

Algorithms and Complexity



Algorithm

An algorithm is a calculation procedure (composed by a finite number of steps) which solves a certain problem, working on a set of *input values* and producing a set of *output values*



Example

The sorting problems is defined as follows:

- **Input set:** sequence of numbers $\langle a_1, a_2, \dots, a_n \rangle$
- **Output set:** permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ of input sequence so that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

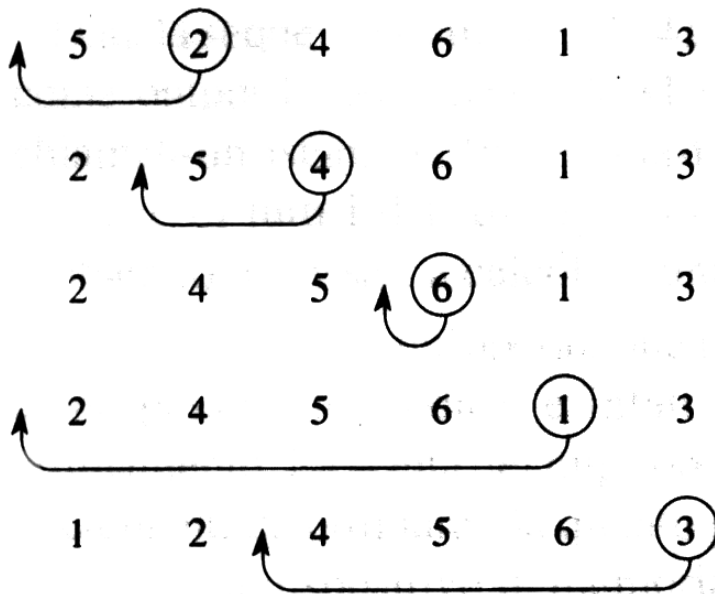
3



Insertion Sort

```
INSERTION-SORT(A)
1  for  $j \leftarrow 2$  to  $\text{length}[A]$ 
2      do  $\text{key} \leftarrow A[j]$ 
3          ▷ Insert  $A[j]$  in the sorted sequence  $A[1.. j-1]$ 
4           $i \leftarrow j - 1$ 
5          while  $i > 0$  e  $A[i] > \text{key}$ 
6              do  $A[i + 1] \leftarrow A[i]$ 
7                   $i \leftarrow i - 1$ 
8           $A[i + 1] \leftarrow \text{key}$ 
```

4



5

Analysis of algorithms

- Analysing algorithms means to predict the amount of resources (I/O, memory, time) required by an algorithm during its execution.
- Such analysis should be independent from the kind of hardware platform on which the algorithm is executed.
- We will assume in the rest of the course that the hardware platform is a common machine with a single CPU (*Random Access Machine* or RAM).

6



Problem Dimension

Analysis of algorithms is usually performed with respect to one or more parameters which characterize the dimensions of the problem.

Example

- In the [sorting algorithm](#) such parameter is the number of elements of the input sequence.
- In the [multiplication](#) between integer numbers, dimension is given by the number of bits used to represent the operands.

7



Hypothesis

- Each statement in the pseudo-code requires a fixed time
- Each statement has a different execution time.

8

Analysis of insertion sort

t_j is the number of times the statement is repeated, for a certain value of j

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|--|--------------------------|-------------|--------------|-------|--|-----|-------|--|---------|-----|--|---------|-------|--|---------|-------|--|--------------------|-------|--|--------------------------|-------|--|--------------------------|-------|--|---------|
| <pre> INSERTION-SORT(A) 1 for j ← 2 to length[A] 2 do key ← A[j] 3 ▷ Insert A[j] in the ordered ▷ sequence A[1 .. j-1] 4 i ← j - 1 5 while i > 0 e A[i] > key 6 do A[i + 1] ← A[i] 7 i ← i - 1 8 A[i + 1] ← key </pre> | <table border="0"> <tr> <td><i>cost</i></td> <td>n°</td> <td><i>times</i></td> </tr> <tr> <td>c_1</td> <td></td> <td>n</td> </tr> <tr> <td>c_2</td> <td></td> <td>$n - 1$</td> </tr> <tr> <td>0</td> <td></td> <td>$n - 1$</td> </tr> <tr> <td>c_4</td> <td></td> <td>$n - 1$</td> </tr> <tr> <td>c_5</td> <td></td> <td>$\sum_{j=2}^n t_j$</td> </tr> <tr> <td>c_6</td> <td></td> <td>$\sum_{j=2}^n (t_j - 1)$</td> </tr> <tr> <td>c_7</td> <td></td> <td>$\sum_{j=2}^n (t_j - 1)$</td> </tr> <tr> <td>c_8</td> <td></td> <td>$n - 1$</td> </tr> </table> | <i>cost</i> | n° | <i>times</i> | c_1 | | n | c_2 | | $n - 1$ | 0 | | $n - 1$ | c_4 | | $n - 1$ | c_5 | | $\sum_{j=2}^n t_j$ | c_6 | | $\sum_{j=2}^n (t_j - 1)$ | c_7 | | $\sum_{j=2}^n (t_j - 1)$ | c_8 | | $n - 1$ |
| <i>cost</i> | n° | <i>times</i> | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_1 | | n | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_2 | | $n - 1$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | | $n - 1$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_4 | | $n - 1$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_5 | | $\sum_{j=2}^n t_j$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_6 | | $\sum_{j=2}^n (t_j - 1)$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_7 | | $\sum_{j=2}^n (t_j - 1)$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c_8 | | $n - 1$ | | | | | | | | | | | | | | | | | | | | | | | | | | |

9

Execution time of insertion sort

$$\begin{aligned}
 T(n) = & c_1 n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\
 & + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1) .
 \end{aligned}$$

10



Best Case

If the array is already sorted, the $t_j = 1$ for every j .

