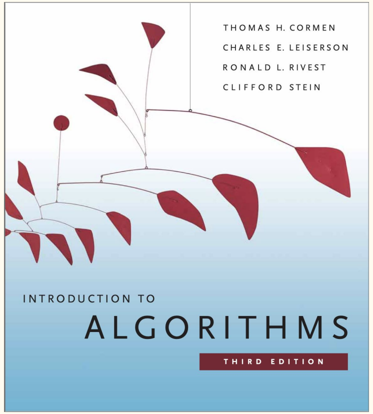
Data Structures and Algorithms

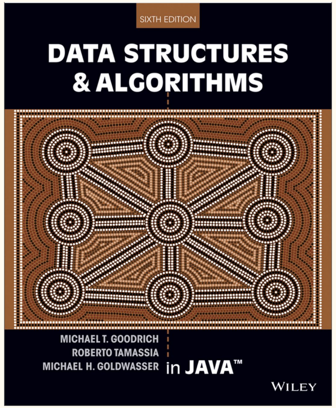
Tutorial 1. Asymptotic notation

Resources

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press 2009.



2. M. T. Goodrich, R. Tamassia, and M. H. Goldwasser. Data Structures and Algorithms in Java. WILEY 2014.



Office hours

Every Wednesday from 13:00 to 14:00 in room 409.

But also other time slots are possible by agreement (just ping me on Telegram @fizruk31337).

But also, use office hours of prof. Khan and your TA.

Motivation example problem

Problem. Given a positive integer number **n**,
find all possible non-negative integer values for variables **a**, **b**, **c** such that

$$\mathrm{a} + \mathrm{b} + \mathrm{c} = \mathrm{n}.$$

Motivation example problem

Problem. Given a positive integer number **n**,
find all possible non-negative integer values for variables **a**, **b**, **c** such that

$$a + b + c = n.$$

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

Motivation example problem

Problem. Given a positive integer number **n**,
find all possible non-negative integer values for variables **a**, **b**, **c** such that

$$a + b + c = n.$$

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

Motivation example problem

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

Which solution is better?

Why? How do we prove it?

Motivation example analysis

Idea #1: run on a computer and see which one is faster.

Motivation example analysis

Idea #1: run on a computer and see which one is faster.

Solution A	
N	**Time**
100	0.09s

Motivation example analysis

Idea #1: run on a computer and see which one is faster.

Solution A	
N	**Time**
100	0.09s
200	0.54s
300	1.82s
400	4.42s
500	8.96s

Motivation example analysis

Idea #1: run on a computer and see which one is faster.

Solution A	
N	**Time**
100	0.09s
200	0.54s
300	1.82s
400	4.42s
500	8.96s

Solution B	
N	**Time**
100	0.02s
200	0.05s
300	0.10s
400	0.17s
500	0.25s

Motivation example analysis

Idea #1: run on a computer and see which one is faster.

Some issues with this approach:

1. Requires actual implementation
   (easy for this example, but can be hard for complicated algorithms)
2. Requires multiple runs on a computer (takes resources)
3. Hard to test on large inputs
   (and an algorithm can be slow on small inputs)
4. Hard to replicate, requires testing under the same environment
   (same computer, same OS, same compiler, etc.)
5. Anything else?

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

How many times is this condition checked (in terms of **n**)?

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

How many times is this condition checked (in terms of **n**)?

$$(n+1)^3$$

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

How many times is statement executed (in terms of **n**)?

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

How many times is statement executed (in terms of **n**)?

$$(n+1)^2$$

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

$$(n+1)^3$$

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

$$(n+1)^2$$

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Solution A:

```
for a from 0 to n
  for b from 0 to n
    for c from 0 to n
      if (a + b + c = n) then
        print (a, b, c)
```

Solution B:

```
for a from 0 to n
  for b from 0 to n
    c := n - b - a
    print (a, b, c)
```

$$(n+1)^3 \geqslant (n+1)^2$$

Motivation example analysis

Idea #2: compute running time as a function of **n**,
based off the pseudocode.

Some issues with this approach:

1. Some formulae cannot be compared uniformly for all n.
2. We do not actually care about precise running time, only its growth rate.

$$(n+1)^3 \qquad \geqslant \qquad (n+1)^2$$

Motivation example analysis

Idea #3: compute asymptotic complexity as a function of **n**.

Motivation example analysis

Idea #3: compute asymptotic complexity as a function of **n**.

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

Motivation example analysis

Idea #3: compute asymptotic complexity as a function of **n**.

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

This term grows fastest!

Motivation example analysis

Idea #3: compute asymptotic complexity as a function of **n**.

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

This term grows fastest!
So for sufficiently large n,
other terms do not matter!

