SCC0602 - Algoritmos e Estruturas de Dados I

Algorithms

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Rafael Martins D'Addio
Monitor:

Today

- History of algorithms
- Importance of algorithms
- Main goal
- Sorting
- Conclusion

What is this course about?

- Solving problems
  - Get me from home to work (and vice-versa)
  - Balance my check book
  - Know where is the party
  - Graduate from USP
- Using a computer to help solve problems
  - Design programs (architecture, algorithms)
  - Write programs
  - Verify programs
  - Document programs

This course is **not** about

- Programming languages
- Computer architecture
- Software architecture
- Software design and implementation principles
  - Issues concerning small and large scale programming
- We will only touch upon the theory of complexity and computability

History

- Name: Persian mathematician Mohammed al-Khowarizmi, in Latin became Algorismus
- First algorithm: Euclidean Algorithm, greatest common divisor, 400-300 B.C.
- 19th century – Charles Babbage, Ada Lovelace
- 20th century – Alan Turing, John von Neumann

Al-Khwarizmi

- Persian mathematician, lived around 800AD
- Wrote a book about how to multiply with Arabic numerals
  - His ideas came to Europe in the 12th century
- Originally, "Algorisme" (old French) referred to just the Arabic number system
  - Eventually it came to mean "Algorithm" as know today

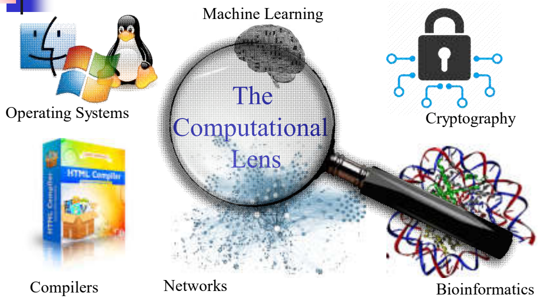
Video

Video

Importance of algorithms

- Algorithms were invented by nature
  - DNA
- Algorithms are fundamental to Computing
- Algorithms are useful
- Algorithms can be fun!

Algorithms are fundamental



Machine Learning

Operating Systems

The Computational Lens

Cryptography

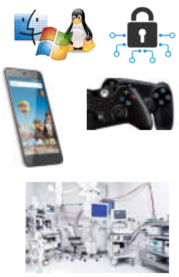
Compilers

Networks

Bioinformatics

Algorithms are useful

- Imagine yourself without them
- As we get more data and problem sizes get bigger, algorithms become more important
- Will help you get a good job



Algorithms are fun

- Algorithm design is both an art and a science
- Many surprises!
- A young area, lots of exciting research questions and opportunities!
- Will help you get a job you like!

Importance of algorithms

- Consider sorting a file of social insurance numbers for all population of São Paulo state
  - Population (n) = 44,000,000 ($n^2 \sim 10^{15}$)
  - An algorithm running in O(n^2) in a computer able to do a billion operations per second will take 10^6 seconds
    - About 11 days
  - An algorithm running in O($nlogn$) time will take only about a second on the same file
- Algorithms matter!

Video

How algorithms shape our world - Kevin Slavin

Video

Data Structures and Algorithms

- Algorithm
  - Outline, the essence of a computational procedure, step-by-step instructions
- Program
  - An implementation of an algorithm in some programming language
- Data structure
  - Organization of data needed by the program

