Algorithms and Data Structures 2

3 - Algorithm analysis

Dr Michele Sevegnani

School of Computing Science
University of Glasgow

michele.sevegnani@glasgow.ac.uk

Outline

- **Common growth rates**
- **Example running times**
- **More on asymptotic notation**
  - Θ-notation
  - Ω-notation
  - o-notation
- **Some exercises for you to try**

Common growth rates

- **We have seen linear $O(n)$ (ARRAY-MAX) and quadratic $O(n^2)$ (INSERTION-SORT)**
- **Other examples**
  - $O(1)$ or constant
  - $O(\log n)$ or logarithmic (to base 2)
  - $O(\sqrt{n}) = O(n^{1/2})$ or fractional power
  - $O(n)$ or linear
  - $O(n \log n)$ (usually just called $n \log n$) or quasilinear
  - $O(n^2)$ or quadratic
  - $O(n^3)$ or cubic

How fast does √n grow?

- **Note: we mean the positive square root when we say √n**
  - A function is a map from one set to another
  - √n is not strictly speaking a function as it takes a single value to two different values
  - Example: √4 is +2 or -2
- **Upper bound O(n)**
  - $\sqrt{n} \leq cn$
  - Pick $n_0 = 4$ and $c = 1$
- **Lower bound O(log n)**
  - $\log n \leq c\sqrt{n}$
  - Pick $n_0 = 64$ and $c = 1$

Comparing growth rates

n	O(1)	O(log n)	O(n log n)	O(n^2)
1	1	0	0	1
2	1	1	2	4
4	1	2	8	16
8	1	3	24	64
16	1	4	64	256
32	1	5	160	1024
64	1	6	384	4096
32	1	5	160	1024
64	1	6	384	4096
128	1	7	896	16384
256	1	8	2048	65536
512	1	9	4608	262144
1024	1	10	10240	1048576
2048	1	11	22528	4194304
4096	1	12	49152	16777216
8192	1	13	106496	67108864
16384	1	14	229376	268435456

Two other complexities (both impossibly slow)

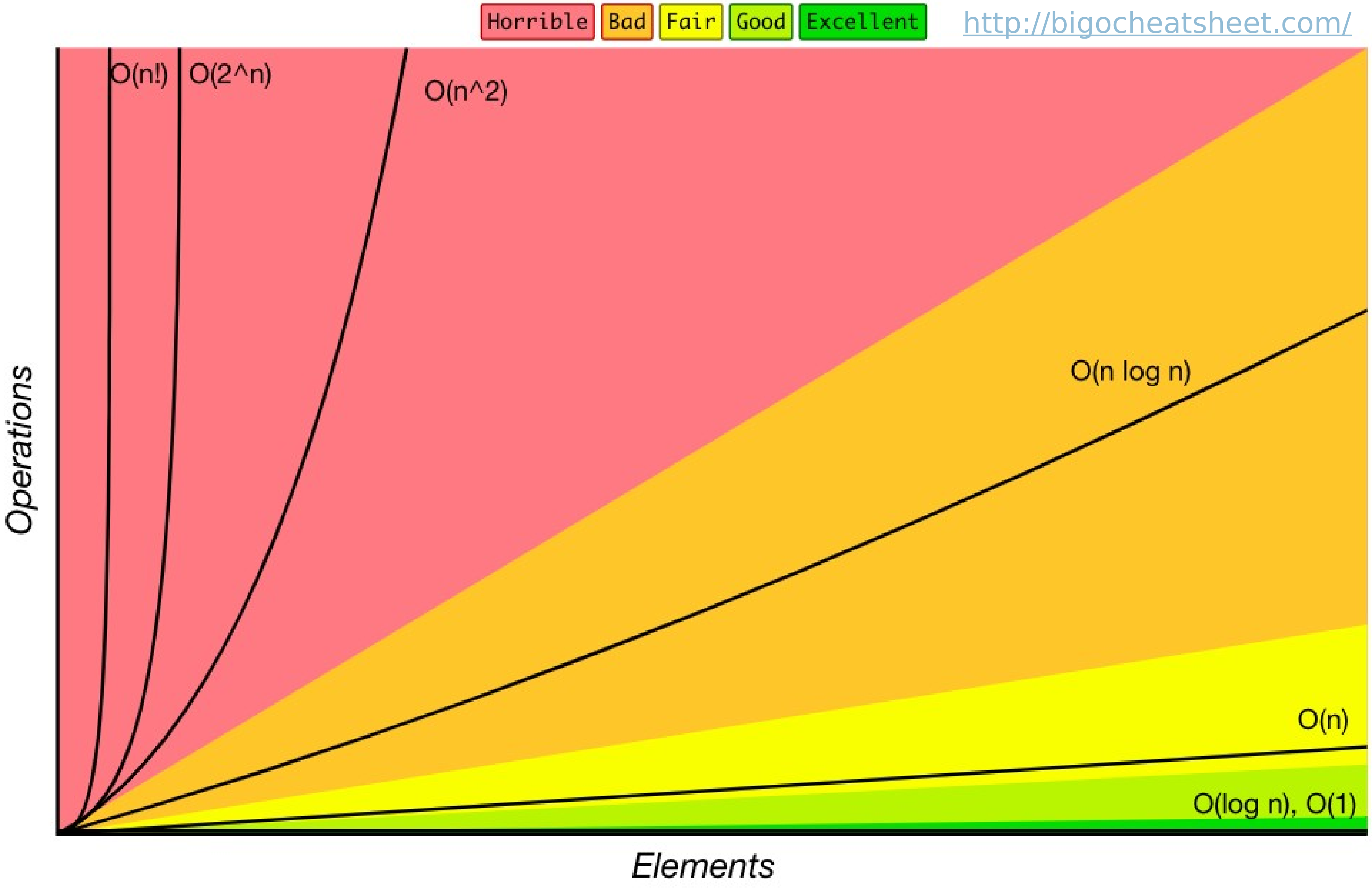
- **Exponential – $O(2^n)$**
  - Example: print out all the combinations/subsets from a set of size n
  - Every element is either in or out of a subset
- **Factorial – $O(n!)$**
  - Example: print out all the permutations of the n elements of a set