Motivation example analysis

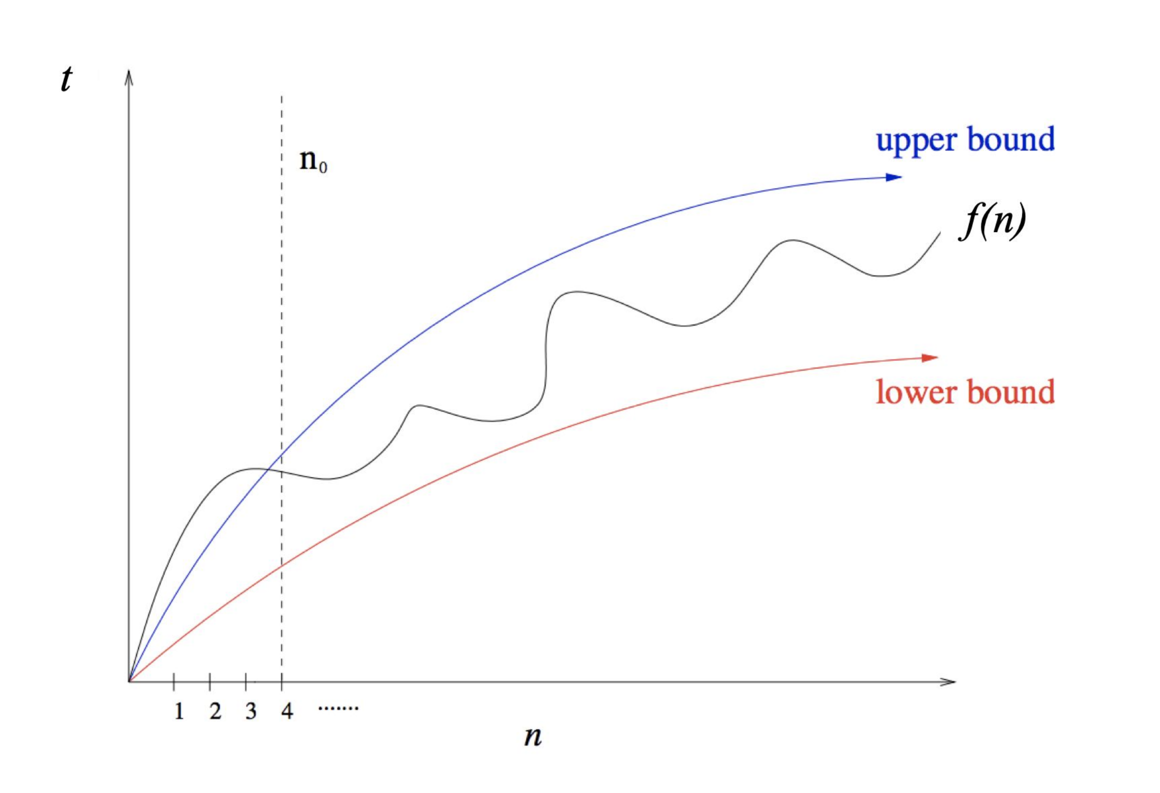
Idea #3: compute asymptotic complexity as a function of **n**.

$$(n+1)^3 = n^3 + 3n^2 + 3n + 1$$

This term grows fastest!

$$n^3 + 3n^2 + 3n + 1 = O(n^3)$$

Asymptotic upper and lower bounds



Asymptotic notation. Big-Oh notation

Definition. Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = O(g(n))$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$

Asymptotic notation. Big-Oh notation

Definition. Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = O(g(n))$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$

$\boxed{\text{Eventually (for large enough n)}}$

Asymptotic notation. Big-Oh notation

Definition. Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = O(g(n))$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$

Eventually (for large enough n)

Asymptotic notation. Big-Oh notation

Definition. Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = O(g(n))$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$

Constant factors do not matter

Asymptotic notation. Big-Oh notation

Example 1. $n^2 + 3n = O(n^3)$

Asymptotic notation. Big-Oh notation

Example 1. $n^2 + 3n = O(n^3)$

Proof.

We need to find constants c and n_0, such that for all $n \geq n_0$

$$n^2 + 3n \leq c \cdot n^3$$

Reformulating inequality (dividing by n^3): $1/n + 3/n^2 \leq c$

Let $n_0 = 10$ and $c = 1$.

Then $1/n + 3/n^2 < 1 = c$ for any $n \geq n_0$.

And so the required inequality is satisfied.

QED.

Asymptotic notation. Big-Oh notation

Remark. Note that all of the following statements are correct:

- $n^2 + 3n = O(n!)$
- $n^2 + 3n = O(2^n)$
- $n^2 + 3n = O(n^3)$
- $n^2 + 3n = O(n^2)$

But only the last one provides a **tight** upper bound, since it cannot be improved any further.

Asymptotic notation. Big-Oh notation

Example 2. $\sin n = O(1)$

Asymptotic notation. Big-Oh notation

Example 2. $\sin n = O(1)$

Proof.

We need to find constants c and n_0, such that for all $n \geq n_0$

$$\sin n \leq c \cdot 1$$

Let $n_0 = 1$ and $c = 1$. Then $\sin n \leq 1 = c$ for any n.

QED.

Asymptotic notation. Big-Oh notation

Remark. Obviously, big-Oh notation is abusing the equality symbol, since it is not symmetric. To be more formally correct, some people (mostly mathematicians, as opposed to computer scientists) prefer to define $O(g(x))$ as a set-valued function, whose value is all functions that do not grow faster than $g(x)$, and use set membership notation to indicate that a specific function is a member of the set thus defined.
Both forms are in common use, but the sloppier equality notation is more common at present.

https://web.mit.edu/16.070/www/lecture/big_o.pdf

Asymptotic notation. Big-Oh notation

Example 3. Explain why the following statement does not make sense?