Main goals

Adaptability

Efficiency

Reusability

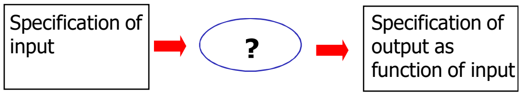
Correctness

Robustness

Quiz

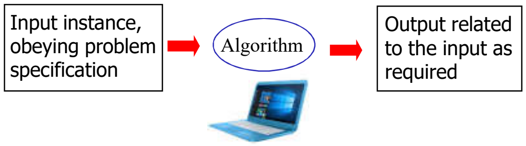
- Mention some measures of efficiency

Algorithmic problem



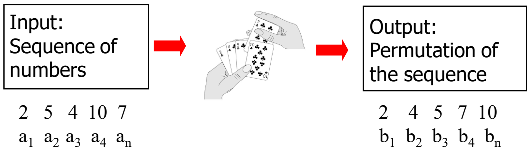
- Infinite number of input *instances* satisfying a specification
  - Example:
    - A sorted, non-decreasing sequence of natural numbers
      - The sequence is of non-zero, finite length:
        - 1, 20, 908, 909, 100000, 1000000000 (sequence of 6 numbers)
        - 3. (sequence of 1 number)

Algorithmic problem


Input instance, obeying problem specification
Algorithm
Output related to the input as required

- Algorithm describes actions on the input instances
- There are infinitely many correct algorithms for the same algorithmic problem

Example: sorting


Input: Sequence of numbers
Output: Permutation of the sequence
2 5 4 10 7
a1 a2 a3 a4 an
2 4 5 7 10
b1 b2 b3 b4 bn

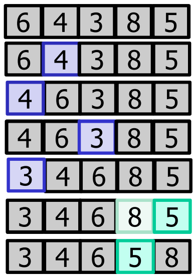
- Correctness
  - For any given input, the algorithm halts with the output:
    - $b_1 < b_2 < b_3 < .... < b_n$
    - $b_1, b_2, b_3, ..., b_n$ is a permutation of $a_1, a_2, a_3,....,a_n$
- Running time
  - Depends on
    - Number of elements (n)
    - How (partially) sorted they are
    - Algorithm used

Insertion Sort

- Initial partially sorted vector has first vector item
- Insert one item at a time
  - In the correct position of a partially sorted vector
- Example
  - Suppose all elements are different
  - How to sort, using insertion sort, the vector below?

6	4	3	8	5

Example: Insertion Sort



Start with the second element (the first element is sorted within itself…)

Pull “4” back until it is in the right place

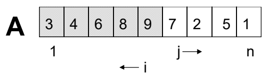
Now look at “3”

Pull “3” back until it is in the right place

“8” is good…look at 5

Fix “5” and the sequence sorted

Insertion Sort



Strategy

- Start with one card in your hand
- Insert a card in the correct position of the already sorted hand
- Continue until all cards are inserted/sorted

```
for j=2 to length(A)
   do key=A[j]
   “insert A[j] into the
   sorted sequence A[1..j-1]”
     i=j-1
     while i>0 and A[i]>key
        do A[i+1]=A[i]
           i--
     A[i+1]:=key
```

Analysis of algorithms

- Efficiency:
  - Running time
  - Space used
- Efficiency as a function of input size:
  - Number of data elements (numbers, points)
  - Number of bits in an input number
  - Number of vertices and edges (graphs)

The RAM model

- Very important to choose the level of detail
- The RAM model:
  - Instructions (each taking constant time):
    - Arithmetic (add, subtract, multiply, etc.)
    - Data movement (load, storage copy)
    - Control (conditional/unconditional branch, subroutine call, return)
  - Data types – integers and floats

Analysis of Insertion Sort

- Time to compute the **running time** as a function of the **input size**

n: length(A)
t_i: #times the while loop is tested in in line 5 for the value of j

```
for j=2 to length(A)
  do key=A[j]
  "insert A[j] into the
  sorted sequence A[1..j-1]"
    i=j-1
    while i>0 and A[i]>key
       do A[i+1]=A[i]
          i--
    A[i+1]:=key
```

cost	times
c_1	n
c_2	n-1
0	n-1
c_3	n-1
c_4	$\sum_{j=2}^{n} t_j$
c_5	$\sum_{j=2}^{n} (t_j - 1)$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	n-1

Analysis of Insertion Sort

$$\begin{aligned} T(n) &= c_1\, n + c_2\, (n-1) + c_3\, (n-1) \\ &\quad + c_4\, (n(n+1)/2 - 1) + \\ &= c_5\, [n(n-1)/2] + c_6\, [n(n-1)/2] \\ &\quad + c_7\, (n-1) \\ &= a * n^2 + b * n + c \end{aligned}$$

(***quadratic function*** of **n**)

Why c_1 occurs n times?

