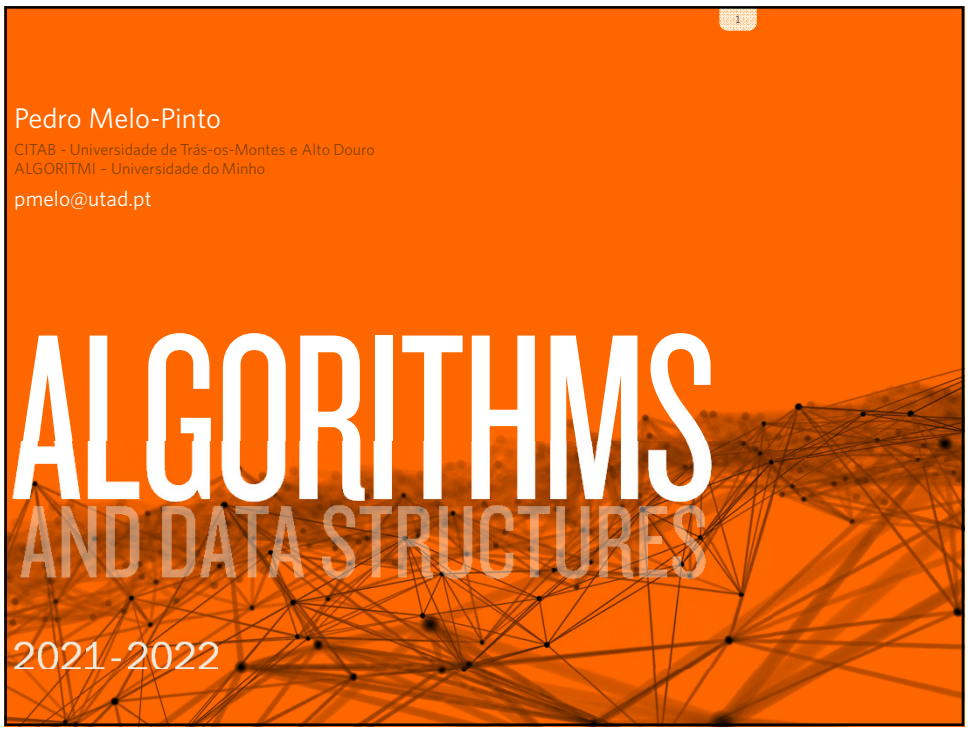
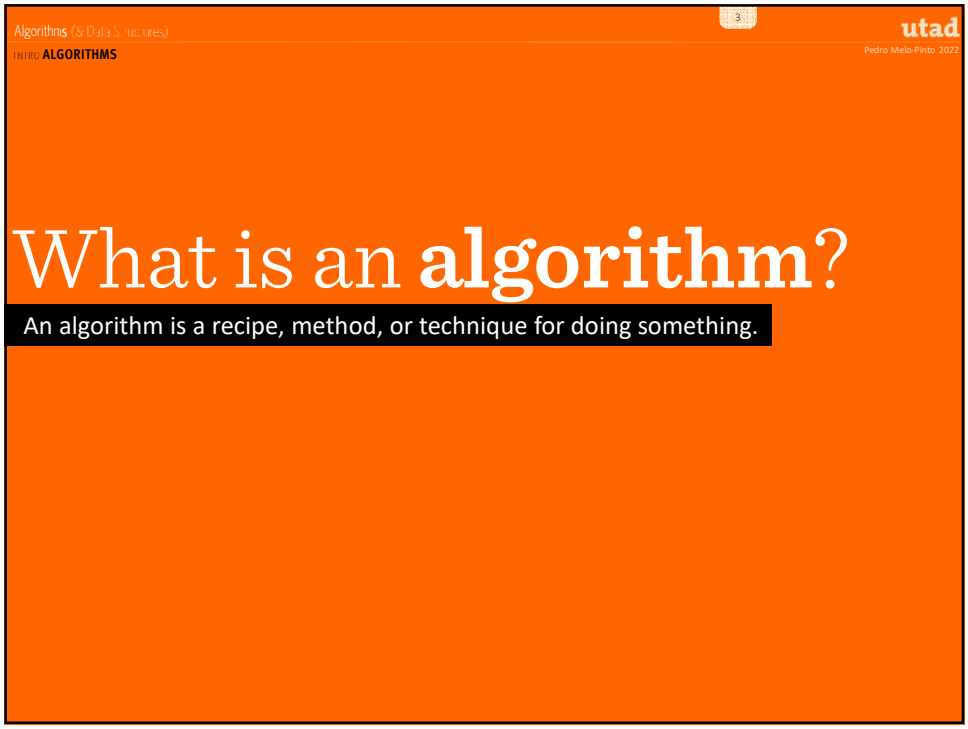
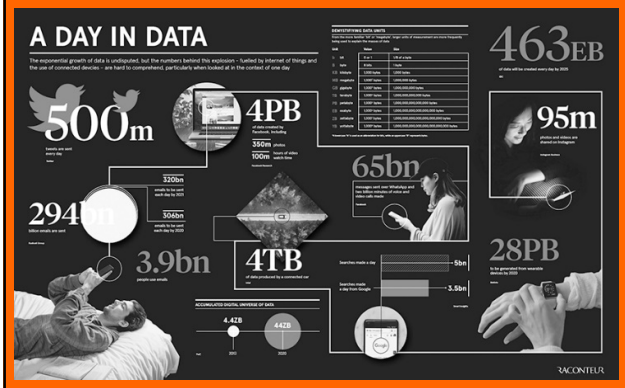

1
Pedro Melo-Pinto
CITAB - Universidade de Trás-os-Montes e Alto Douro
ALGORITMI – Universidade do Minho
pmelo@utad.pt
ALGORITHMS
AND DATA STRUCTURES
2021-2022


2
Algorithms (& Data Structures)
utad
Pedro Melo-Pinto 2022
AL01
INTRODUCTION


3
Algorithms (& Data Structures)
INTRO ALGORITHMS
utad
Pedro Melo-Pinto 2022
What is an algorithm?
An algorithm is a recipe, method, or technique for doing something.


4
Algorithms (& Data Structures)
INTRO ALGORITHMS
utad
Pedro Melo-Pinto 2022
A NEW
REALITY

A NEW REALITY



Each day (2019):
500 million tweets are sent
294 billion emails are sent (306bn by 2020; 320bn by 2021)
4 petabytes of data created by Facebook
4 terabytes of data produced by a connected car
65 billion messages sent over WahtsApp
5 billion searches made a day

(Raconteur, https://www.raconteur.net/infographics/a-day-in-data)

Today's society is based on the information and knowledge extracted (also) from the available data.

Never as of today so many data have been produced - around 1,7 MB per person each second, representing over 2,5 exabytes (2,5 x 10^{18} bytes) per day.

The accumulated digital universe of data is of 44 zettabytes (44 x 10^{21} bytes) in 2020, and a total of 463 exabytes of data is estimated to be produced every day by 2025.

how much data is produced every day?

2.5 Exabytes (10^{18}bytes = 1 million terabytes)

or

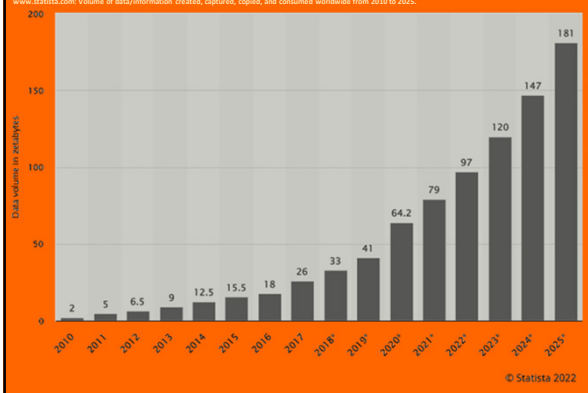
5 million laptops

250 000 Libraries of Congress

90 yrs of HD video

Level, Northeastern University, 2016

www.statista.com: Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025.



DATA

That is set to rise steeply to **181** zettabytes by 2025.

Arne von See. 2021

What is an **algorithm**?

"Algorithms are the ideas behind computer programs."

Steven Skiena (1997)

What is an **algorithm**?

An algorithm is a recipe, method, or technique for doing something.

Made up of a finite set of rules or operations that are unambiguous and simple to follow.

Wilson R.A. and Keil F.C. eds., 1999
The MIT Encyclopedia of the Cognitive Sciences

computer

What is a algorithm?

A **well-defined computational procedure** that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***.

A **sequence** of **computational steps** that transform the ***input*** into the ***output***.

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

computer

What is a algorithm?

A **well-defined computational procedure** that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***.

A **tool** for **solving** a well-specified **computational problem**.

problem ***specification*** includes what the input is, what the desired output should be;
the algorithm describes a specific computational procedure for achieving the desired output for a given input.

algorithm specification may be in English, as a computer program, even as a hardware design.

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

computer

Which are the most important **algorithms**?

For (computer) scientists

1. **A* search algorithm**
2. **Beam Search**
3. **Binary search**
4. Branch and bound
5. Buchberger's algorithm
6. **Data compression**
7. Diffie-Hellman key exchange
8. **Dijkstra's algorithm**
9. Discrete differentiation
10. **Dynamic programming**
11. Euclidean (greatest common divisor) algorithm
12. Expectation-maximization algorithm (EM-Training)
13. Fast Fourier transform (FFT)
14. Gradient descent
15. **Hashing**
16. **Heaps (heap sort)**
17. **Karatsuba multiplication**
18. Lenstra-Lenstra-Lovasz lattice reduction algorithm
19. **Maximum flow**
20. **Merge sort**

Which are the most important **algorithms**?

For job (coding) interviews

to implement:
Binary Search (with or without recursion)
Linear search
Bubble Sort
Insertion Sort
Quicksort (iterative or not)
MergeSort
Counting sort
Radix sort
Level Order Search in a Binary Tree
Swap two numbers without using the third variable
Check if Two String are Anagram
Sieve of Eratosthenes for Prime Number

to know:
Difference between stable and unstable sorting
Difference between Comparison and Non-Comparison
Sorting Depth First Search for a binary tree

example 1

Add the first *n* integers:

Input: A set of n integers $\{a_1, a_2, \ldots, a_n\}$.

