Analysis and Design of Algorithms

Asymptotic Notations

Contents

- Analysis of Algorithm- PROBLEM SIZE
- Analysis Of Algorithm: COMPLEXITY
- Complexity of Algorithms
- Asymptotic Notation
- Numerical Comparison of Different Algorithms
- Classification of Algorithms

Analysis of Algorithm- PROBLEM SIZE

- *The field of computer science, which studies efficiency of algorithms, is known as analysis of algorithms.*
- Characterize an algorithm as a function of the "problem size".
- E.g.
  - Input data = array → problem size is N (length of array)
  - Input data = matrix → problem size is N x M

Analysis Of Algorithm: COMPLEXITY

Some questions to answer:

- ☑ How fast can we solve a problem?
- ☑ There may be many algorithms for a given problem. Which algorithm to use?
- ☑ What are the classical algorithm design techniques?
- ☑ Are there problems inherently difficult to solve?
- ☑ How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency
- ☑ Does there exist a better algorithm?
- ☑ Lower bounds
- ☑ Optimality

Complexity of Algorithms

- The **complexity** of an **algorithm M**

  is the **function f(n)**

  which gives the **running time and/or storage space requirement of the algorithm in terms of the size 'n' of the input data.**
- Mostly, the **storage space** required by an algorithm is simply a **multiple of the data size 'n'.**
- **Complexity** shall refer to the **running time** of the algorithm.

Complexity of Algorithms

- The function f(n), gives the **running time of an algorithm**, depends not only on the size ‘n’ of the input data **but also on the particular data.**
- The complexity function f(n) for certain cases are:

  1. Best Case: The minimum possible value of f(n) is called the best case.

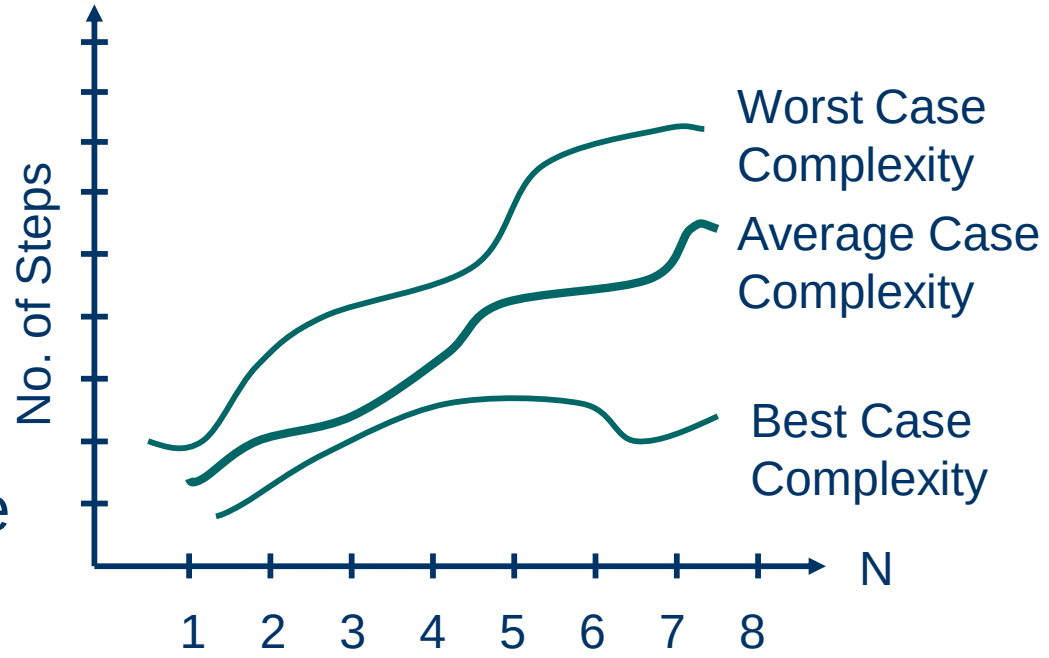
  2. Average Case : The expected value of f(n).

  3. Worst Case: The maximum value of f(n) for any key possible input.

Complexity of Algorithms

Analyzing of an algorithm is concerned of three cases:

- Worst Case Complexity
- Best Case Complexity
- Average Case Complexity



Complexity of Algorithms

- Worst case: $C_{worst}(n)$ – maximum over inputs of size n
- Best case: $C_{best}(n)$ – minimum over inputs of size n
- Average case: $C_{avg}(n)$ – "average" over inputs of size n

- ☑ Number of times the basic operation will be executed on typical input.
- ☑ NOT the average of worst and best case
- ☑ Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs. So,

avg = expected under uniform distribution.

Asymptotic Notation

- Need to abstract further
- Give an “idea” of how the algorithm performs
- n steps vs. n+5 steps
- n steps vs. n^2 steps

Asymptotic analysis - terminology

- Special classes of algorithms:

logarithmic:	**$O(\log n)$**
linear:	**$O(n)$**
quadratic:	**$O(n^2)$**
polynomial:	**$O(n^k)$, $k \geq 1$**
exponential:	**$O(a^n)$, $n > 1$**

- Polynomial vs. exponential ?
- Logarithmic vs. polynomial ?

Analyzing Algorithms

- Suppose **'M'** is an **algorithm**, and

  Suppose '**n**' is the **size of the input data**.

  Clearly the **complexity f(n)** of M **increases as n increases.** It is usually the **rate of increase of f(n) we want to examine.**
- This is usually done by **comparing f(n) with some standard functions.**
- The most common computing times are:

  $O(1)$, $O(\log_2 n)$, $O(n)$, $O(n.\log_2 n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, $n!$ and n^n.

Numerical Comparison of Different Algorithms

n	$\log n$	$n \log n$	n^2	n^3	2^n
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4294967296

Classification of Algorithms

- If '**n**' is the **number of data items** to be processed
  or **degree of polynomial**
  or the **size** of the file to be sorted or searched
  or the **number of nodes in a graph** etc.

- **n=1 means**

  Next instructions of most programs are executed once or at most only a few times. If all the instructions of a program have this property, we say that its running time is a constant.

Classification of Algorithms

- **Log n means**

  When the running time of a program is logarithmic, the program gets slightly slower as n grows. This running time commonly occurs in programs that solve a big problem by transforming it into a smaller problem, cutting the size by some constant fraction.

Classification of Algorithms

- **n means**

  When the running time of a program is linear, it is generally the case that a small amount of processing is done on each input element. This is the optimal situation for an algorithm that must process n inputs.

- **n.log n means**

  This running time arises for algorithms that solve a problem by breaking it up into smaller sub-problems, solving then independently, and then combining the solutions. When n doubles, the running time more than doubles.

Classification of Algorithms

- **n^2 means**

  When the running time of an algorithm is quadratic, it is practical for use only on relatively small problems. Quadratic running times typically arise in algorithms that process all pairs of data items (perhaps in a double nested loop) whenever n doubles, the running time increases four fold.

- **n^3 means**

  Similarly, an algorithm that process triples of data items (perhaps in a triple–nested loop) has a cubic running time and is practical for use only on small problems. Whenever n doubles, the running time increases eight fold.

Classification of Algorithms

- **2^n means**

  Few algorithms with exponential running time are likely to be appropriate for practical use, such algorithms arise naturally as “brute–force” solutions to problems. Whenever n doubles, the running time squares.