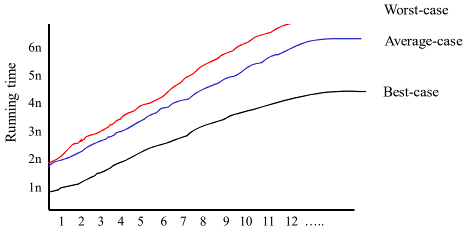
cost	times
c_1	n
c_2	n-1
0	n-1
c_3	n-1
c_4	$\sum_{j=2}^{n} t_j$
c_5	$\sum_{j=2}^{n} (t_j - 1)$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	n-1

Best/Worst/Average Case

- **Best case**:
  - Elements already sorted → *t_j=1,* running time = *f(n),* i.e., *linear* time
- **Worst case**:
  - Elements are sorted in inverse order → *t_j=j,* running time = *f(n^2),* i.e., *quadratic* time
- **Average case**:
  - *t_j=j/2,* running time = *f(n^2),* i.e., *quadratic* time

Best/Worst/Average Case (3)

- For inputs of all sizes:



Best/Worst/Average Case (4)

- **Worst case** is usually used:
  - It is an upper-bound
    - In some applications knowing the **worst-case** time complexity is of crucial importance
      - E.g., air traffic control, surgery
  - For some algorithms **worst case** occurs fairly often
  - The **average case** is often as bad as the **worst case**
  - Finding the **average case** can be very difficult

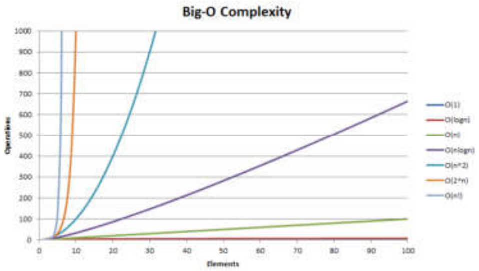
Complexities

O(1) – **constant** time, the time is independent of *n*, e.g. array look-up
O(log *n*) – **logarithmic** time, usually the log is base 2, e.g. binary search
O(*n*) – **linear** time, e.g. linear search
O(*n**log *n*) – e.g. efficient sorting algorithms
O(n^2) – **quadratic** time, e.g. selection sort
O(n^k) – **polynomial** (where *k* is a constant)
O(2^n) – **exponential** time, very slow!

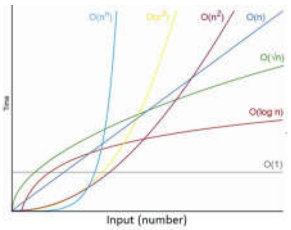
Order of growth of some common functions

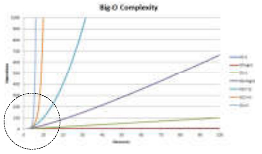
$O(1) < O(\log n) < O(n) < O(n * \log n) < O(n^2) < O(n^3) < O(2^n)$

