Week 2

Growth of functions

1. Growth of Functions
   - Analysing algorithms
2. Examples

Motivation

Last week we studied Insertion Sort, and described its time-complexity as "roughly n^2":

1. Using 10^9 operations per second, that means roughly 32 years on a single processor for the case $n = 10^9$.
2. n^2 came from $1 + 2 + 3 + \cdots + n$ steps, which more precisely is $\frac{n}{2} \cdot n = \frac{1}{2} n^2$, for the worst-case.
3. So we could say it's actually $\frac{1}{2} \cdot 32 = 16$ years.
4. And considering the average case, we calculated that actually then only $\frac{1}{4} n^2$ steps would be needed, i.e., 8 years.
5. But stop, we forgot about the real factors in the analysis, those c_i.
6. And we don't know about the real processors anyway ... ?!?

Motivation (cont.)

What we actually really need is that it takes just SECONDS.

This week we learn

- a method to speak about "n^2"
- without speaking about the factors above.

This will help us to emphasise the most important aspect of complexity.

Overview

- We consider an important tool for the analysis of algorithms: **Big-Oh**.
- Important also the relatives **Big-Omega**,
- and especially **Big-Theta**.

Reading from CLRS for week 2

- Chapter 2, Section 2
- Chapter 3

What is “growth of functions”?

- We want a way to describe behaviour of functions

  in the limit.
- That is, we are studying so-called asymptotic efficiency.
- In again other words, we describe the “growth of functions”.
- The motivation is to focus on what's important, by

  abstracting away low-order terms and constant factors.
- It is how we indicate running times of algorithms.
- It yields a way to compare “sizes” of functions:
  - O corresponds (asymptotically) to $\leq$
  - Ω corresponds (asymptotically) to $\geq$
  - Θ corresponds (asymptotically) to $=$.

What is given?

We start with a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$:

1. Its input n is the input size, as we have *chosen* to measure.
2. Its output $f(n)$ is the runtime, as we have *chosen* to specify.

So $f(n)$ is the generally *unknown, but specified*

- worst-case-, or
- best-case-, or
- average-case-

runtime for input size n.

It's like an unknown, an "x", we want to learn about.

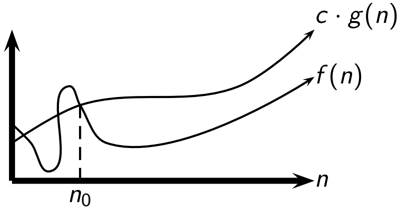
Now, more complicated than with simple equations you know, where one needs to solve for a single number, we need to "solve" for a whole function

— and that function is in general VERY complicated.
Thus we "don't want to know it too precisely".

O-Notation

$\mathrm{O}(g(n))$ is the set of all functions $f(n)$ for which there are constants $c \in \mathbb{R}_{>0}$ and $n_0 \in \mathbb{N}_0$ such that

$$f(n) \le c \cdot g(n) \qquad \text{for all } n \ge n_0.$$



$g(n)$ is an asymptotic upper bound for $f(n)$.

If $f(n) \in \mathrm{O}(g(n))$, we write $f(n) = \mathrm{O}(g(n))$.

O-Notation examples

$2n^2 = \mathrm{O}(n^2)$, with $c = 2$ and $n_0 = 0$.

Example of functions in $\mathrm{O}(n^2)$ (functions which are asymptotically upper-bounded by n^2):

- $2n^2$, and even $10^{10000}n^2 = 10^{10000} \cdot n^2$
- $n^2 + n$ (since $\leq 2n^2$)
- $n^2 + 1000n$ (since $\leq 1001n^2$)
- $1000n^2 + 1000n$ (since $\leq 2000n^2$)
- n, and $n/1000$, and $n \cdot 1000$
- $n^{1.999999}$
- $n^2 / \lg n = n^2 / \lg(n)$.