«The running time of this algorithm is at least $O(n^2)$»

Asymptotic notation. Big-Oh notation

Example 3. Explain why the following statement does not make sense?

«The running time of this algorithm is at least $O(n^2)$»

Explanation. Big-Oh notation is used to provide upper bound, but «at least» implies a lower bound.

Asymptotic notation. Big-Oh notation

Example 4. $2^{n+1} = O(2^n)$?

Asymptotic notation. Big-Oh notation

Example 4. $2^{n+1} = O(2^n)$?

Solution. Yes, $2^{n+1} = O(2^n)$. To prove this we need to find constants c and n_0 such that

$$2^{n+1} \leq c \cdot 2^n$$

Let $c = 2$ and $n_0 = 1$. Then $2^{n+1} = 2 \cdot 2^n = c \cdot 2^n$. QED.

Asymptotic notation. Big-Oh notation

Example 5. $2^{2n} = O(2^n)$?

Asymptotic notation. Big-Oh notation

Example 5. $2^{2n} = O(2^n)$?

Solution. No, $2^{2n} \neq O(2^n)$. To prove this we need to show that for any constants c and n_0, there exists some $n \geq n_0$, such that

$$2^{2n} > c \cdot 2^n$$

We simply need to find n such that $2^n > c$. Since c is a constant that does not depend on n, we can always find such n.
More precisely, $n = 1 + [\log_2 c]$.

Asymptotic notation. Big-Oh notation

Definition (big-Oh notation). Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = O(g(n))$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$

Asymptotic notation. Big-Omega notation

Definition (big-Omega notation). Let f(n) and g(n) be functions from positive integers to positive reals. Then we write

$$\mathrm{f(n)} = \Omega(\mathrm{g(n)})$$

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$

Asymptotic notation. Big-Omega notation

Definition (big-Omega notation). Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

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

if and only if there exist constants c and n_0 such that for all $n \geq n_0$ we have $f(n) \geq c \cdot g(n)$

Equivalently. $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

Asymptotic notation. Theta notation

Definition (Theta notation). Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

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

if and only if there exist constants c_1, c_2 and n_0 such that for all $n \geq n_0$ we have $c_1 \cdot g(n) \geq f(n) \geq c_2 \cdot g(n)$.

Asymptotic notation. Theta notation

Definition (Theta notation). Let f(n) and g(n) be functions from positive integers to positive reals. Then we write

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

if and only if there exist constants c_1, c_2 and n_0 such that for all $n \geq n_0$ we have $c_1 \cdot g(n) \geq f(n) \geq c_2 \cdot g(n)$.

Equivalently. $f(n) = \theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic notation. Little-Oh notation

Definition (little-Oh notation). Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

$$f(n) = o(g(n))$$

if and only if for any constant c there exists constant n_0 such that for all $n \geq n_0$ we have $f(n) < c \cdot g(n)$.

Asymptotic notation. Little-Omega notation

Definition (little-Omega notation). Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals. Then we write

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

if and only if for any constant c there exists constant n_0 such that for all $n \geq n_0$ we have $f(n) > c \cdot g(n)$.

Equivalently. $f(n) = \omega(g(n))$ if and only if $g(n) = o(f(n))$.

Asymptotic notation. Summary

Notation	Definition	Analogy
$f(n) = O(g(n))$	see above	$\leq$
$f(n) = o(g(n))$	see above	$<$
$f(n) = \Omega(g(n))$	$g(n)=O(f(n))$	$\geq$
$f(n) = \omega(g(n))$	$g(n)=o(f(n))$	$>$
$f(n) = \theta(g(n))$	$f(n)=O(g(n))$ and $g(n)=O(f(n))$	$=$

Asymptotic notation. Big-Oh vs Theta

Remark. A common error is to confuse big-Oh and theta.

For example, one might say

«heapsort is $O(n \log n)$»

when the intended meaning was

«heapsort is $\theta(n \log n)$».

Both statements are true, but the latter is a stronger claim.

Asymptotic notation. Exercises

Exercise 6. Let f(n) and g(n) be asymptotically non-negative functions. Prove that

$$\max(f(n), g(n)) = \theta(f(n) + g(n))$$

Exercise 7. Show that for any real constants a and b, where $b > 0$, we have

$$(n+a)^b = \theta(n^b)$$

Asymptotic notation. More exercises

Exercise 8. Assume

$$f(n) = O(n^2)$$
$$g(n) = O(\log n)$$

Prove that

$$f(n) \cdot g(n) = O(n^2 \cdot \log n)$$

Asymptotic notation. More exercises

Exercise 9. Assume

$$f(n) = O(g(n))$$
$$g(n) = O(h(n))$$

Prove that

$$f(n) = O(h(n))$$

Summary

- Motivation for asymptotic complexity of algorithms
- Asymptotic notation
- Asymptotic analysis
- Properties of asymptotics