Growth Functions

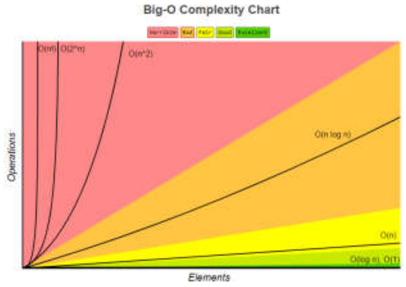


Growth Functions





Growth Functions

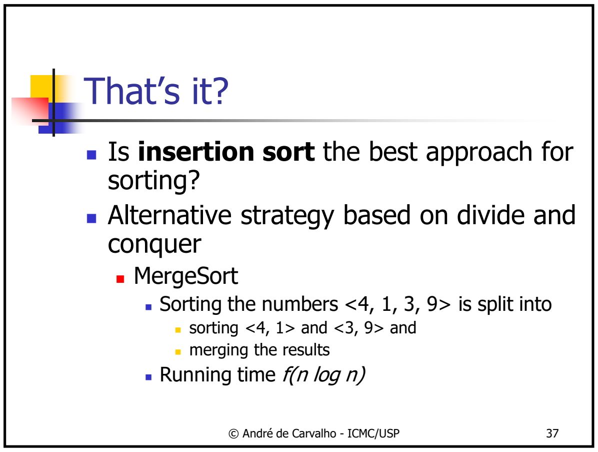


Growth rates

Growth Rates Compared

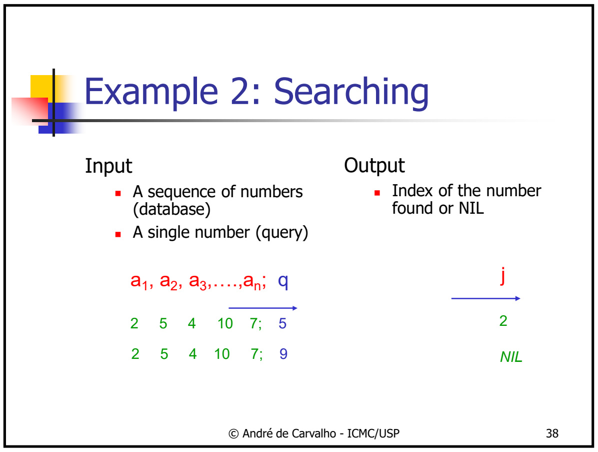
	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40320	20.9T	Don't ask!

8



That's it?

- Is **insertion sort** the best approach for sorting?
- Alternative strategy based on divide and conquer
  - MergeSort
    - Sorting the numbers <4, 1, 3, 9> is split into
      - sorting <4, 1> and <3, 9> and
      - merging the results
    - Running time *f(n log n)*



Example 2: Searching

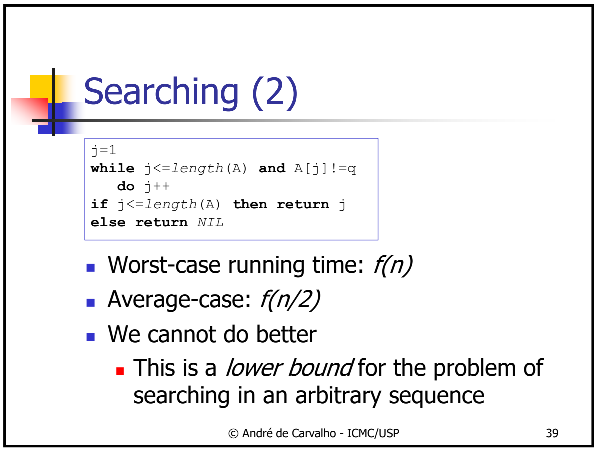
Input
- A sequence of numbers (database)
- A single number (query)

Output
- Index of the number found or NIL

$a_1, a_2, a_3, \ldots, a_n;\ q \longrightarrow j$

2 5 4 10 7; 5 → 2

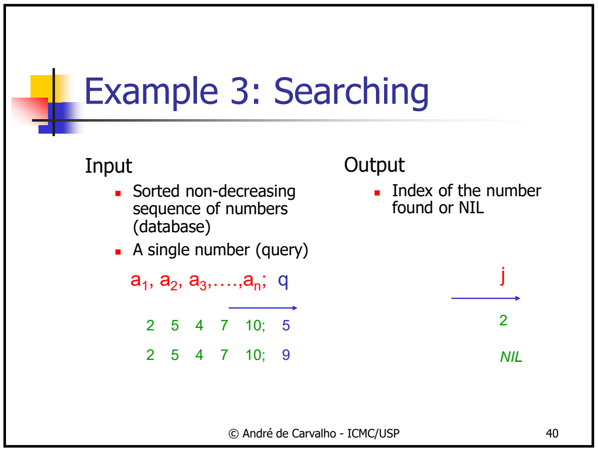
2 5 4 10 7; 9 → *NIL*



Searching (2)

```
j=1
while j<=length(A) and A[j]!=q
   do j++
if j<=length(A) then return j
else return NIL
```