Output: Start with sum=0. Add iteratively the value of each integer to the sum until there are no more elements.

An instance of the addition problem:	(input)	A set of 6 integers $\{1, 2, 3, 4, 5, 6\}$ $n = 6$.
Expected output for given instance:	(output)	21.

example 1

Add the first *n* integers:

Or add the numbers in pairs, observing that

1 + 100 = 101
2 + 99 = 101
3 + 98 = 101
...
48 + 53 = 101
49 + 52 = 101
50 + 51 = 101

The total is 50 * 101, which is 5050.

When we are looking for the sum of a sequence, we call it a series.

There is a special formula we can use to find the sum of a series: $S = \frac{n(n+1)}{2}$

example 2

Multiplying Integers :

Input: A set of *n* integers $\{a_1, a_2, \ldots, a_n\}$.

Output: Adding as many copies of one of them, as the value of the other one. Don't forget that $a_1 \times a_2 \times a_3 \Leftrightarrow (a_1 \times a_2) \times a_3$

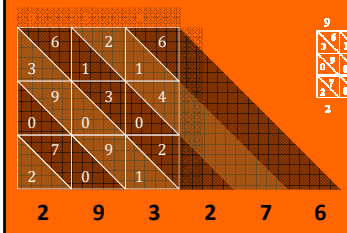
Rule 1: The product of a positive integer and a negative integer is a negative integer.
Rule 2: The product of two negative integers or two positive integers is a positive integer.

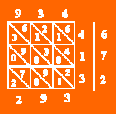
An instance of the multiplying problem:	(input)	A set of 4 integers {3, 10, 4, 2} or [(3 x 10) x 4] x 2.
Expected output for given instance:	(output)	240.

example 2

Multiplying Integers :

Ok. But how do we actually do it for, let's say, 934 x 314 ?





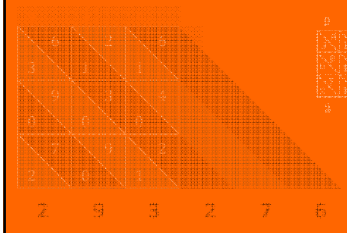
934 x 314 = 293276

Anonymous, 1478
Arte dell'Abbaco. Treviso

example 2

Multiplying Integers :

Ok. But how do we actually do it for, let's say, 934 x 314 ?



934 x 314 = 293276

Is it fast?

Algorithms (& Data Structures) 19 utad
INTRO ALGORITHMS Pedro Melo-Pinto 2022

example 2

Multiplying Integers :

Is it fast?

We tell that multiplication is $O(n^2)$, or that it runs in time $O(n^2)$

Big-Oh notation indicates how the running time scales with the size of the input

Algorithms (& Data Structures) 20 utad
INTRO ALGORITHMS Pedro Melo-Pinto 2022

example 2

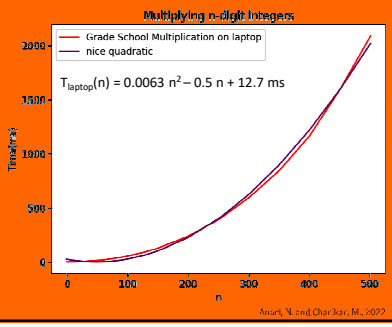
Multiplying Integers :

Is it fast?

Runtime *scales like* n^2

Big-Oh notation indicates how the running time scales with the size of the input

Can we do better?



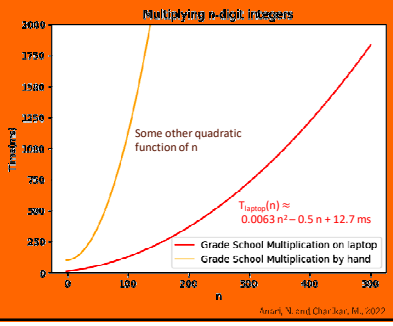
example 2

Multiplying Integers :

Not if we do it by hand

Is it fast?

Runtime *scales like* n^2



Big-Oh notation indicates how the running time scales with the size of the input

example 2

Multiplying Integers :

What if we divide the problem into smaller ones?

divide-and-conquer

Algorithms (& Data Structures) 23 utad
INTRO ALGORITHMS Pedro Melo-Pinto 2022

example 2

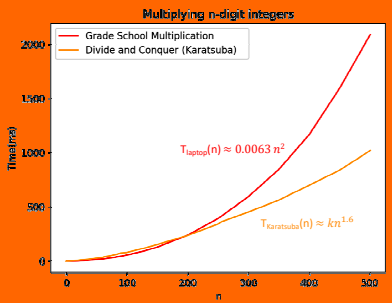
Multiplying Integers :

Karatsuba's Algorithm

$x \times y = (a \times c) \times 10^n + (z - a \times c - b \times d) \times 10^{n/2} + (b \times d)$

$O(n^{1.6})$

Can we do better?



Algorithms (& Data Structures) 24 utad
INTRO ALGORITHMS Pedro Melo-Pinto 2022

example 2

Multiplying Integers :

Karatsuba (1962)
$O(n^{1.6})$

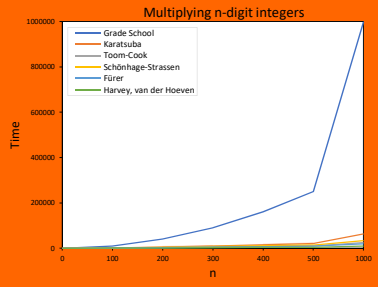
Toom-Cook (1963)
$O(n^{1.465})$

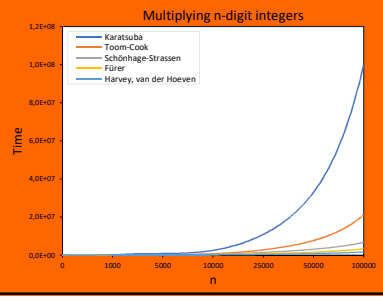
Schönhage-Strassen (1971)
$O(n \log(n) \log\log(n))$

Fürer (2007)
$O(n \log(n) \times 2^{O(\log^*(n))})$

Harvey, van der Hoeven (2019)
$O(n \log(n))$

use Fast Fourier Transform (FFT) to compute polynomial multiplication





example 3

The Sorting Problem :

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation or reordering $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

An instance of the sorting problem:	(input)	A sequence of 6 numbers (31, 41, 59, 26, 41, 58).
Expected output for given instance:	(output)	The permutation of the input (26, 31, 41, 41, 58 , 59).

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

example 3

sorting is a fundamental operation in computer science
(many programs use it as an intermediate step)

we have a large number of good sorting algorithms at our disposal

the best algorithm for a given application depends on
- the number of items to be sorted
- the extent to which the items are already somewhat sorted
- possible restrictions on the item values
- the architecture of the computer
- the kind of storage devices to be used
- and others

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

How can we present an **algorithm**?

Visual, diagrammatic techniques are one way of presenting the control flow of an algorithm in a clear and readable fashion. There are various ways of "drawing" algorithms, as opposed to writing them down.

One of the best known of these involves writing the elementary instructions in rectangular boxes and the tests in diamond shaped ones, and using arrows to describe how the processor runs around executing the algorithm.
The resulting objects are called **flowcharts**.

Harel, D., 2004
Algorithmics, The Spirit of Computing.

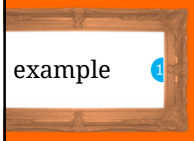
How can we present an **algorithm**?

Or we can write them down:
using a natural language as we do everyday communicating with each other,
or using a more "computer-like" type of language called pseudocode.

How can we present an **algorithm**?

Or we can write them down:
using a natural language as we do everyday communicating with each other,
or using a more "computer-like" type of language called pseudocode.

Informal high-level description of an algorithm:
an outline of an algorithm, written in a form that can easily be converted into real programming statements.



How can we present an **algorithm**?

Linear equation (with one unknown) root :

Input: A linear equation $ax + b = 0$, where a and b are constants and $a \neq 0$.
Output: The root value.

example 1

How can we present an **algorithm**?

NATURAL LANGUAGE

Linear equation (with one unknown) root :

Input: A linear equation $ax + b = 0$, where a and b are constants and $a \neq 0$.

Output: The root value.

1. program begins
2. read a and b
3. if a is different from zero, then
4. calculate x value as x = -b/a
5. print x value
6. else
7. print "either every number is a solution of the equation or the equation is inconsistent"
8. program ends

example 1

How can we present an **algorithm**?

PSEUDOCODE

Linear equation (with one unknown) root :

Input: A linear equation $ax + b = 0$, where a and b are constants and $a \neq 0$.

Output: The root value.

1. BEGIN
2. READ a, b
3. IF a ≠ 0 THEN
4. x ← -b/a
5. PRINT x
6. ELSE
7. PRINT "either every number is a solution …"
8. END

example 1

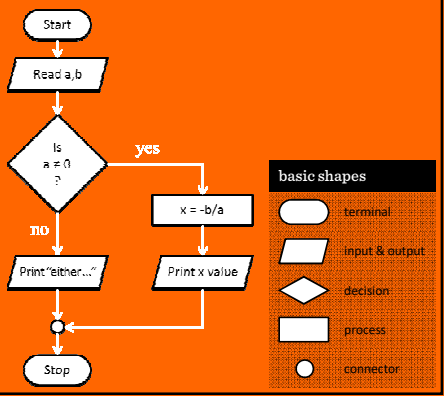
How can we present an **algorithm**?

FLOWCHART

Linear equation (with one unknown) root :

Input: A linear equation $ax + b = 0$, where a and b are constants and $a \neq 0$.

Output: The root value.



What we want to know:

does it work?
is it fast?
can we do better?