A handy lookup table

f(n)	O(f(n))	Description	How good?
A	O(1)	Constant	Nearly immediate
$A+B\log_2 n$	$O(\log n)$	Logarithmic	Stupendously fast
$A+B\sqrt{n}$	$O(\sqrt{n})$	Square root	Very fast
$A+Bn$	$O(n)$	Linear	Fast
$A+B\log_2 n+Cn$	$O(n)$	linear	Fast
$A+Bn\log_2 n$	$O(n \log n)$	n log n	Fairly fast
$A+Bn+Cn^2$	$O(n^2)$	Quadratic	Slow for large n
$A+Bn+Cn^2+Dn^3$	$O(n^3)$	Cubic	Slow for most n
$A+B2^n$	$O(2^n)$	Exponential	Impossibly slow
$An!$	$O(n!)$	Factorial	Impossibly slow

Or a graph

http://bigocheatsheet.com/



Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

```
SQUARES1(n)
  i := 0
  sum := 0
  while i < n
    increment i
    sum := sum + (i * i)
  return sum
```

Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

```
SQUARES1(n)
  i := 0
  sum := 0
  while i < n
    increment i
    sum := sum + (i * i)
  return sum
```

Operations

O(1)

O(1)

Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

```
SQUARES1(n)
  i := 0
  sum := 0
  while i < n
    increment i
    sum := sum + (i * i)
  return sum
```

Operations

O(1)
O(1)
O(n)
O(n)
O(n)

Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

SQUARES1(n)	Operations
i := 0	O(1)
sum := 0	O(1)
while i < n	O(n)
increment i	O(n)
sum := sum + (i * i)	O(n)
return sum	O(1)

Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

```
SQUARES1(n)
  i := 0
  sum := 0
  while i < n
    increment i
    sum := sum + (i * i)
  return sum
```

Operations

O(1)
O(1)
O(n)
O(n)
O(n)
O(1)

- **T(n) = O(1)+O(1)+O(n)+O(n)+O(n)+O(1) = O(n)**
- **Can we do better?**

Example 1

- **Input: integer n**
- **Output: the sum of the first n squares**

```
SQUARES1(n)
  i := 0
  sum := 0
  while i < n
    increment i
    sum := sum + (i * i)
  return sum
```