- Worst-case running time: *f(n)*
- Average-case: *f(n/2)*
- We cannot do better
  - This is a *lower bound* for the problem of searching in an arbitrary sequence



Example 3: Searching

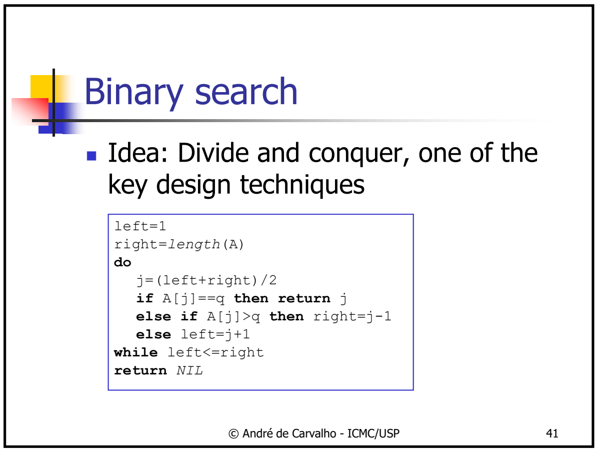
Input
- Sorted non-decreasing sequence of numbers (database)
- A single number (query)

Output
- Index of the number found or NIL

$a_1, a_2, a_3, \ldots, a_n;\ q \longrightarrow j$

2 5 4 7 10; 5 → 2

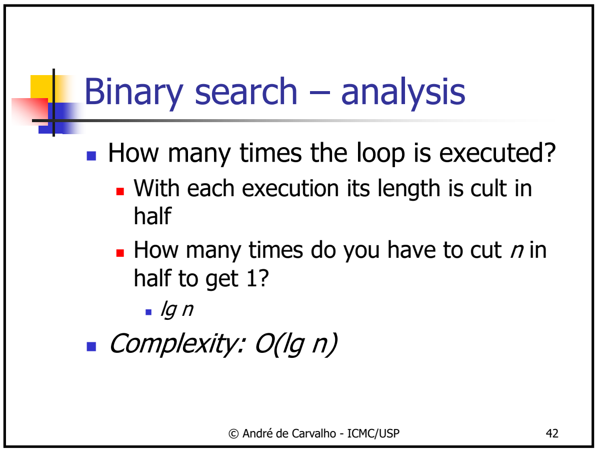
2 5 4 7 10; 9 → *NIL*



Binary search

- Idea: Divide and conquer, one of the key design techniques

```
left=1
right=length(A)
do
   j=(left+right)/2
   if A[j]==q then return j
   else if A[j]>q then right=j-1
   else left=j+1
while left<=right
return NIL
```



Binary search – analysis

- How many times the loop is executed?
  - With each execution its length is cult in half
  - How many times do you have to cut *n* in half to get 1?
    - *lg n*
- *Complexity: O(lg n)*

Animations

http://cs.armstrong.edu/liang/animation/web/InsertionSort.html

http://www.algomation.com/algorithm/insertion-sort-animated

Conclusion

- Algorithms
- Sorting
- Insertion sort
- Merge sort
- Binary search

Next Week

- Correctness of algorithms
- Asymptotic analysis, big *O* notation

Acknowledgement

- A large part of this material were adapted from
  - Simonas Šaltenis, Algorithms and Data Structures, Aalborg University, Denmark
  - Mary Wootters, Design and Analysis of Algorithms, Stanford University, USA

Questions