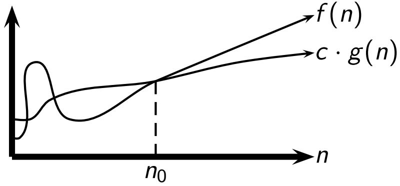
But NOT

- $n^{2.000001}$
- $n^2 \cdot \lg n$.

Ω-Notation

$\Omega(g(n))$ is the set of all functions $f(n)$ for which there are constants $c \in \mathbb{R}_{>0}$ and $n_0 \in \mathbb{N}_0$ such that

$$f(n) \geq c \cdot g(n) \qquad \text{for all } n \geq n_0.$$



$g(n)$ is an asymptotic lower bound for $f(n)$.

If $f(n) \in \Omega(g(n))$, we write $f(n) = \Omega(g(n))$.

Ω-Notation Examples

$n^3 = \Omega(n^2)$.

Functions in $\Omega(n^2)$ (functions which are asymptotically lower-bounded by n^2):

- $\frac{1}{2}n^2$
- $n^2 + n$
- $n^2 - n$
- $\frac{1}{1000}n^2 + 1000n$
- $\frac{1}{1000}n^2 - 1000n$
- n^3
- $n^{2.0000001}$
- $n^2 \cdot \lg n$
- 2^{2^n}

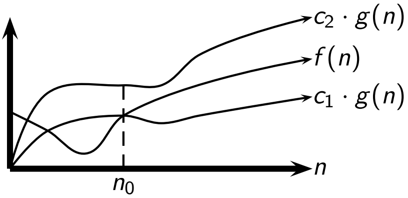
But NOT

- $n^{1.99999}$
- $n^2 / \lg n$.

Θ-Notation

$\Theta(g(n))$ is the set of all functions $f(n)$ for which there are constants $c_1, c_2 \in \mathbb{R}_{>0}$ and $n_0 \in \mathbb{N}_0$ such that

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \qquad \text{for all } n \geq n_0.$$



$g(n)$ is an asymptotic tight (exact) bound for $f(n)$.

If $f(n) \in \Theta(g(n))$, we write $f(n) = \Theta(g(n))$.

Θ-Notation (cont'd)

Examples 1

$n^2/2 - 2n = \Theta(n^2)$, with $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$, and $n_0 = 8$.

Theorem 2

$f(n) = \Theta(g(n))$ *if and only if*

$$f(n) = O(g(n)) \textit{ AND } f(n) = \Omega(g(n)).$$

Leading constants and lower order terms do not matter.