What we want to know:

does it work?
is it fast?
can we do better?
can we do it right?

embedded ethics

Types of problems

Many problems can be solved quickly

(i.e., in close to linear time, or at least a time that is some small polynomial function of the input size)

Types of problems

However, there are problems for which
no efficient solution is known

intractable problems
As they grow large, we are unable to solve them in reasonable time

Types of problems

What constitutes reasonable time?

Standard working definition: **polynomial time**

on an input of size n
the worst-case running tim
for some constant k

Types of problems

There are *intractable* problems

no efficient solution is known

As they grow large, we are unable to solve them in reasonable time

n	log(n)	$log_2(n)$	$log_{10}(n)$	n	n^2	n^3	2^n	n^n	n!
1	0,00E+00	0,00E+00	0,00E+00	2,50E-10	2,50E-10	2,50E-10	5,00E-10	2,50E-10	2,50E-10
10	5,76E-10	8,30E-10	2,50E-10	2,50E-09	2,50E-08	2,50E-07	2,56E-07	2,50	9,07E-04
100	1,15E-09	1,66E-09	5,00E-10	2,50E-08	2,50E-06	2,50E-04	3,17E+20	2,50E+190	2,33E+148
200	1,32E-09	1,91E-09	5,75E-10	5,00E-08	1,00E-05	2,00E-03	4,02E+50	#NUM!	#NUM!
300	1,43E-09	2,06E-09	6,19E-10	7,50E-08	2,25E-05	0,01	5,09E+80	#NUM!	#NUM!
400	1,50E-09	2,16E-09	6,51E-10	1,00E-07	4,00E-05	0,02	6,46E+110	#NUM!	#NUM!
500	1,55E-09	2,24E-09	6,75E-10	1,25E-07	6,25E-05	0,03	8,18E+140	#NUM!	#NUM!
600	1,60E-09	2,31E-09	6,95E-10	1,50E-07	9,00E-05	0,05	1,04E+171	#NUM!	#NUM!
700	1,64E-09	2,36E-09	7,11E-10	1,75E-07	1,23E-04	0,09	1,32E+201	#NUM!	#NUM!
800	1,67E-09	2,41E-09	7,26E-10	2,00E-07	1,60E-04	0,13	1,67E+231	#NUM!	#NUM!
900	1,70E-09	2,45E-09	7,39E-10	2,25E-07	2,03E-04	0,18	2,11E+261	#NUM!	#NUM!
1000	1,73E-09	2,49E-09	7,50E-10	2,50E-07	2,50E-04	0,25	2,68E+291	#NUM!	#NUM!
10000	2,30E-09	3,32E-09	1,00E-09	2,50E-06	0,03	250,00	#NUM!	#NUM!	#NUM!
100000	2,88E-09	4,15E-09	1,25E-09	2,50E-05	2,50	250000,00	#NUM!	#NUM!	#NUM!
1000000	3,45E-09	4,98E-09	1,50E-09	0,0003	250,00	250000000,00	#NUM!	#NUM!	#NUM!

growth of different functions of n (think of n as the number of items to be processed)

Types of problems

There are *intractable* problems

no efficient solution is known

As they grow large, we are unable to solve them in reasonable time

possible solutions:

- **A Use a heuristic**
- **B Solve approximately**
- **C Use an exponential time solution**

Types of problems

optimization **vs** decision

An optimization problem tries to find an optimal solution
A decision problem tries to answer a yes/no question

Many problems will have decision and optimization versions

Some problems are decidable, but *intractable*:
as they grow large, we are unable to solve them in reasonable time

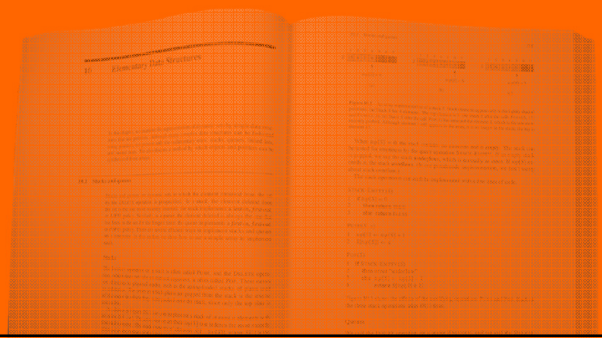
Types of problems

class **P** (polynomial-time)

The class of decision problems that have
polynomial-time deterministic algorithms.

These type of problems are called tractable
Problems not in P are **intractable or unsolvable**

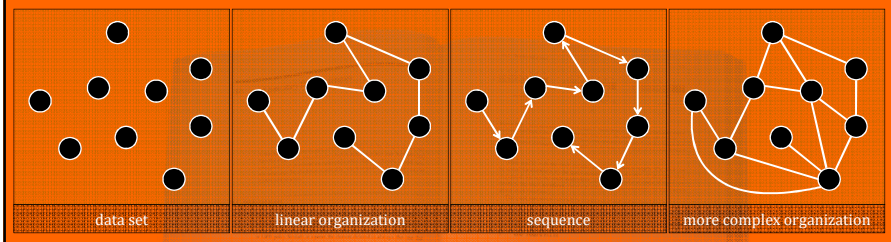
Algorithms & data structures
Every algorithm **needs data**



Algorithms & data structures

We will use several data structures.

A data structure is a way to store and organize data in order to facilitate access and modifications.



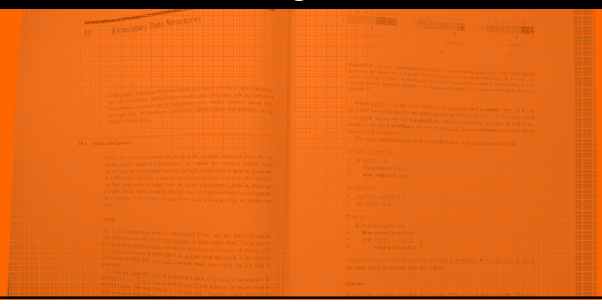
Algorithms & data structures

We will use several data structures.

A data structure is a way to store and organize data in order to facilitate access and modifications.

No single data structure works well for all purposes.

It is important to know the strengths and limitations of several of them.



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

AL**02**

ANALYSIS OF ALGORITHMS

Back to the sorting example

example

The Sorting Problem :

Input: A sequence of n numbers $(a_1, a_2, \ldots, a_n)$.

Output: A permutation or reordering $(a'_1, a'_2, \ldots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

An instance of the sorting problem: (input) A sequence of 6 numbers (31, 41, 59, 26, 41, 58).

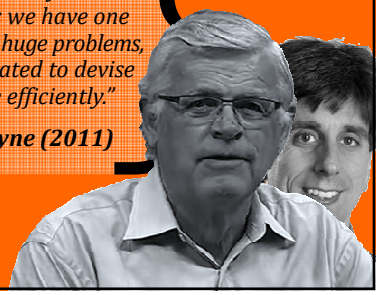
Expected output for given instance: (output) The permutation of the input (26, 31, 41, 41, 58, 59).

we have a large number of good sorting algorithms at our disposal

Why to choose an algorithm?

"When we use a computer to help us solve a problem, we typically are faced with a number of possible approaches. For small problems, it hardly matters which approach we use, as long as we have one that correctly solves the problem. For huge problems, however, we quickly become motivated to devise methods that use time and space efficiently."

Robert Sedgewick & Kevin Wayne (2011)



Why to choose an algorithm?

Correcteness & **Efficiency**

Time efficiency
Space (resources) **efficiency**

Algorithm correctness

An algorithm is said to be ***correct*** if, for every input instance, it halts with the correct output.

Wilson R.A. and Keil F.C. eds. 1999
The MIT Encyclopedia of the Cognitive Sciences

we have a large number of good sorting algorithms at our disposal

HOW TO CHOOSE ONE ?

The choice of the best algorithm for a particular task can be a complicated process, perhaps involving sophisticated mathematical analysis. The branch of computer science that comprises the study of such questions is called *analysis of algorithms*.

Sedgewick R., Wayne K., 2011
Algorithms.

How to analyse an algorithm?

A. Develop a good implementation.
B. Identify unknown quantities (q) representing the basic operations.
C. Determine the cost of each basic operation.
D. Develop a realistic model for the input.
E. Analyse the frequency of execution of the unknown quantities.
F. Calculate the total running time: $\sum_q frequency(q) \times cost(q)$

Donald Knuth (1960s)

Sedgewick R., Flajolet P., 2013
An Introduction to the Analysis of Algorithms.

How to analyse an algorithm?

A. Analyse worst-case cost.
B. Use O-notation for upper bound.

→ **Classify algorithms by these costs.**

Aho, Hopcroft and Ullman (1970s)
Cormen, Leiserson, Rivest and Stein (present day)

Sedgewick R., Flajolet P., 2013
An Introduction to the Analysis of Algorithms.

