing some end.

An abstract recipe, prescribing a process which may be carried out by a human, a computer or by other means.

Any well-defined computational procedure that makes some value, or set of values, as input and produces some value, of set of values, as output. An algorithm is thus a finite sequence of computational steps that transform the input into the output.

What Is An Algorithm

An algorithm is a procedure that consists of

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions
- that terminates in a finite number of steps.

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions
- that terminates in a finite number of steps.

A program is

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions
- that terminates in a finite number of steps.

A program is

- an implementation of an algorithm, or algorithms.

What Is An Algorithm

An algorithm is a procedure that consists of

- a finite set of instructions which,
- given an input from some set of possible inputs,
- enables us to obtain an output through a systematic execution of the instructions
- that terminates in a finite number of steps.

A program is

- an implementation of an algorithm, or algorithms.
- A program does not necessarily terminate.

Fibonacci Algorithm

Leonardo Fibonacci

Leonardo Fibonacci (1170 - 1250)

Leonardo Fibonacci

Leonardo Fibonacci (1170 - 1250)

Fibonacci helped the spread of the decimal system in Europe, primarily through the publication in the early 13th century of his Book of Calculation, the Liber Abaci. (Source: Wikipedia)

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Formally,

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Formally,

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

Q: What is F_{100} or F_{200}?

An Exponential Algorithm


```
FIBO1(n)
a nature number n;

if n = 0 then return(0);
if n = 1 then return(1);
return(FIBO1(n − 1)+FIBO1(n − 2));
```

Three Questions about An Algorithm

1. Is it correct?
2. How much time does it take, as a function of n?
3. Can we do better?

Three Questions about An Algorithm

SHANGHAI JIAO TONG UNIVERSITY

1. Is it correct?
2. How much time does it take, as a function of n?
3. Can we do better?

The first question is trivial, as this algorithm is precisely Fibonacci's definition of F_n

How Much Time

Let $T(n)$ be the number of computer steps needed to compute `FIBO1`(n)

How Much Time

Let $T(n)$ be the number of computer steps needed to compute $\texttt{FIBO1}(n)$

For $n \leq 1$,

$$T(n) \leq 2$$

How Much Time

Let $T(n)$ be the number of computer steps needed to compute $\texttt{FIBO1}(n)$

For $n \leq 1$,

$$T(n) \leq 2$$

For $n \geq 1$,

$$T(n) = T(n-1) + T(n-2) + 3$$

How Much Time

Let $T(n)$ be the number of computer steps needed to compute $\texttt{FIBO1}(n)$

For $n \leq 1$,

$$T(n) \leq 2$$

For $n \geq 1$,

$$T(n) = T(n-1) + T(n-2) + 3$$

It is easy to shown, for all $n \in \mathbb{N}$,

$$T(n) \geq F_n$$

How Much Time

Let $T(n)$ be the number of computer steps needed to compute $\texttt{FIBO1}(n)$

For $n \leq 1$,

$$T(n) \leq 2$$

For $n \geq 1$,

$$T(n) = T(n-1) + T(n-2) + 3$$

It is easy to shown, for all $n \in \mathbb{N}$,

$$T(n) \geq F_n$$

It is exponential to n.

Why Exponential Is Bad?

SHANGHAI JIAO TONG UNIVERSITY

$$T(200) \geq F_{200} \geq 2^{138} \approx 2.56 \times 10^{42}$$

Why Exponential Is Bad?

$$T(200) \geq F_{200} \geq 2^{138} \approx 2.56 \times 10^{42}$$

In 2010, the fastest computer in the world is the Tianhe-1A system at the National Supercomputer Center in Tianjin.

Why Exponential Is Bad?

$$T(200) \geq F_{200} \geq 2^{138} \approx 2.56 \times 10^{42}$$

In 2010, the fastest computer in the world is the Tianhe-1A system at the National Supercomputer Center in Tianjin.

Its speed is

$$2.57 \times 10^{15}$$

steps per second.

Why Exponential Is Bad?

SHANGHAI JIAO TONG UNIVERSITY

$$T(200) \geq F_{200} \geq 2^{138} \approx 2.56 \times 10^{42}$$

In 2010, the fastest computer in the world is the Tianhe-1A system at the National Supercomputer Center in Tianjin.

Its speed is

$$2.57 \times 10^{15}$$

steps per second.

Thus to compute F_{200} Tianhe-1A needs roughly

$$10^{27} \text{ seconds} \geq 10^{22} \text{ years.}$$

Why Exponential Is Bad?

