Algoritma & Pemrograman Saintifik

Gatot F. Hertono, Ph.D

Departemen Matematika

SCMA601401

Let's consider these algorithms

Problem: given a sequence of *n* elements already sorted in ascending order

1,5,10, 23,45, 100

Design an algorithm to check whether a value *x* occurs in the sequence

Simple idea: Check one by one by comparing *x* to each elements in the sequence from the beginning to the end

1,5,10, 23,45, 100

x

- How many comparisons do we need?
- Is there any other algorithm faster than this idea?

Running Time

Running time could be defined as:

- Number of basic/primitive operations or 'steps' executed;
- Constant amount of time for each line of pseudocode.

Basic/Primitive operations include:

- Arithmetic operations – such as '+', '-', '*' , '/' etc.
- Logical comparisons – such as '>' , '<' , '=' , '≠' , '≤' , '≥'.
- Function calls.

Assumption:

- Operations are executed sequentially
- All operations cost 1 unit

- The running time depends on the input. Example: an already sorted sequence is easier to sort.
- The running time of an algorithm is determined by its input size *n*

$$T_A(n) = time\ of\ algorithm\ A\ on\ length\ n\ inputs$$

Examples of Basic/Primitive Operations

Algorithm	***Input Types***	***Basic Operations***
List Searching	*List* with *n* elements	Comparation
List Sorting	*List* with *n* elements	Comparation
Matrix Product	*n* x *n matrices*	Scalar Products
Prime Factorisation	*n* digit numbers	Scalar Division
Polynomial Evaluation	*n* degree polynomial	Scalar Products
Tree Traversal	*Tree* with *n* nodes	*Visiting* a node

Basic operation: the operation that contributes the most towards the running time of the algorithm

Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*

Notes:

- Eliminates dependence on the speed of our computer, otherwise impossible to verify and to compare

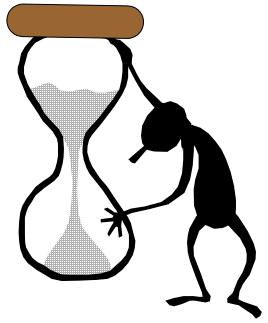
Time Complexity

The complexity of an algorithm is determined by the number of basic operations and how many time the algorithm computes those basic operations.

Notes: The complexity analysis is machine independent.

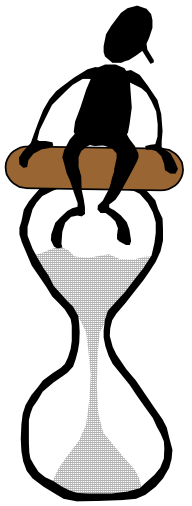
Time complexity of an algorithm will determine the running time depends on its input size, i.e. the time complexity is a function of input size.

Time Complexity maps
"input size"
to
"time" T(n) executed.



Purpose

- To estimate how long a program will run.
- To estimate the largest input that can reasonably be given to the program.
- To compare the efficiency of different algorithms.
- To help focus on the parts of code that are executed the largest number of times.
- To choose an algorithm for an application.



Time Complexity: an example

Contoh:

Algoritma	*Time*	*Jumlah*
Power1(real x, positive integer n)		
1 hasil ← x	t_1	1
2 **for** I ← 1 **to** n - 1 **do**	t_2	n-1
3 hasil ← hasil * x	t_3	n-1
4 **return** hasil	t_4	1

dengan demikian dapat dikatakan bahwa *running time* algoritma POWER1( ) adalah:

$$T(n) = t_1 + t_2(n - 1) + t_3(n - 1) + t_4$$

$$= (t_2 + t_3).(n - 1) + (t_1 + t_4)$$

Best, Worst and Average Case

Sometimes, given two different inputs with a same size, an algorithm can have different running time.

Example:
Suppose a sorting algorithm has some inputs with a same size but different order:

In ascending order

-Input 1: 10, 5, 23, 45, 1, 100 → **Average case**

-Input 2: 1,5,10, 23,45, 100 → **Best case**

-Input 3: 100, 45, 23, 10, 5, 1 → **Worst case**

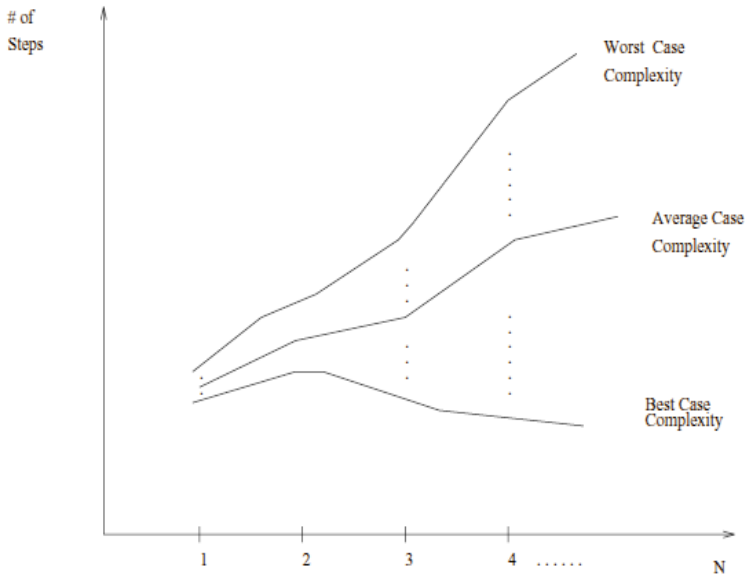
Do those inputs give the same running time?

Best, Worst and Average Case (cont.)

The *worst case complexity* of the algorithm is the function defined by the maximum number of steps taken on any instance of size n.

The *best case complexity* of the algorithm is the function defined by the minimum number of steps taken on any instance of size n.

The *average-case complexity* of the algorithm is the function defined by an average number of steps taken on any instance of size n.



Remark: All cases are relative to the algorithm under consideration.

Best, Worst and Average Cases (cont.)

Let I_n denote a set of all input with size n of an algorithm and $\tau(i)$ denote the number of primitive operations of the corresponding algorithm when given input *i*.

Mathematically, we can define:

Best-case Complexity: is a function $B(n) \ni B(n) = \min\{ \tau(i) \mid i \in I_n \}$

Worst-case Complexity: is a function $W(n) \ni W(n) = \max\{ \tau(i) \mid i \in I_n \}$

Average-case Complexity: is a function $A(n) \ni A(n) = \sum_{i \in I_n} \tau(i).p(i)$

where $p(i)$ is the probability of i occurs as an input of an algorithm.