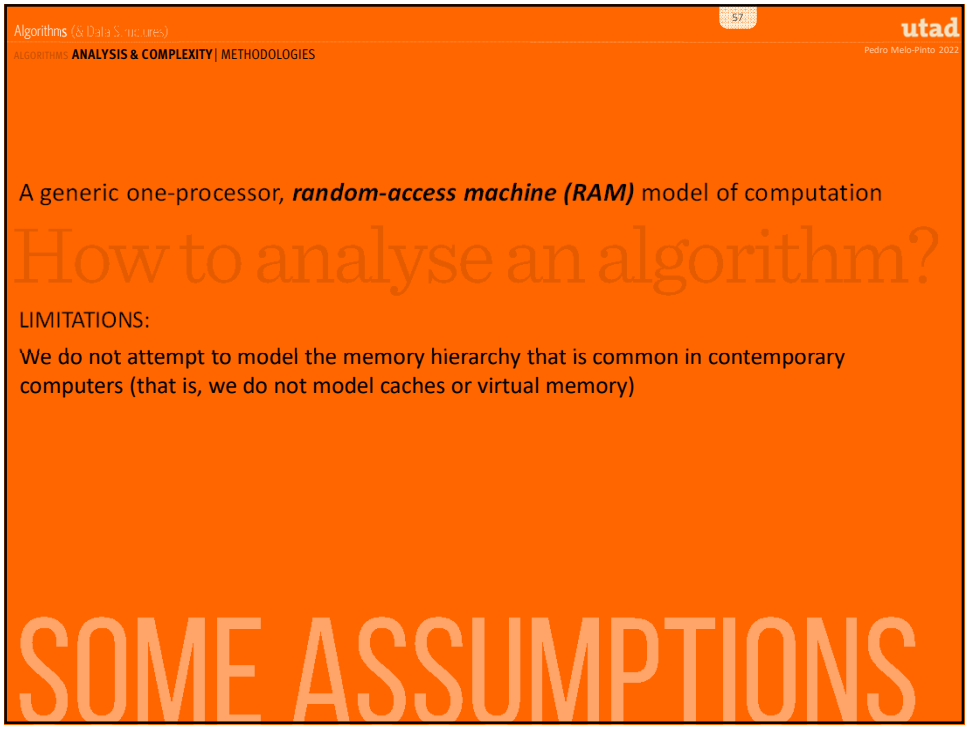
A generic one-processor, ***random-access machine (RAM)*** model of computation

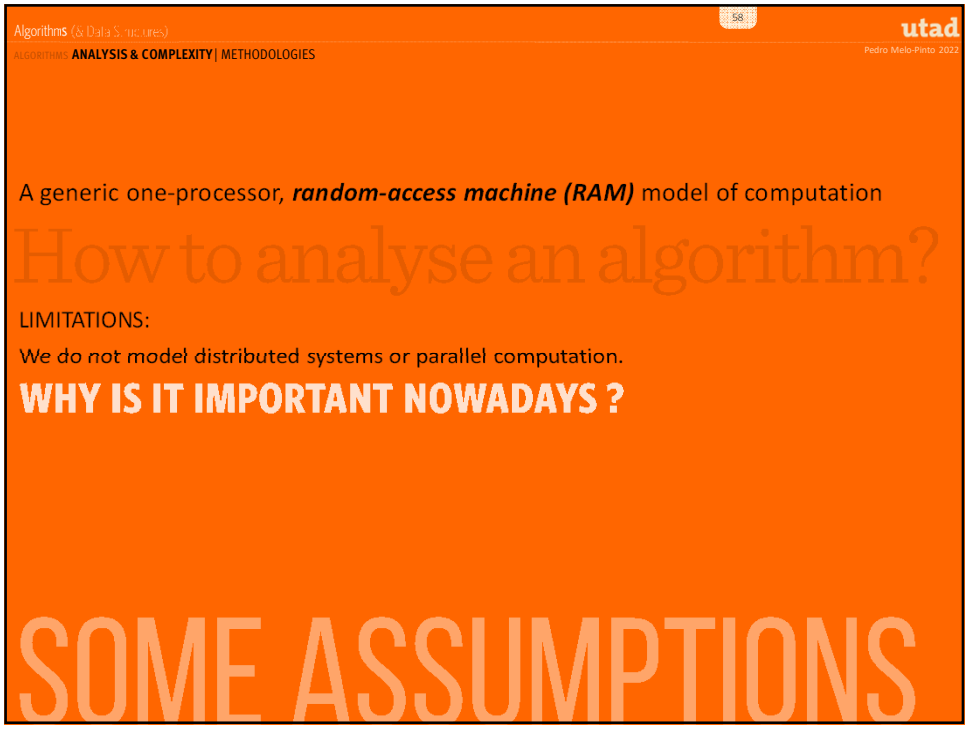
How to analyse an algorithm?

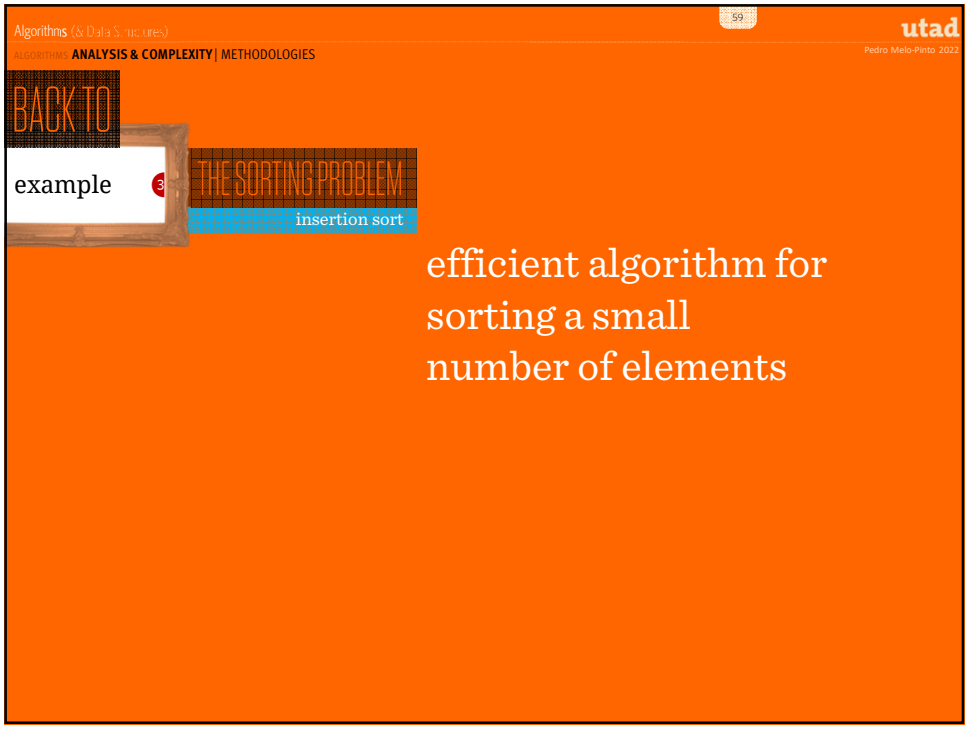
Our algorithms will be implemented as computer programs
In the RAM model, instructions are executed one after another, with no concurrent operations

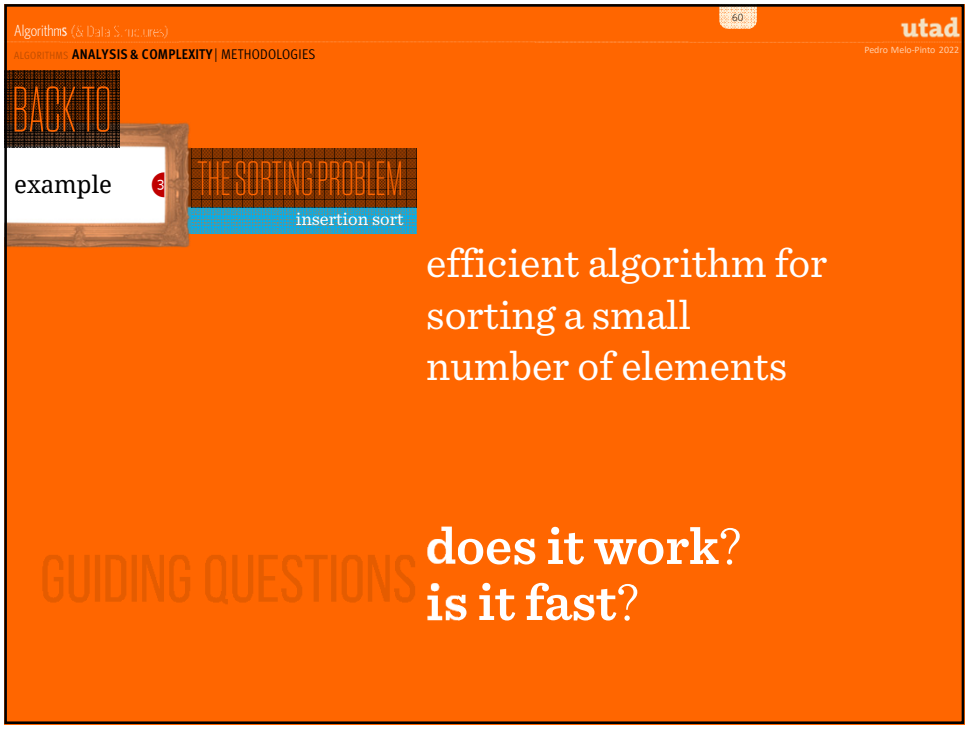
Our guide is how real computers are designed
The RAM model contains instructions commonly found in real computers:
arithmetic (such as add, subtract, multiply, divide, remainder, floor, ceiling), data movement (load, store, copy), and control (conditional and unconditional branch, subroutine call and return)
Each such instruction takes a constant amount of time