Example Analysis

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into sorted sequence A[1..j−1]:
4      i = j−1
5      while i > 0 and A[i] > key
6          A[i+1] = A[i]
7          i = i−1
8      A[i+1] = key
```

- Each single line, executed once, costs constant time.
- The **for** -loop on line 1 is executed $\mathrm{O}(n)$ times.‘
- The **while** -loop on lines 5-7 is executed $\mathrm{O}(n)$ times.

Thus overall worst-case runtime is: $\mathrm{O}(n) \cdot \mathrm{O}(n) = \mathrm{O}(n^2)$.

In fact, as seen last week, worst-case runtime is $\Theta(n^2)$.

Setting up the analysis

Assume we have given an algorithm $\mathcal{A}$ we want to analyse.

In order to do so, we have to come up with a **well-defined function**

$$f : \mathbb{N} \to \mathbb{R}_{\geq 0},$$

where $f(n)$ for some notion of "input size" n is the "resource usage" we want to analyse.

- So we have to define what n means; for example the input is an array, and we consider the length of the array.
- And we have to define what kind of "resource" to count; for example the number of comparisons between the elements of the array.

But STILL we do not have a *function* f:

For an input-"size" n
in general there are many possible inputs,
and thus many possible values of $f(n)$.

Determining how to count the steps

Fix input-size n. Consider all possible values

$$f(n) = y_1, y_2, \ldots$$

Remember the y_i are the number of "steps" for the possible inputs of size n.

worst-case $f(n) := \max(y_1, y_2, \ldots)$

best-case $f(n) := \min(y_1, y_2, \ldots)$

average-case Specify a probability $0 \leq \mu_i \leq 1$ for outcome y_i, and let $f(n) := \sum_i \mu_i \cdot y_i$.

Only now, finally, did we arrive at a FUNCTION f, and can apply the tools for the analysis of the growth of FUNCTIONS.

Choosing the right type of analysis

Alright, we have now our $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$.

The goal of the **asymptotic analysis** is (ALWAYS) to find some **simpler function** g with $f = \Theta(g)$.

In our example above, we considered $\mathcal{A}$ as InsertionSort, chose n as the length of the input array, chose $f(n)$ as the worst-case number of comparison, and obtained $f(n) = \Theta(n^2)$.

1. Typically one starts for this by showing $f = O(g_1)$, an **upper bound**: for **almost all** n holds $f(n) \leq c_1 \cdot g_1(n)$, for some (large) c_1.
2. Then one looks for a **lower bound** $f = \Omega(g_2)$: for almost all n holds $f(n) \geq c_2 \cdot g_2(n)$, for some (small) c_2.
3. If $g_1 = g_2$, then $f = \Theta(g_1) = \Theta(g_2)$, and one is done.
4. Otherwise one tries to find smaller g_1 and/or larger g_2.

Choosing the right type of analysis

Alright, we have now our $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$.

The goal of the **asymptotic analysis**
is (ALWAYS) to find some **simpler function** g
with $f = \Theta(g)$.

In our example above, we considered $\mathcal{A}$ as InsertionSort, chose n as the length of the input array, chose $f(n)$ as the worst-case number of comparison, and obtained $f(n) = \Theta(n^2)$.

1. Typically one starts for this by showing $f = O(g_1)$, an **upper bound**: for **almost all** n holds $f(n) \leq c_1 \cdot g_1(n)$, for some (large) c_1.
2. Then one looks for a **lower bound** $f = \Omega(g_2)$: for almost all n holds $f(n) \geq c_2 \cdot g_2(n)$, for some (small) c_2.
3. If $g_1 = g_2$, then $f = \Theta(g_1) = \Theta(g_2)$, and one is done.
4. Otherwise one tries to find smaller g_1 and/or larger g_2.

Choosing the right type of analysis

Alright, we have now our $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$.

The goal of the **asymptotic analysis**
is (ALWAYS) to find some **simpler function** g
with $f = \Theta(g)$.

In our example above, we considered $\mathcal{A}$ as InsertionSort, chose n as the length of the input array, chose $f(n)$ as the worst-case number of comparison, and obtained $f(n) = \Theta(n^2)$.

1. Typically one starts for this by showing $f = O(g_1)$, an **upper bound**: for **almost all** n holds $f(n) \leq c_1 \cdot g_1(n)$, for some (large) c_1.