SHANGHAI JIAO TONG UNIVERSITY

$$T(200) \geq F_{200} \geq 2^{138} \approx 2.56 \times 10^{42}$$

In 2010, the fastest computer in the world is the Tianhe-1A system at the National Supercomputer Center in Tianjin.

Its speed is

$$2.57 \times 10^{15}$$

steps per second.

Thus to compute F_{200} Tianhe-1A needs roughly

$$10^{27} \text{ seconds} \geq 10^{22} \text{ years.}$$

In 2022, the fastest is Frontier, 1.102×10^{18} per second.

Moore's Law

Moore's Law:

Computer speeds have been doubling roughly every 18 months.

Moore's Law:

Computer speeds have been doubling roughly every 18 months.

The running time of `FIB01` is proportional to

$$2^{0.694n} \approx 1.6^n$$

Thus, it takes 1.6 times longer to compute F_{n+1} than F_n.

Moore's Law

Moore's Law:

Computer speeds have been doubling roughly every 18 months.

The running time of `FIB01` is proportional to

$$2^{0.694n} \approx 1.6^n$$

Thus, it takes 1.6 times longer to compute F_{n+1} than F_n.

So if we can reasonably compute F_{100} with this year's technology, then next year we will manage F_{101}, and so on . . .

Moore's Law

Moore's Law:

Computer speeds have been doubling roughly every 18 months.

The running time of `FIB01` is proportional to

$$2^{0.694n} \approx 1.6^n$$

Thus, it takes 1.6 times longer to compute F_{n+1} than F_n.

So if we can reasonably compute F_{100} with this year's technology, then next year we will manage F_{101}, and so on . . .

Just one more number every year!

Moore's Law

Moore's Law:

Computer speeds have been doubling roughly every 18 months.

The running time of `FIB01` is proportional to

$$2^{0.694n} \approx 1.6^n$$

Thus, it takes 1.6 times longer to compute F_{n+1} than F_n.

So if we can reasonably compute F_{100} with this year's technology, then next year we will manage F_{101}, and so on . . .

Just one more number every year!

Such is the curse of exponential time.

Three Questions

1. Is it correct?
2. How much time does it take, as a function of n?
3. Can we do better?

Three Questions

1. Is it correct?
2. How much time does it take, as a function of n?
3. Can we do better?

Now we know $\texttt{FIB1}(n)$ is correct and inefficient, so can we do better?

An Polynomial Algorithm


```
FIBO2(n)
a nature number n;

if n = 0 then return(0);
create an array f[0...n];
f[0] = 0; f[1] = 1;
for i = 2 to n do
    f[i] = f[i-1] + f[i-2];
end
return(f[n]);
```

An Analysis

The correctness of `FIBO2` is trivial.

An Analysis

The correctness of `FIBO2` is trivial.

How long does it take?

An Analysis

The correctness of `FIBO2` is trivial.

How long does it take?

The inner loop consists of a single computer step and is executed $n - 1$ times. Therefore the number of computer steps used by `FIBO2` is linear in n.

A More Careful Analysis

We count the number of basic computer steps executed by each algorithm and regard these basic steps as taking a constant amount of time.

A More Careful Analysis

We count the number of basic computer steps executed by each algorithm and regard these basic steps as taking a constant amount of time.

It is reasonable to treat addition as a single computer step if small numbers are being added, e.g., 32-bit numbers.

A More Careful Analysis

We count the number of basic computer steps executed by each algorithm and regard these basic steps as taking a constant amount of time.

It is reasonable to treat addition as a single computer step if small numbers are being added, e.g., 32-bit numbers.

The n-th Fibonacci number is about $0.694n$ bits long, and this can far exceed 32 as n grows.

A More Careful Analysis

We count the number of basic computer steps executed by each algorithm and regard these basic steps as taking a constant amount of time.

It is reasonable to treat addition as a single computer step if small numbers are being added, e.g., 32-bit numbers.

The n-th Fibonacci number is about $0.694n$ bits long, and this can far exceed 32 as n grows.

Arithmetic operations on arbitrarily large numbers cannot possibly be performed in a single, constant-time step.

A More Careful Analysis

The addition of two n-bit numbers takes time roughly proportional to n (next lecture).

A More Careful Analysis

The addition of two n-bit numbers takes time roughly proportional to n (next lecture).

`FIB01`, which performs about F_n additions, uses a number of basic step roughly proportional to nF_n.

A More Careful Analysis

The addition of two n-bit numbers takes time roughly proportional to n (next lecture).

FIBO1, which performs about F_n additions, uses a number of basic step roughly proportional to nF_n.