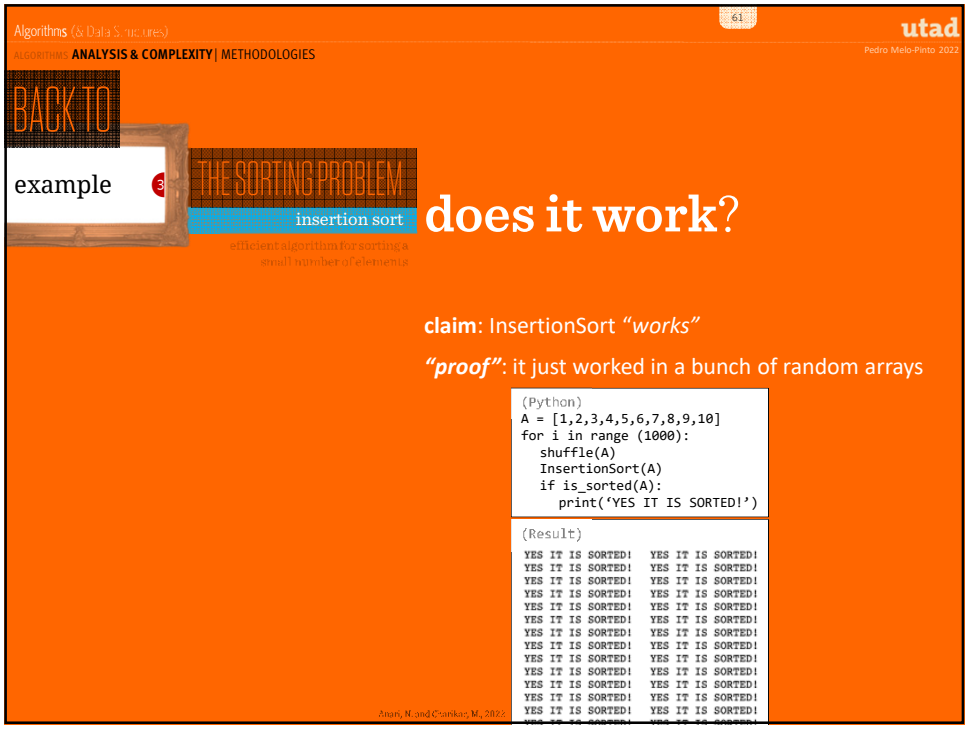
SOME ASSUMPTIONS

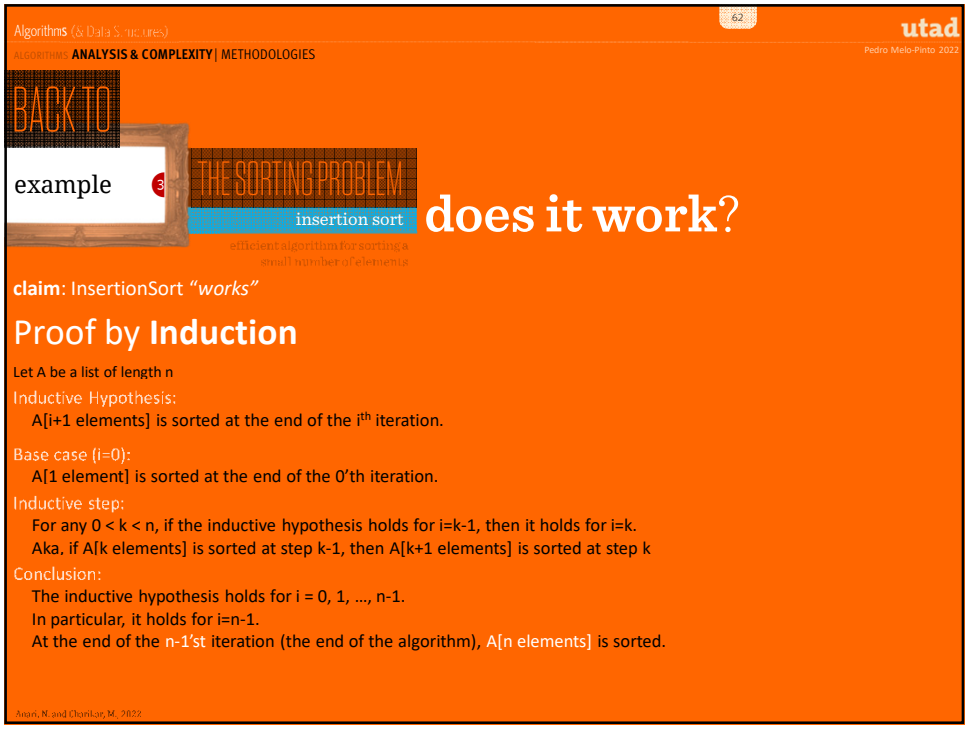

57
Algorithms (& Data Structures)
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
utad
Pedro Melo-Pinto 2022
A generic one-processor, random-access machine (RAM) model of computation
How to analyse an algorithm?
LIMITATIONS:
We do not attempt to model the memory hierarchy that is common in contemporary computers (that is, we do not model caches or virtual memory)
SOME ASSUMPTIONS


58
Algorithms (& Data Structures)
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
utad
Pedro Melo-Pinto 2022
A generic one-processor, random-access machine (RAM) model of computation
How to analyse an algorithm?
LIMITATIONS:
We do not model distributed systems or parallel computation.
WHY IS IT IMPORTANT NOWADAYS ?
SOME ASSUMPTIONS


Algorithms (& Data Structures)
59
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
BACK TO
example 3
THE SORTING PROBLEM
insertion sort
efficient algorithm for
sorting a small
number of elements


Algorithms (& Data Structures)
60
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
BACK TO
example 3
THE SORTING PROBLEM
insertion sort
efficient algorithm for
sorting a small
number of elements
GUIDING QUESTIONS
does it work?
is it fast?


Algorithms (& Data Structures)
61
utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
Pedro Melo-Pinto 2022
BACK TO
example 3
THE SORTING PROBLEM
insertion sort
efficient algorithm for sorting a small number of elements
does it work?
claim: InsertionSort "works"
"proof": it just worked in a bunch of random arrays
(Python)
A = [1,2,3,4,5,6,7,8,9,10]
for i in range (1000):
shuffle(A)
InsertionSort(A)
if is_sorted(A):
print('YES IT IS SORTED!')
(Result)
YES IT IS SORTED!
Anaí, N. and Chorikov, M., 2022

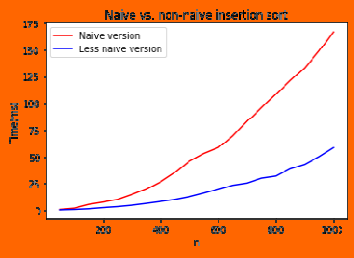

Algorithms (& Data Structures)
62
utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
Pedro Melo-Pinto 2022
BACK TO
example 3
THE SORTING PROBLEM
insertion sort
efficient algorithm for sorting a small number of elements
does it work?
claim: InsertionSort "works"
Proof by Induction
Let A be a list of length n
Inductive Hypothesis:
A[i+1 elements] is sorted at the end of the ith iteration.
Base case (i=0):
A[1 element] is sorted at the end of the 0'th iteration.
Inductive step:
For any 0 < k < n, if the inductive hypothesis holds for i=k-1, then it holds for i=k.
Aka, if A[k elements] is sorted at step k-1, then A[k+1 elements] is sorted at step k
Conclusion:
The inductive hypothesis holds for i = 0, 1, …, n-1.
In particular, it holds for i=n-1.
At the end of the n-1'st iteration (the end of the algorithm), A[n elements] is sorted.
Anaí, N. and Chorikov, M., 2022

Algorithms (& Data Structures) 63 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

BACK TO example 3 THE SORTING PROBLEM insertion sort

efficient algorithm for sorting a small number of elements

is it fast?



The "same" algorithm can be slower or faster depending on the implementations.

It can also be slower or faster depending on the hardware that we run it on.

and...

EFFICIENCY DEPENDS ON THE INPUT

Algorithms (& Data Structures) 64 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

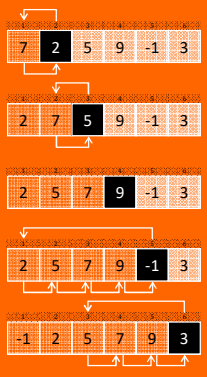
BACK TO example 3 THE SORTING PROBLEM insertion sort

efficient algorithm for sorting a small number of elements

EFFICIENCY DEPENDS ON THE INPUT

BACK TO example 3 THE SORTING PROBLEM
insertion sort

Theory of Algorithms
concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")



PROGRAM

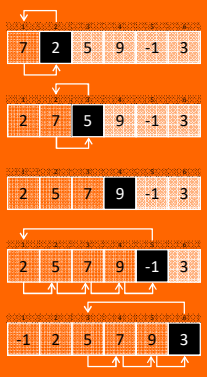
```
void InsertionSort(int v[N], int n)
{
     int i,j,aux;

     for(i=2;i<=n;i++)
     {
          aux = v[i-1];
          j = i-1;
          while(j>0 && v[j-1]>aux)
          {
               v[j] = v[j-1];
               j = j - 1;
          }
          v[j] = aux;
     }
}
```

Let's start with Knuth's approach

BACK TO example 3 THE SORTING PROBLEM
insertion sort

Theory of Algorithms
concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")



PROGRAM

```
void InsertionSort(int v[N], int n)
{
     int i,j,aux;

     for(i=2;i<=n;i++)
     {
          aux = v[i-1];
          j = i-1;
          while(j>0 && v[j-1]>aux)
          {
               v[j] = v[j-1];
               j = j - 1;
          }
          v[j] = aux;
     }
}
```

Ⓐ the running time of the algorithm is the sum of running times for each statement executed

Ⓑ each statement i is executed in a constant time c_i

Ⓒ a statement that takes c_i to execute and executes n times will contribute $c_i \times n$ to the total running time

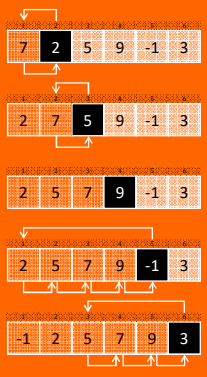
BACK TO example 3

THE SORTING PROBLEM

insertion sort

Theory of Algorithms

concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")



PROGRAM

```
void InsertionSort(int v[N], int n)
{
     int i,j,aux;
     for(i=2;i<=n;i++)
     {
          aux = v[i-1];
          j = i-1;
          while(j>0 && v[j-1]>aux)
          {
               v[j] = v[j-1];
               j = j - 1;
          }
          v[j] = aux;
     }
}
```

COST	TIMES
c_1	n
c_2	n-1
c_3	n-1
c_4	$\sum_{i=2}^{n} t_i$
c_5	$\sum_{i=2}^{n}(t_i - 1)$
c_6	$\sum_{i=2}^{n}(t_i - 1)$
c_7	n-1

n : #elements
t_i : n. iterations of while loop for each i

running time :

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=2}^{n} t_i + c_5\sum_{i=2}^{n}(t_i - 1) + c_6\sum_{i=2}^{n}(t_i - 1) + c_7(n-1)$$

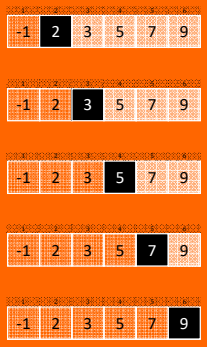
BACK TO example 3

THE SORTING PROBLEM

insertion sort

Theory of Algorithms

concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")



PROGRAM

```
void InsertionSort(int v[N], int n)
{
     int i,j,aux;
     for(i=2;i<=n;i++)
     {
          aux = v[i-1];
          j = i-1;
          while(j>0 && v[j-1]>aux)
          {
               v[j] = v[j-1];
               j = j - 1;
          }
          v[j] = aux;
     }
}
```

best case:

input array is already sorted

$t_i = 1$

running time :

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

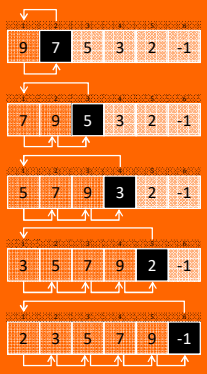
BACK TO example 3

THE SORTING PROBLEM

insertion sort

Theory of Algorithms

concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")



PROGRAM

```
void InsertionSort(int v[N], int n)
{
     int i,j,aux;

     for(i=2;i<=n;i++)
     {
          aux = v[i-1];
          j = i-1;
          while(j>0 && v[j-1]>aux)
          {
               v[j] = v[j-1];
               j = j - 1;
          }
          v[j] = aux;
     }
}
```

worst case:

input array is sorted in reverse order

$t_i = i$

running time:

$$T(n) = \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

Most of the times, we do worst case analysis to analyse algorithms. In the worst case analysis, we guarantee an upper bound on the running time of an algorithm.

