7 : Analysis of Algorithms

IT3206– Data Structures and Algorithms

Level II - Semester 3

Overview

- This section will introduce approaches to the analysis of algorithms while emphasizing the requirement of an algorithm analysis mechanism.
- The running time of an algorithm and the rate of growth of algorithms are discussed further in this section.
- Moreover, this section will introduce the concept of asymptotic analysis and explain the big-O notation in detail.

Intended Learning Outcomes

- At the end of this lesson, you will be able to;
  - Explain running time analysis and rate of growth
  - Illustrate types of algorithm analysis and asymptotic notations

List of subtopics

7.1 Introduction to analysis of algorithms

7.2 Types of analysis

7.3 Big-O notation

7.1 Introduction to analysis of algorithms

What is an Algorithm?

An algorithm is the step-by-step unambiguous instructions to solve a given problem.

- In the traditional study of algorithms, there are two main criteria for judging the merits of algorithms:
  - Correctness: does the algorithm give solution to the problem in a finite number of steps?
  - Efficiency: how much resources (in terms of memory and computation time) does it take to execute

7.1 Introduction to analysis of algorithms contd.

Analyzing an algorithm

Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.

- Occasionally, resources such as memory, communication bandwidth, or computer hardware are of primary concern, but most often it is computational time that we want to measure.

- Generally, by analyzing several candidate algorithms for a problem, we can identify a most efficient one.

- Such analysis may indicate more than one viable candidate, but we can often discard several inferior algorithms in the process.

7.1 Introduction to analysis of algorithms contd.

Analyzing an algorithm cont.

- To go from city “A” to city “B”, there can be many ways of accomplishing this: by flight, by bus, by train and also by bicycle. Depending on the availability and convenience, we choose the one that suits us.

- Similarly, in computer science, multiple algorithms are available for solving the same problem (for example, a sorting problem has many algorithms, like insertion sort, selection sort, quick sort and many more).

- Algorithm analysis helps us to determine which algorithm is most efficient in terms of time and space consumed.

- The goal of the analysis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.)

7.1 Introduction to analysis of algorithms contd.

Running Time Analysis

- It is the process of determining how processing time increases as the size of the problem (input size) increases.

- Input size is the number of elements in the input, and depending on the problem type, the input may be of different types.

- The following are the common types of inputs.
  - Size of an array
  - Polynomial degree
  - Number of elements in a matrix
  - Number of bits in the binary representation of the input
  - Vertices and edges in a graph.

7.1 Introduction to analysis of algorithms contd.

Comparing Algorithms?

1. Compare execution times?

   Not recommended: execution times are specific to a particular computer.

2. Count the number of statements executed?

   Not recommended: Number of statements vary with the programming language as well as the style of the individual programmer.

7.1 Introduction to analysis of algorithms contd.

Comparing Algorithms – Ideal Solution

- Uses a high-level description of the algorithm instead of an implementation
- Express running time as a function of the input size n
- Compare different functions corresponding to running times.
- Such an analysis is independent of hardware, software environment.

7.1 Introduction to analysis of algorithms contd.

Comparing Algorithms – Example

- Associate a “cost” with each statement.
- Find the “total cost” by finding the total number of times each statement is executed.

	Cost
`for (i=0; i>n ; i++)`	C2
`arr[i] = 0;`	C1

$$(n+1)*C2 + n*C1 = (C1+C2)n + C2$$

7.1 Introduction to analysis of algorithms contd.

Comparing Algorithms – Example

	Cost
`sum = 0;`	C1
`for (i=0; i>n ; i++)`	C2
`    for (j=0; j<n; j++)`	C2
`       sum += arr[i][j]`	C3

$$C1 + C2(n+1) + C2(n+1)n + C3*n^2$$

7.1 Introduction to analysis of algorithms

Rate of Growth

- The rate at which the running time increases as a function of input is called rate of growth.

- Let us assume that you go to a shop to buy a car and a bicycle. If your friend sees you there and asks what you are buying, then in general you say buying a car.

  This is because the cost of the car is high compared to the cost of the bicycle (approximating the cost of the bicycle to the cost of the car).

$$Total\ Cost = cost\_of\_car + cost\_of\_bicycle$$
$$Total\ Cost \approx cost\_of\_car\ (approximation)$$

7.1 Introduction to analysis of algorithms

Rate of Growth Contd.

- or the above-mentioned example, we can represent the cost of the car and the cost of the bicycle in terms of function, and for a given function ignore the low order terms that are relatively insignificant (for large value of input size, n).