Operations

O(1)
O(1)
O(n)
O(n)
O(n)
O(1)

Summation rule

- **T(n) = O(1)+O(1)+O(n)+O(n)+O(n)+O(1) = O(n)**
- **Can we do better?**

Example 2

- **Input: integer n**
- **Output: the sum of the first n squares**

Summation rule

```
SQUARES2(n)
  sum := n * (n+1) * (2*n+1)/6
  return sum
```

Operations

O(1)

O(1)

- **T(n) = O(1) + O(1) = O(1)**
  - No loops!

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1)  ≤  n)
    increment i

  return i
```

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1) ≤  n)
    increment i

  return i
```

Example: find the integer square root of 13

i=0; $1^2 \leq 13$
i=1; $2^2 \leq 13$
i=2; $3^2 \leq 13$
i=3; $4^2 > 13$
return 3

3 iterations of the loop

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1) ≤  n)
    increment i

  return i
```

Operations

O(1)

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1) ≤  n)
    increment i

  return i
```

Operations

$O(1)$

$O(\sqrt{n})$

$O(\sqrt{n})$

- **Loop iterated for $i=0,1,..$ until $(i+1)^2 > n$**
  - until $i>\sqrt{n} -1$
  - number of times loop iterated is approximately $\sqrt{n}$ (rounded down to nearest integer)

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1) ≤ n)
    increment i
  return i
```

Operations

$O(1)$

$O(\sqrt{n})$

$O(\sqrt{n})$

$O(1)$

Example 3

- **Input: integer n**
- **Output: the integer part of the square root of n**

```
INT-SQRT1(n)
  i := 0
  while ((i+1)*(i+1) ≤  n)
    increment i

  return i
```

Operations

$O(1)$

$O(\sqrt{n})$

$O(\sqrt{n})$

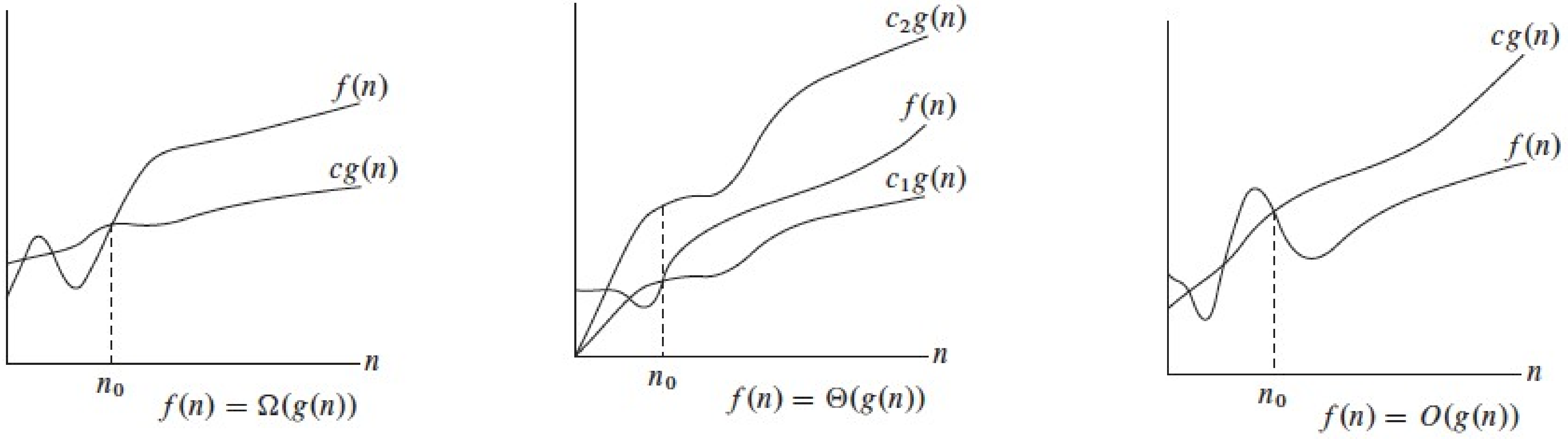
$O(1)$

- $\mathbf{T(n) = O(1)+O(\sqrt{n})+O(\sqrt{n})+O(1) = O(\sqrt{n})}$

More on asymptotic notation

- **Let f(n) be a function of n then**
  - $f(n) = \Omega(g(n))$ if there are constants c and n_0 such that $f(n) \geq cg(n)$ when $n \geq n_0$
  - $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
  - $f(n) = O(g(n))$ if there are constants c and n_0 such that $f(n) \leq cg(n)$ when $n \geq n_0$

Big-Omega
Big-Theta



Example

- **Let $f(n) = 5n^3+2$**
- **$f(n)$ is always bounded above by $6n^3$ (with $n>2$)**
  - $f(n)$ is $O(n^3)$
  - But it is also true that $f(n) = O(n^4)$ and $O(n^{15})$
  - Upper bound (may or may not be asymptotically tight)
- **$f(n)$ is always bounded below by $4n^3$ (with $n>0$)**
  - $f(n)$ is $\Omega(n^3)$
  - Lower bound
- **$f(n)$ is both $O(n^3)$ and $\Omega(n^3)$**
  - Thus $f(n)$ is $\Theta(n^3)$
  - Tight bound

Non-tight upper bounds (little-oh notation)

- **Let $f(n)$ be a function of n then $f(n)=o(g(n))$ if $f(n)=O(g(n))$ and $f(n) \neq \Theta(g(n))$**
  - Intuitively, $f(x)=o(g(x))$ means that $g(x)$ grows much faster than $f(x)$
- **Facts**
  - For every function $f(n)$, $f(n)=o(g(n))$ and $f(n)=O(g(n))$
  - Not every function that is big-O of g is also little-o of g
- **Alternative definition: $f(n)=o(g(n))$ if**

- **From previous example: let $f(n) = 5n^3+2$**
  - $f(n)$ is $O(n^4)$ and $\Theta(n^3)$, hence $f(n) = o(n^4)$

Exercises for you to try

- Analyse the running time of the following algorithms (using **big-Oh** notation)

