Chapter : 1

Introduction

Introduction to Algorithms

- Algorithm: It is the finite sequence of operations/instructions which transform the given input to correct output.

- Algorithmics: it is the branch that performs the study of algorithms

Properties of algorithms

- **Input** from a specified set,
- **Output** from a specified set (solution),
- **Definiteness** of every step in the computation,
- **Correctness** of output for every possible input,
- **Finiteness** of the number of calculation steps,
- **Effectiveness** of each calculation step and
- **Generality** for a class of problems.

Problems & Instance

- Problem: Multiply two positive integers
- Instance:
  - (10,2) is proper instance for above problem
  - (-5,2) is not proper instance
  - (10,2.5) is again not proper instance

- Algorithm must work correctly on every instance it claims to solve
- How to show that it works incorrect?
  - Find any one instance for which it doesn’t work correctly

Problems & Instance (contd..)

- **Domain of definition (The set of instances):**

To prove the correctness of the algorithm, one needs to limit the size of instance.

Any real computing device has a limit on the size of instances it can handle, either because the numbers involved get too big or because we run out of storage.

Size of instance

- If we are searching an array, the "size" of the input could be the size of the array
- If we are merging two arrays, the "size" could be the sum of the two array sizes
- If we are computing the n^{th} Fibonacci number, or the n^{th} factorial, the "size" is n
- We choose the "size" to be the parameter that most influences the actual time/space required
  - It is *usually* obvious what this parameter is
  - Sometimes we need two or more parameters

Efficiency of Algorithms

- Which algorithm needs to be chosen when more than one algorithm is available?

Three Approaches:

- Empirical (Posteriori)

  programming all the techniques and trying them of different instances.

- Theoretical (Priori):

  determining mathematically the quantity of resources needed as a function of the size of instance.

  Resources: computing time, storage space.

- Hybrid approach:

  Algo's efficiency is determined theoretically and required numerical parameters are determined empirically.

Limitations of Empirical Approach

- The algorithm has to be implemented, which may take a long time and could be very difficult.
- Results may not be indicative for the running time on other inputs that are not included in the experiments.
- In order to compare two algorithms, the same hardware and software must be used.

Theoretical Approach

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, *n*.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

Principle of Invariance

- What is the unit for storage space measurement?
- What is the unit for time measurement?

- Principle of Invariance: Two different implementations of the same algorithms will not differ in efficiency by more than some multiplicative constant.

- Example: $t_1(n)$ and $t_2(n)$ are the time for any algorithm for different implementations, then there exist "c" & "d" such that ..

$$t_1(n) <= c * t_2(n)$$
$$t_2(n) <= d * t_1(n)$$

Means, the running time of either implementation is bounded by a constant multiple of the running time of the other.

Principle of Invariance (contd..)

- Principle suggest that there is no such unit exist.

We only express the time taken by an algorithm within a multiplicative constant.

In the order of t(n)

Frequently occurring orders:

-Linear
-Quadratic
-Cubic
-Polynomial
-Exponential, etc.

Hidden constants:

n^2 days and n^3 seconds

Common time complexities

BETTER

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

WORSE

The Growth Rate of the Six Popular functions

n	$\log n$	n	$n\log n$	n^2	n^3	2^n
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,094	262,144	$1.84 * 10^{19}$
128	7	128	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	256	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	512	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	1,024	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$

Average, Best and Worst Case

- Usually we would like to find the *average* time to perform an algorithm
- However, Sometimes the “average” isn’t well defined
    - Example: Sorting an “average” array
        - Time typically depends on how out of order the array is
- Sometimes finding the average is too difficult

- Often we have to be satisfied with finding the *worst* (longest) time required
    - Sometimes this is even what we want (say, for time-critical operations)
- The *best* (fastest) case is seldom of interest

Why to look for efficiency?

What to choose: Better hardware or better algorithm?

Case 1: Algo1 on machine1 (takes $10^{-4} * 2^n$ seconds)

for n=10, t=1/10 sec
for n=20, t≈2 minutes
for n=30, t ≥ 1 day
for n=38, t ≥ 1 year

Case 2: Algo1 on machine2 (takes $10^{-6} * 2^n$ seconds, 100x faster)

for n=10, t=1/1000 sec
.. ..
for n=45, t ≥ 1 year

Why to look for efficiency?

What happens with better algorithm?

Case 3: Algo2 on machine1 (takes $10^{-2} * n^3$ seconds)

for n=10, t=10 sec

for n=20, t≈1 or 2 minutes

for n=30, t≈4.5 minutes

.. ..

for n=200, t ≥ 1 day

for n=1500, t ≈ 1 year

Case 4: Algo2 on machine2

even faster than case 3..!!