The average case analysis is not easy to do in most of the practical cases since we must know or predict the mathematical distribution of all possible inputs.

The best case analysis is bogus. Guaranteeing a lower bound for an optimal case still allow for an algorithm to take years to run with a different input.

BACK TO example 3 THE SORTING PROBLEM
insertion sort

Theory of Algorithms
concentrates on determining the growth of the worst-case performance of the algorithm (an "upper bound")

Counting individual operations:

It's very tedious!

In order to use this to understand running time, we need to know how long each operation takes, plus a whole bunch of other stuff...

Asymptotic notation

Gives us a meaningful way to talk about the running time of an algorithm, independent of programming language, computing platform, etc., without having to count all the operations.

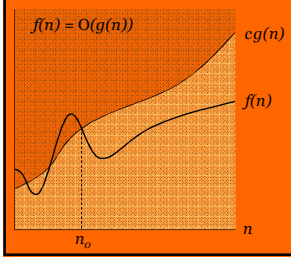
Mathematical notations Big-Oh

asymptotic **upper** bound

Definition Given a function $g(n)$,

$\mathbf{O}(g(n))$ denotes the set of all $f(n)$ such that $\left|\frac{f(N)}{g(N)}\right|$ is bounded from above as $n \to \infty$.
(*asymptotically* bounds a function from above)

$\mathbf{O}(g(n)) = \{ f(n) : \exists c > 0, n_0 > 0$ such that $\forall n \geq n_0 : f(n) \leq c\, g(n) \}$
O-notation provides a way to express an upper bound.



Sedgewick R., Flajolet P., 2013 An introduction to the Analysis of Algorithms.
Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009 Introduction to Algorithms.

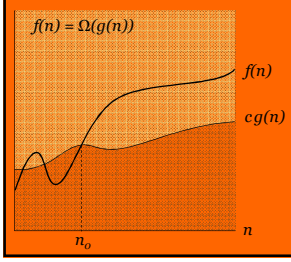
Mathematical notations Big-Omega

asymptotic **lower** bound

Definition Given a function $g(n)$,

$\mathbf{\Omega}(g(n))$ denotes the set of all $f(n)$ such that $\left|\frac{f(N)}{g(N)}\right|$ is bounded from below by a (strictly) positive number as $n \to \infty$.

$\mathbf{\Omega}(g(n)) = \{ f(n) : \exists c > 0, n_0 > 0$ such that $\forall n \geq n_0 : f(n) \geq c\, g(n) \}$
Ω-notation provides a way to express a lower bound.



Sedgewick R., Flajolet P., 2013 An introduction to the Analysis of Algorithms.
Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009 Introduction to Algorithms.

Mathematical notations Big-Theta

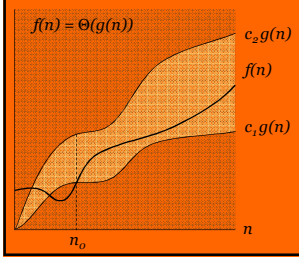
asymptotically **tight** bound

Definition Given a function $g(n)$,

$\Theta(g(n))$ denotes the set of all $f(n)$ such that $\left|\frac{f(N)}{g(N)}\right|$ is bounded from both above and below as $n \to \infty$.

$\Theta(g(n)) = \{ f(n) : \exists c_1, c_2 > 0, n_0 > 0$ such that $\forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

Θ-notation provides a way to express matching upper and lower bounds.



Sedgewick R., Flajolet P., 2013 An introduction to the Analysis of Algorithms.

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

Mathematical notations Big-Oh

asymptotic **upper** bound

Suppose the running time of an algorithm is:

$$T(n) = 10\,n^2 + 3n + 7 \text{ ms}$$

this constant factor of 10 depends a lot on the computing platform...

these lower-order terms don't really matter as n gets large.

Mathematical notations Big-Oh

asymptotic **upper** bound

Suppose the running time of an algorithm is:

$$T(n) = 10\,n^2 + 3n + 7 \text{ ms}$$

We're just left with the n^2 term!
That's what's meaningful.

$$T(n) = O(n^2)$$

BACK AGAIN TO

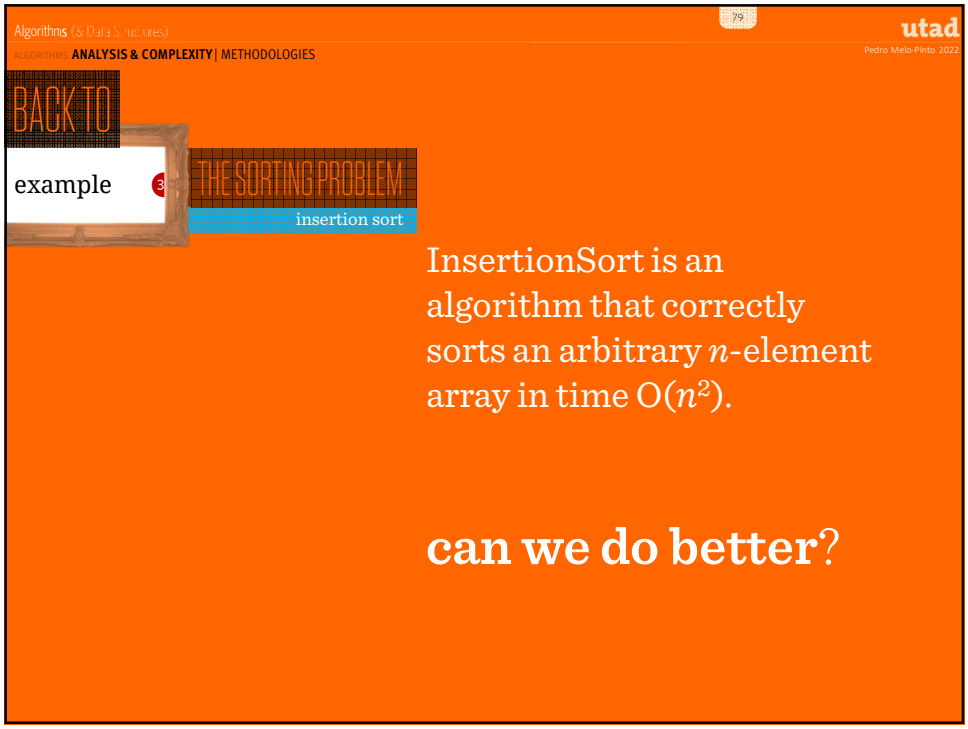
example 1 THE SORTING PROBLEM

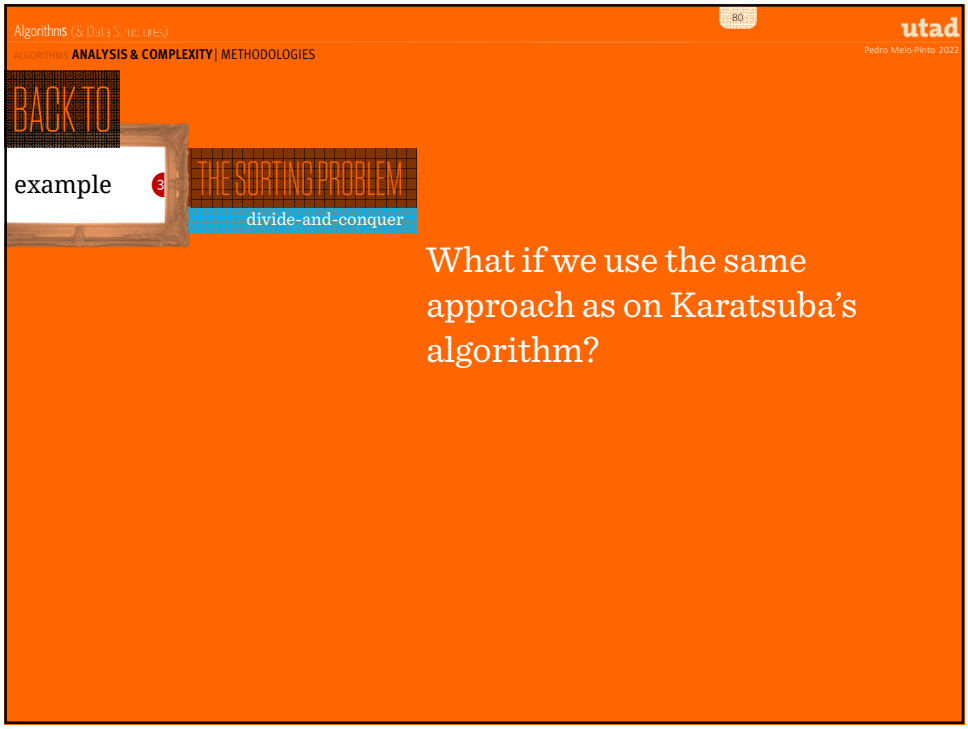
insertion sort

best case:
running time $T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$
is $\mathbf{O}(n)$

worst case:
running time $T(n) = \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$
is $\mathbf{O}(n^2)$

average case:
running time $T(n) = \left(\frac{c_4}{4} + \frac{c_5}{4} + \frac{c_6}{4}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{4} - \frac{c_5}{4} - \frac{c_6}{4} + c_7\right)n - \left(c_2 + c_3 + \frac{c_4}{2} + c_7\right)$
is $\mathbf{O}(n^2)$

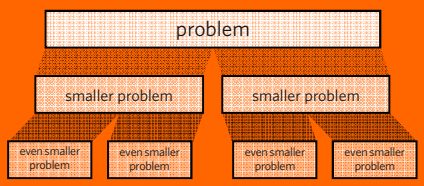

Algorithms (& Data Structures)
79
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
BACK TO
example 3
THE SORTING PROBLEM
insertion sort
InsertionSort is an algorithm that correctly sorts an arbitrary n-element array in time $O(n^2)$.
can we do better?