```
sum := 0
  for i = 1 to n
    for j = 1 to i
      sum := sum + 1
```

```
sum := 0
for i = 1 to n
  for j = 1 to i*i
    for k = 1 to j
      sum := sum + 1
```

```
sum := 0
for i = 1 to n
  for j = 1 to i*i
    if j mod i = 0 then
    for k = 1 to j
      sum := sum + 1
```

Solutions

```
sum := 0
  for i = 1 to n
    for j = 1 to i
      sum := sum + 1
```

- **Explicitly iterating over i**
  - $i = 1$ inner loop is executed 1 times
  - $i = 2$ inner loop is executed 2 times
  - …
  - $i = n$ inner loop is executed n times
- **Summing up**
  - $T(n) = 1 + 2 + \ldots + n = n(n+1)/2 = O(n^2)$
- **Alternatively note that**
  - i can grow up to n
  - j can grow up to $i = n$
- **Then apply the rule for nested loops to obtain**
  - $O(n * n) = O(n^2)$

Solutions

```
sum := 0
for i = 1 to n
  for j = 1 to i*i
    for k = 1 to j
      sum := sum + 1
```

- **Note that**
  - i can grow up to n
  - j can grow up to $i^2=n^2$
  - k can grow up to $j=n^2$
- **Going from the inner loop out we get**
  - $O(n * n^2 * n^2) = O(n^5)$

Solutions

```
sum := 0
for i = 1 to n
  for j = 1 to i*i
    if j mod i = 0 then
    for k = 1 to j
      sum := sum + 1
```

- **Note that**
  - i can grow up to n
  - j can grow up to $i^2=n^2$
  - k can grow up to $j=n^2$
- **However, if $j=i, 2i, 3i, \ldots, i^2$ then the inner loop is executed up to n^2 times, else inner loop is not executed**
  - For any i there are in^2 executions
  - $T(n) = \ = O(n^4)$

Summary

- Common growth rates
- Comparing growth rates
- Asymptotic lower bounds: Ω-notation
- Asymptotic tight bounds: Θ-notation
- Non-tight upper bounds: o-notation