The number of steps taken by FIBO2 is proportional to n^2, and still polynomial in n.

A More Careful Analysis

The addition of two n-bit numbers takes time roughly proportional to n (next lecture).

`FIBO1`, which performs about F_n additions, uses a number of basic step roughly proportional to nF_n.

The number of steps taken by `FIBO2` is proportional to n^2, and still polynomial in n.

Q: Can we do better?

A More Careful Analysis

The addition of two n-bit numbers takes time roughly proportional to n (next lecture).

`FIBO1`, which performs about F_n additions, uses a number of basic step roughly proportional to nF_n.

The number of steps taken by `FIBO2` is proportional to n^2, and still polynomial in n.

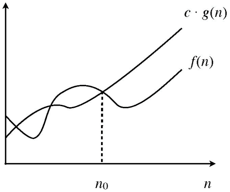
Q: Can we do better?

- Exercise 0.4

Big-O Notation

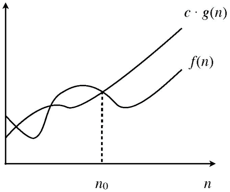
Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



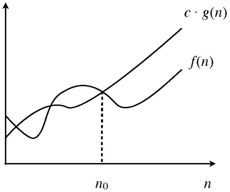
Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $O(n^2)$.
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.

Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.



Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $O(n^2)$.
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion sort makes $O(n^2)$ compares to sort n elements.

Quiz

Let $f(n) = 3n^2 + 17n \log_2 n + 1000$. Which of the following are true?

- **A** $f(n)$ is $O(n^2)$.
- **B** $f(n)$ is $O(n^3)$.
- **C** Both A and B.
- **D** Neither A nor B.

Big O notational abuses

One-way "equality". $O(g(n))$ is a set of functions, but computer scientists often write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

Big O notational abuses

One-way "equality". $O(g(n))$ is a set of functions, but computer scientists often write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

Example

Consider $g_1(n) = 5n^3$ and $g_2(n) = 3n^2$.

- We have $g_1(n) = O(n^3)$ and $g_2(n) = O(n^3)$.
- But, do not conclude $g_1(n) = g_2(n)$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \max\{n_1, n_2\}$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \max\{n_1, n_2\}$.

Sums. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(\max\{g_1, g_2\})$.

Big O notation: properties

Shanghai Jiao Tong University

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \max\{n_1, n_2\}$.

Sums. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(\max\{g_1, g_2\})$.

Transitivity. If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Big O notation: properties

Reflexivity. f is $O(f)$.

Constants. If f is $O(g)$ and $c > 0$, then $c\,f$ is $O(g)$.

Products. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 f_2$ is $O(g_1 g_2)$.

Proof.

- $\exists c_1 > 0$ and $n_1 \geq 0$ such that $0 \leq f_1(n) \leq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
- $\exists c_2 > 0$ and $n_2 \geq 0$ such that $0 \leq f_2(n) \leq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
- Then, $0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n)$ for all $n \geq \max\{n_1, n_2\}$.

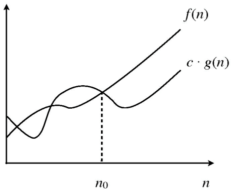
Sums. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(\max\{g_1, g_2\})$.

Transitivity. If f is $O(g)$ and g is $O(h)$, then f is $O(h)$.

Ex. $f(n) = 5n^3 + 3n^2 + n + 1234$ is $O(n^3)$.

Big Ω notation

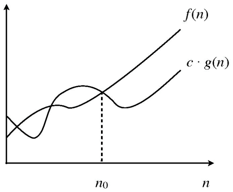
Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.



Big Ω notation

Shanghai Jiao Tong University

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.



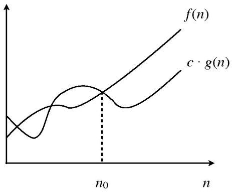
Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is both $\Omega(n^2)$ and $\Omega(n)$.
- $f(n)$ is not $\Omega(n^3)$.

Big Ω notation

Lower bounds. $f(n)$ is $\Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$.



Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is both $\Omega(n^2)$ and $\Omega(n)$.
- $f(n)$ is not $\Omega(n^3)$.

Typical usage. Any compare-based sorting algorithm requires $\Omega(n \log n)$ compares in the worst case.

Quiz

Which is an equivalent definition of big Omega notation?

A $f(n)$ is $\Omega(g(n))$ iff $g(n)$ is $O(f(n))$.