Algorithms (& Data Structures)
80
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
BACK TO
example 3
THE SORTING PROBLEM
divide-and-conquer
What if we use the same approach as on Karatsuba's algorithm?

Algorithms (& Data Structures) 81 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

BACK TO example 3 THE SORTING PROBLEM
divide-and-conquer

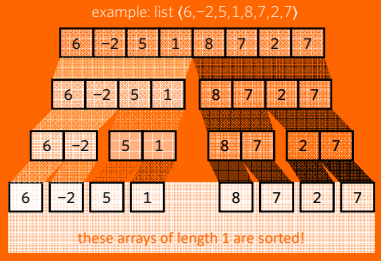
What if we use the same approach as on Karatsuba's algorithm?



Algorithms (& Data Structures) 82 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

BACK TO example 3 THE SORTING PROBLEM
mergesort

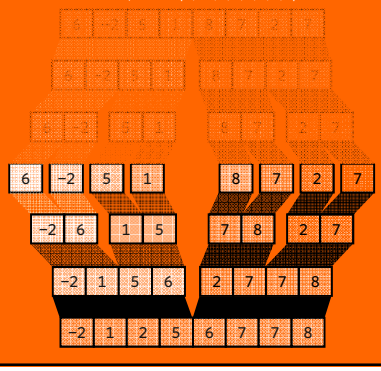
First, recursively break up the array all the way down to the base cases



BACK TO example 3 THE SORTING PROBLEM
mergesort

Then, merge them all back up!

example: list ⟨6,−2,5,1,8,7,2,7⟩



BACK TO example 3 THE SORTING PROBLEM
mergesort

does it work?
is it fast?

MERGESORT PSEUDOCODE

```
function mergesort
pass in: ARRAY INT myArray[ ]
var: n, do tipo INT

n ← size of myArray[]

if n ≤ 1
    pass out: myArray                                  if myArray has length 1,
end if                                                 it is already sorted

L ← call: mergesort(myArray[0,..,n/2-1])               sort the left half
R ← call: mergesort(myArray[n/2,..,n-1 ])              sort the right half

pass out: call: merge(L,R)                             merge the two halves
end function
```

Anon, N. and Cherliar, M., 2022

BACK TO example 3

THE SORTING PROBLEM mergesort

does it work? is it fast?

MERGESORT PSEUDOCODE

```
function mergesort
pass in: ARRAY INT myArray[ ]
var: n, do tipo INT
n ← size of myArray[]

if n ≤ 1
    pass out: myArray
end if

L ← call: mergesort(myArray[0,…,n/2-1])
R ← call: mergesort(myArray[n/2,…,n-1])

pass out: call: merge(L,R)
end function
```

claim: MergeSort *"works"*

Proof by **Induction**

Let A be a list of length n

Inductive Hypothesis:
In every recursive call on an array of length at most i, `mergesort` returns a sorted array.

Base case (i=1):
A[1 element] is always sorted.

Inductive step:
Need to show: if the inductive hypothesis holds for k<i, then it holds for k=i.
Aka, **need to show** that if L and R are sorted, then `merge(L,R)` is sorted

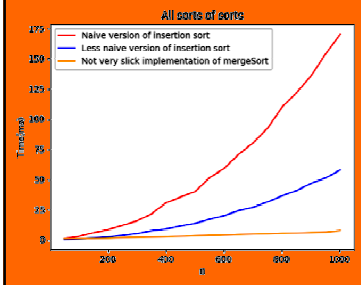
Conclusion:
In the top recursive call, `mergesort` returns a sorted array.

Anani, N. and Cherliker, M., 2022

BACK TO example 3

THE SORTING PROBLEM merging sort

does it work? is it fast?



Empirically:
Seems to work.
Seems fast.

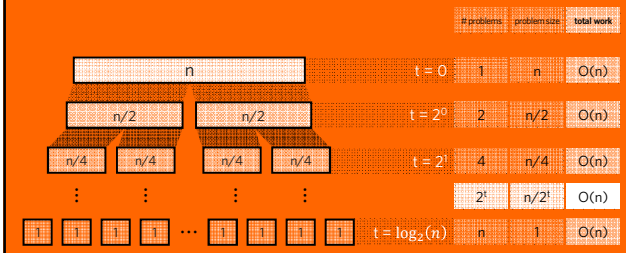
Anani, N. and Cherliker, M., 2022

Algorithms (& Data Structures) 87 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

BACK TO example 3 THE SORTING PROBLEM merging sort

does it work? is it fast?

claim: MergeSort runs in time $O(n \log(n))$



Total time: O(n log(n))

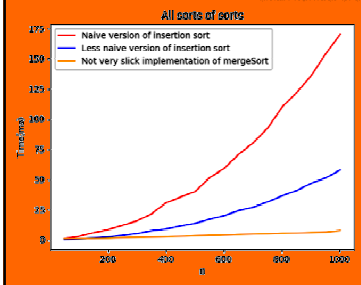
Adapted from Amari, N. and Charikar, M., 2022.

Algorithms (& Data Structures) 88 utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES Pedro Melo-Pinto 2022

BACK TO example 3 THE SORTING PROBLEM insertion sort

efficient algorithm for sorting a small number of elements

is it fast?



It´s a lot better than InsertionSort.

A running time of O(n log(n)) is a lot better than $O(n^2)$.

BACK TO example 3 THE SORTING PROBLEM merging sort

does it work? is it fast?

claim: MergeSort runs in time $O(n \log(n))$

We can analyze it in a different way (using recurrence relations)

BACK TO example 3 THE SORTING PROBLEM merging sort

does it work? is it fast?

01 The Master Theorem

Recurrence analysis

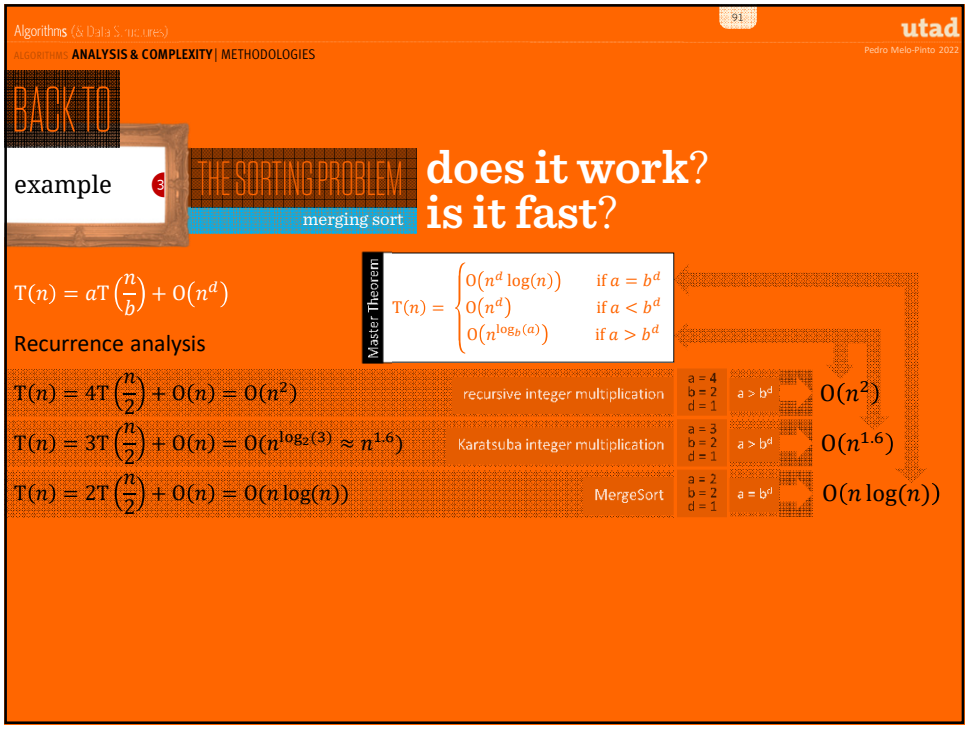
Suppose that $a \geq 1, b > 1,$ and d are constants (independent of n).