Then:

$$\begin{aligned} T(n) &= c_1 n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

Which can be written as follows

$$T(n) = an + b$$

11



Worst Case

If an array is sorted with inverse order at step j , j comparisons and swaps: $t_j = j$.

Remember that:

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

12



Worst Case (2)

Therefore

$$\begin{aligned}
 T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) + \\
 &\quad + c_6 \left(\frac{n(n-1)}{2} \right) + c_7 \left(\frac{n(n-1)}{2} \right) + c_8(n-1) \\
 &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n + \\
 &\quad - (c_2 + c_4 + c_5 + c_8) .
 \end{aligned}$$

In the worst case:

$$T(n) = an^2 + bn + c$$

13



Average Case

Worst case analysis is important as it defines an upper bound to resources required by an algorithm.

In some cases the **average case** (or *mean value*) is worthy of being analyzed.

14



Importance of complexity analysis

Analysis of resources of an algorithm (also known as complexity analysis) allows describing performance of the algorithm depending on **problem dimension**.

Choosing an algorithm with lower complexity is the best choice independently from the used technology and it can make a difference for problems with big dimensions.

15



Example

Suppose you can get the solution of a problem with 2 algorithms:

- One with complexity $T(n) = 2n^2$
- The other with complexity $T(n) = 50n \log_2 n$

Suppose you have these two machines:

- [Intel Core i7 Extreme 965EE](#) performing 76000 MIPS at 3.2 GHz for the first algorithm
- [AMD Athlon FX-60](#) performing 18000 MIPS at 2.6 GHz for the second algorithm.
- **MIPS** = Millions of Instructions Per Second

16



Example (II)

Execution time is proportional to $T(n) / \text{MIPS}$:

- When $n = 1 \text{ K} (10^3)$
 - Intel – 76000 MIPS: 0.026 ms
 - AMD – 18000 MIPS: 0.027 ms
- When $n = 1 \text{ M} (10^6)$
 - Intel – 76000 MIPS : 26310 ms
 - AMD – 18000 MIPS : 55,36 ms
- When $n = 1 \text{ G} (10^9)$
 - Intel – 76000 MIPS : 26,3 Ms = 7305 hours ! = 304 days !!!
 - AMD – 18000 MIPS : 83,04 s

17



Asymptotic Notation

In complexity analysis the **asymptotic notation** is often used to discover the complexity of an algorithm when problem dimension increases.

The asymptotic notation is based on 3 notations:

- Theta Notation Θ
- "Big O" Notation O
- Omega Notation Ω .

18



Θ Notation

Given an algorithm of complexity $T(n)$.

$T(n) = \Theta(g(n))$ if and only if there exist three positive constants c_1, c_2 e n_0 so that

$$0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$$

for each $n \geq n_0$.

In this case $g(n)$ is the **asymptotic limit** for $T(n)$.

19



“Big O” Notation

$T(n) = O(g(n))$ if and only if there exist two positive constants c and n_0 so that:

$$0 \leq T(n) \leq c \cdot g(n)$$

for each $n \geq n_0$.

In this case $g(n)$ is a **upper asymptotic limit** for $T(n)$.

20

Ω Notation

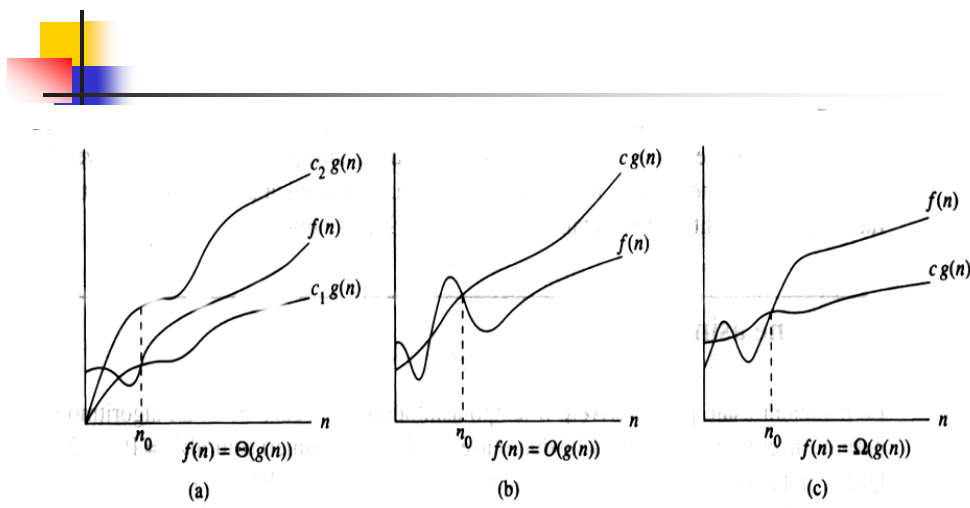
$T(n) = \Omega(g(n))$ if and only if there exist two positive constants c and n_0 so that

$$0 \leq c \cdot g(n) \leq T(n)$$

for each $n \geq n_0$.

In this case $g(n)$ is a **lower asymptotic limit** for $T(n)$.

21



22



Theorem

Given two functions $g(n)$ and $T(n)$,

$T(n) = \Theta(g(n))$ if and only if

- $T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$