2. Then one looks for a **lower bound** $f = \Omega(g_2)$: for almost all n holds $f(n) \geq c_2 \cdot g_2(n)$, for some (small) c_2.
3. If $g_1 = g_2$, then $f = \Theta(g_1) = \Theta(g_2)$, and one is done.
4. Otherwise one tries to find smaller g_1 and/or larger g_2.

The four main cases

First consider that f is **worst-case** run-time:

1. $f = O(g_1)$ means that for almost all n and **all inputs** of size n the run-time of $\mathcal{A}$ is at most $c_1 \cdot g_1(n)$, for some (fixed!) large c_1.
2. $f = \Omega(g_2)$ means that for almost all n **there exist inputs** of size n, such that the run-time of $\mathcal{A}$ is at least $c_2 \cdot g_2(n)$, for some (fixed!) small c_2.

Now assume that f is **best-case** run-time:

1. $f = O(g_1)$ means that for almost all n **there exist inputs** of size n, such that the run-time of $\mathcal{A}$ is at most $c_1 \cdot g_1(n)$, for some (fixed!) large c_1.
2. $f = \Omega(g_2)$ means that for almost all n and **all inputs** of size n the run-time of $\mathcal{A}$ is at least $c_2 \cdot g_2(n)$, for some (fixed!) small c_2.

The four main cases

First consider that f is **worst-case** run-time:

1. $f = O(g_1)$ means that for almost all n and **all inputs** of size n the run-time of $\mathcal{A}$ is at most $c_1 \cdot g_1(n)$, for some (fixed!) large c_1.
2. $f = \Omega(g_2)$ means that for almost all n **there exist inputs** of size n, such that the run-time of $\mathcal{A}$ is at least $c_2 \cdot g_2(n)$, for some (fixed!) small c_2.

Now assume that f is **best-case** run-time:

1. $f = O(g_1)$ means that for almost all n **there exist inputs** of size n, such that the run-time of $\mathcal{A}$ is at most $c_1 \cdot g_1(n)$, for some (fixed!) large c_1.
2. $f = \Omega(g_2)$ means that for almost all n and **all inputs** of size n the run-time of $\mathcal{A}$ is at least $c_2 \cdot g_2(n)$, for some (fixed!) small c_2.

The four main levels

For us, growth rates of functions $f(n)$ roughly fall into four categories:

Constant	there exists $K > 0$ with $f(n) \leq K$ for almost all n : here we have $f(n) = \Theta(1)$.
Logarithmic	$f(n) = \log(n)$ for some fixed base: $f(n) = \Theta(\lg(n))$
Polynomial	$f(n) = n^{\alpha}$ for any $\alpha > 0$
Exponential	$f(n) = \alpha^{n}$ for some $\alpha > 1$.

1. Any bounded function is asymptotically strictly smaller than any logarithmic function.
2. Any logarithmic function is asymptotically strictly smaller than any polynomial function.
3. Any polynomial function is asymptotically strictly smaller than any exponential function.

The four main levels (cont.)

Within the realm of polynomial functions:

n^{α} is asymptotically strictly smaller than n^{β} for $0 < \alpha < \beta$

Within the realm of exponential functions:

α^{n} is asymptotically strictly smaller than β^{n} for $1 < \alpha < \beta$.

That "$f(n)$ is asymptotically strictly smaller than $g(n)$" means:

$$f(n) = O(g(n)) \text{ and NOT } f(n) = \Omega(g(n)).$$

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO
7. $2^n = O(3^n)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO
7. $2^n = O(3^n)$? YES
8. $2^n = \Omega(3^n)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO

7. $2^n = O(3^n)$? YES
8. $2^n = \Omega(3^n)$? NO

9. $120n^2 + \sqrt{n} + 99n = O(n^2)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO

7. $2^n = O(3^n)$? YES
8. $2^n = \Omega(3^n)$? NO

9. $120n^2 + \sqrt{n} + 99n = O(n^2)$? YES
10. $120n^2 + \sqrt{n} + 99n = \Theta(n^2)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO
7. $2^n = O(3^n)$? YES
8. $2^n = \Omega(3^n)$? NO
9. $120n^2 + \sqrt{n} + 99n = O(n^2)$? YES
10. $120n^2 + \sqrt{n} + 99n = \Theta(n^2)$? YES
11. $\sin(n) = O(1)$?

Big-Oh, Omega, Theta by examples

1. $5n + 111 = O(n)$? YES
2. $5n + 111 = O(n^2)$? YES
3. $5n + 111 = \Omega(n)$? YES
4. $5n + 111 = \Omega(n^2)$? NO
5. $5n + 111 = \Theta(n)$? YES
6. $5n + 111 = \Theta(n^2)$? NO
7. $2^n = O(3^n)$? YES
8. $2^n = \Omega(3^n)$? NO
9. $120n^2 + \sqrt{n} + 99n = O(n^2)$? YES
10. $120n^2 + \sqrt{n} + 99n = \Theta(n^2)$? YES
11. $\sin(n) = O(1)$? YES