B $f(n)$ is $\Omega(g(n))$ iff there exist constants $c > 0$ such that

$$f(n) \geq c \cdot g(n) \geq 0$$

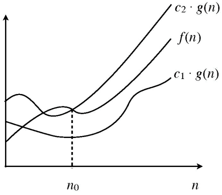
for infinitely many n.

C Both A and B.

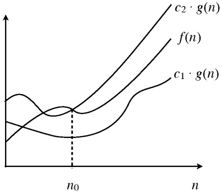
D Neither A nor B.

Big Θ notation

Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0, c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Big Θ notation



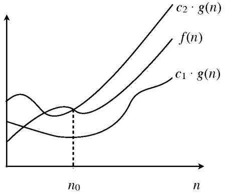
Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0, c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.

Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $\Theta(n^2)$.
- $f(n)$ is neither $\Theta(n^3)$ nor $\Omega(n)$.

Big Θ notation



Tight bounds. $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1 > 0, c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.

Example

Let $f(n) = 32n^2 + 17n + 1$.

- $f(n)$ is $\Theta(n^2)$.
- $f(n)$ is neither $\Theta(n^3)$ nor $\Omega(n)$.

Typical usage. Mergesort makes $\Theta(n \log n)$ compares to sort n elements.

Quiz

Which is an equivalent definition of big Theta notation?

A $f(n)$ is $\Theta(g(n))$ iff $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$.

B $f(n)$ is $\Theta(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < +\infty$.

C Both A and B.

D Neither A nor B.

Asymptotic bounds and limits

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ *for some constant* $0 < c < \infty$ *then* $f(n)$ *is* $\Theta(g(n))$.

Asymptotic bounds and limits

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ *for some constant* $0 < c < \infty$ *then* $f(n)$ *is* $\Theta(g(n))$.

Proof.

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ *for some constant* $0 < c < \infty$ *then* $f(n)$ *is* $\Theta(g(n))$.

Proof.

By definition of the limit, for any $\varepsilon > 0$, there exists n_0 such that

$$c - \varepsilon \leq \frac{f(n)}{g(n)} \leq c + \varepsilon$$

for all $n \geq n_0$.

Choose $\varepsilon = 1/2c > 0$.

Multiplying by $g(n)$ yields $1/2c \cdot g(n) \leq f(n) \leq 3/2c \cdot g(n)$ for all $n \geq n_0$.

Thus, $f(n)$ is $\Theta(g(n))$ by definition, with $c_1 = 1/2c$ and $c_2 = 3/2c$.

Asymptotic bounds and limits

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *then* $f(n)$ *is* $O(g(n))$ *but not* $\Omega(g(n))$.

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, *then* $f(n)$ *is* $\Omega(g(n))$ *but not* $O(g(n))$.

Asymptotic bounds for some common functions

SHANGHAI JIAO TONG UNIVERSITY

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

Asymptotic bounds for some common functions

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

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

Asymptotic bounds for some common functions

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. $\log_a n$ is $O(n^d)$ for every $a > 1$ and every $d > 0$.

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

Asymptotic bounds for some common functions

Polynomials. Let $f(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. $\log_a n$ is $O(n^d)$ for every $a > 1$ and every $d > 0$.

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

Exponentials and polynomials. n^d is $O(r^n)$ for every $r > 1$ and every $d > 0$.

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

Asymptotic bounds for some common functions

Factorials. $n!$ is $2^{\Theta(n \log n)}$.

Asymptotic bounds for some common functions

Factorials. $n!$ is $2^{\Theta(n \log n)}$.

Stirling's formula:

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$$

Big O notation with multiple variables

Upper bounds. $f(m,n)$ is $O(g(m,n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and$n_0 \geq 0$ such that $f(m,n) \leq c \cdot g(m,n)$ for all $n \geq n_0$ and $m \geq m_0$.

Big O notation with multiple variables

Upper bounds. $f(m,n)$ is $O(g(m,n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and$n_0 \geq 0$ such that $f(m,n) \leq c \cdot g(m,n)$ for all $n \geq n_0$ and $m \geq m_0$.

Example

$f(m,n) = 32mn^2 + 17mn + 32n^3$.

- $f(m,n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $f(m,n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Big O notation with multiple variables

Upper bounds. $f(m,n)$ is $O(g(m,n))$ if there exist constants $c>0$, $m_0 \geq 0$, and$n_0 \geq 0$ such that $f(m,n) \leq c \cdot g(m,n)$ for all $n \geq n_0$ and $m \geq m_0$.

Example

$f(m,n) = 32mn^2 + 17mn + 32n^3$.

- $f(m,n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $f(m,n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $O(m+n)$ time to find a shortest path from s to t in a digraph with n nodes and m edges.