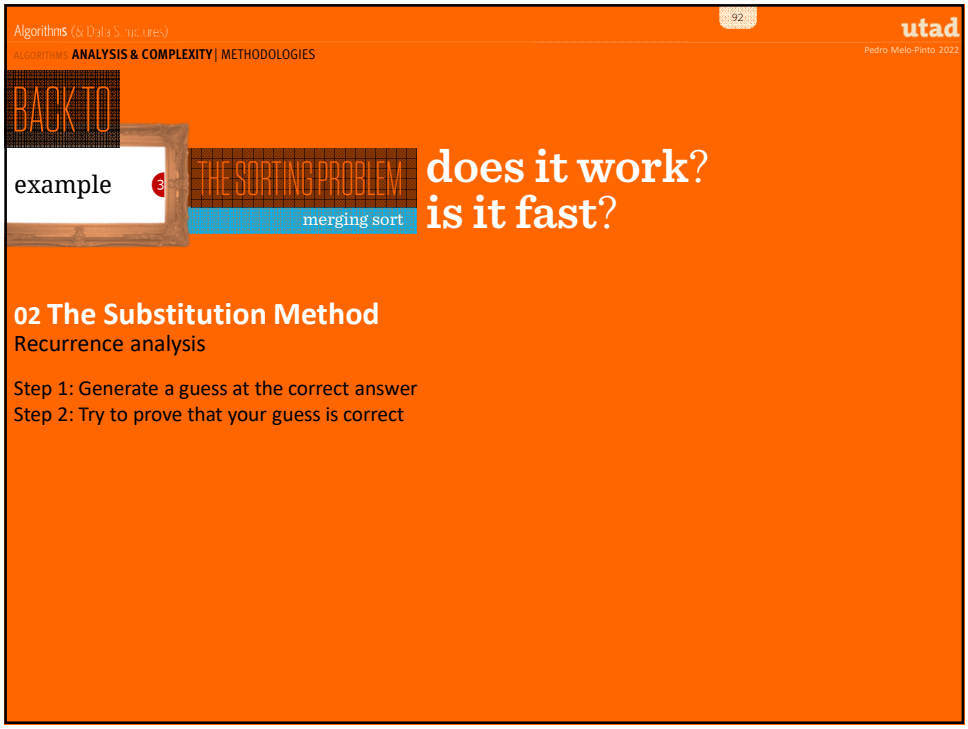
Suppose that $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$.

Then:

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Parameters:
a : number of sub-problems
b : factor by which input size shrinks
d : need to do n^d work to create all the sub-problems and combine their solutions.


Algorithms (& Data Structures)
91
utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
Pedro Melo-Pinto 2022
BACK TO
example 3
THE SORTING PROBLEM
merging sort
does it work?
is it fast?
$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$
Recurrence analysis
Master Theorem
$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$
$T(n) = 4T\left(\frac{n}{2}\right) + O(n) = O(n^2)$
recursive integer multiplication
a = 4 b = 2 d = 1
a > b^d
$O(n^2)$
$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2(3)} \approx n^{1.6})$
Karatsuba integer multiplication
a = 3 b = 2 d = 1
a > b^d
$O(n^{1.6})$
$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log(n))$
MergeSort
a = 2 b = 2 d = 1
a = b^d
$O(n \log(n))$


Algorithms (& Data Structures)
92
utad
ALGORITHMS ANALYSIS & COMPLEXITY | METHODOLOGIES
Pedro Melo-Pinto 2022
BACK TO
example 3
THE SORTING PROBLEM
merging sort
does it work?
is it fast?
02 The Substitution Method
Recurrence analysis
Step 1: Generate a guess at the correct answer
Step 2: Try to prove that your guess is correct

BACK TO example 3 THE SORTING PROBLEM merging sort

does it work? is it fast?

02 The Substitution Method

Recurrence analysis

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log(n))$$

Step 1: Generate a guess at the correct answer

$$T(n) = 2^t T\left(\frac{n}{2^t}\right) + tn$$

For $t = \log(n) \rightarrow T(n) = nT(1) + \log(n)\, n = n(\log(n) + 1)$

BACK TO example 3 THE SORTING PROBLEM merging sort

does it work? is it fast?

02 The Substitution Method

Recurrence analysis

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log(n))$$

Step 2: Try to prove that your guess is correct

Inductive Hypothesis: $T(n) = n(\log(n) + 1)$

Base Case (n=1): $T(1) = 1$

Inductive Step: Assume Inductive Hypothesis for $1 \leq n < k$:

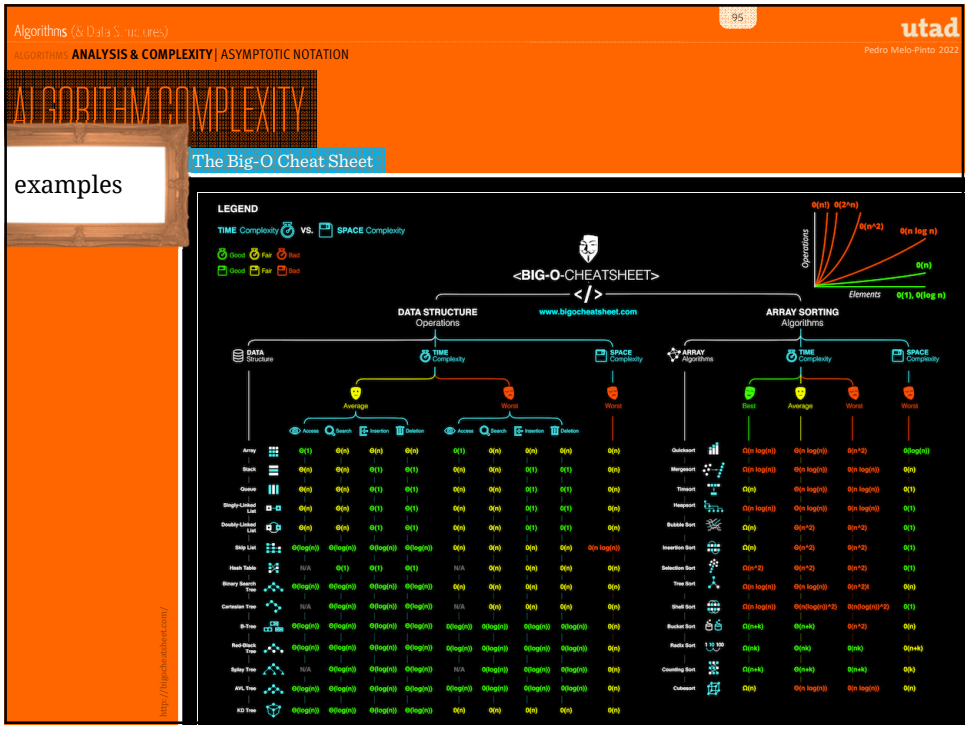
Suppose that $T(n) = n(\log(n) + 1)$ for all $1 \leq n < k$.

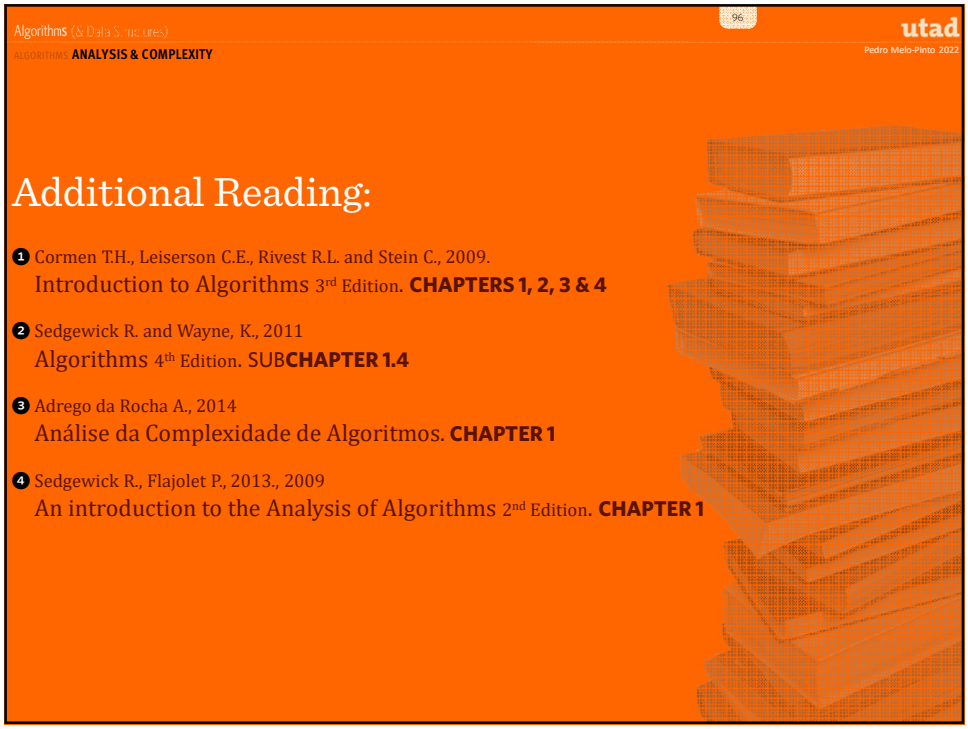
Prove Inductive Hypothesis for $n = k$:

$$T(k) = 2 \cdot T\left(\frac{k}{2}\right) + k = 2 \cdot \left(\frac{k}{2}\left(\log\left(\frac{k}{2}\right) + 1\right)\right) + k = k(\log(k) + 1)$$

So Inductive Hypothesis holds for $n = k$.

Conclusion: For all $n \geq 1$, $T(n) = n(\log(n) + 1) = O(n\log(n))$


Algorithms (& Data Structures)
95
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY | ASYMPTOTIC NOTATION
ALGORITHM COMPLEXITY
examples
The Big-O Cheat Sheet
LEGEND
TIME Complexity VS. SPACE Complexity
<BIG-O-CHEATSHEET>
DATA STRUCTURE Operations
www.bigocheatsheet.com
ARRAY SORTING Algorithms
http://bigocheatsheet.com/


Algorithms (& Data Structures)
96
utad
Pedro Melo-Pinto 2022
ALGORITHMS ANALYSIS & COMPLEXITY
Additional Reading:
1 Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009.
Introduction to Algorithms 3rd Edition. CHAPTERS 1, 2, 3 & 4
2 Sedgewick R. and Wayne, K., 2011
Algorithms 4th Edition. SUBCHAPTER 1.4
3 Adrego da Rocha A., 2014
Análise da Complexidade de Algoritmos. CHAPTER 1
4 Sedgewick R., Flajolet P., 2013., 2009
An introduction to the Analysis of Algorithms 2nd Edition. CHAPTER 1