- As an example, in the case below, n^4, $2n^2$, 100n and 500 are the individual costs of some function and approximate to n^4 since n^4 is the highest rate of growth.

$$n^4 + 2n^2 + 100n + 500 \approx n^4$$

7.1 Introduction to analysis of algorithms

Rate of Growth Example

Assume that running time of an algorithm can be represented in f(n).

$$f(n) = n^2 + 100n + 10^2$$

$$n = 1 \rightarrow \mathbf{1^2} + 100 + 10^2$$

$$n = 10^2 \rightarrow \mathbf{10^4} + 10^4 + 10^2$$

$$n = 10^6 \rightarrow \mathbf{10^{12}} + 10^8 + 10^2$$

$$n = 10^{12} \rightarrow \mathbf{10^{24}} + 10^{14} + 10^2$$

n^2 term dominates as the number of inputs increase

What is the term that dominates as the number of inputs increase?

7.1 Introduction to analysis of algorithms

Commonly Used Rates of Growth

Time Complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
logn	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
nlogn	Linear Logarithmic	Sorting n items by 'divide-and-conquer' - Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

7.2 Types of analysis

- For a given algorithm, we can represent the best, worst and average cases in the form of expressions. As an example, let f(n) be the function which represents the given algorithm.

$f(n)=n^2+500$ *for worst case*

$f(n)=n+100n+500$ *for best case*

Similarly for the average case. The expression defines the inputs with which the algorithm takes the average running time (or memory).

7.2 Types of analysis contd.

- **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are.

- **Best case**
  - Provides a lower bound on running time
  - Input is the one for which algorithm runs the fastest

$$Lower\ Bound \leq Running\ Time \leq Upper\ Bound$$

- **Average case**
  - Provides a prediction about the running time of the algorithm.
  - Assumes that the input is random

7.2 Types of analysis contd.

Asymptotic Analysis

- Complexity refers to the rate at which the storage or time grows as a function of the problem size.

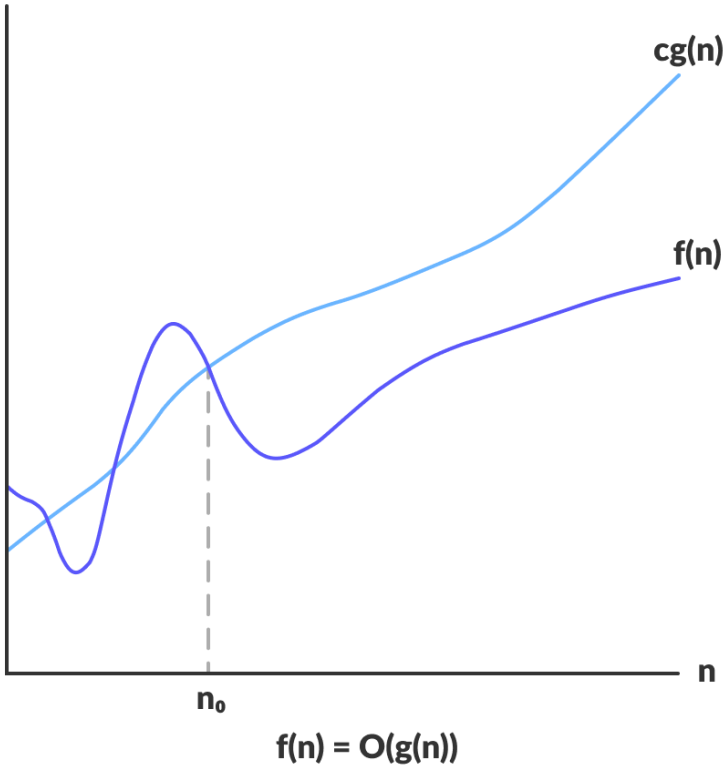
- Asymptotic Analysis is based on the idea that as the problem size grows, the complexity can be described as a simple proportionality to some known function.

Notations used.

- Big-O (O)
- Omega (Ω)
- Theta (Θ)

7.3 Big-O notation (O)

- Given functions f(n) and g(n),
  we say that f(n) = O(g(n)) if
  There exists positive constants c and n_0 such that
  $f(n) \leq cg(n)$ for all $n \geq n_0$



7.3 Big-O notation contd.

Big-O and Growth Rate

- The big-O notation gives an upper bound on the growth rate of a function.
- The statement " f(n) = O(g(n))" means that the growth rate of f(n) is no more than the growth rate of g(n).
- We can use big-O notation to rank functions according to their growth rate.

7.3 Big-O notation contd.

Properties of Big-O

- Big-O is inherently imprecise; hence the smallest possible g(n) is selected.

- If $f(n) = n^2+200n+45$
- For g(n) we can chose anything greater than n^2
  - Eg: $O(n^4)$, $O(n^6)$
- But $O(n^2)$ is used for the Big-O as it gives better representation.

7.3 Big-O notation contd.

Properties of Big-O contd.

- Big-O is transitive.

If f(n) is O(g(n))

and

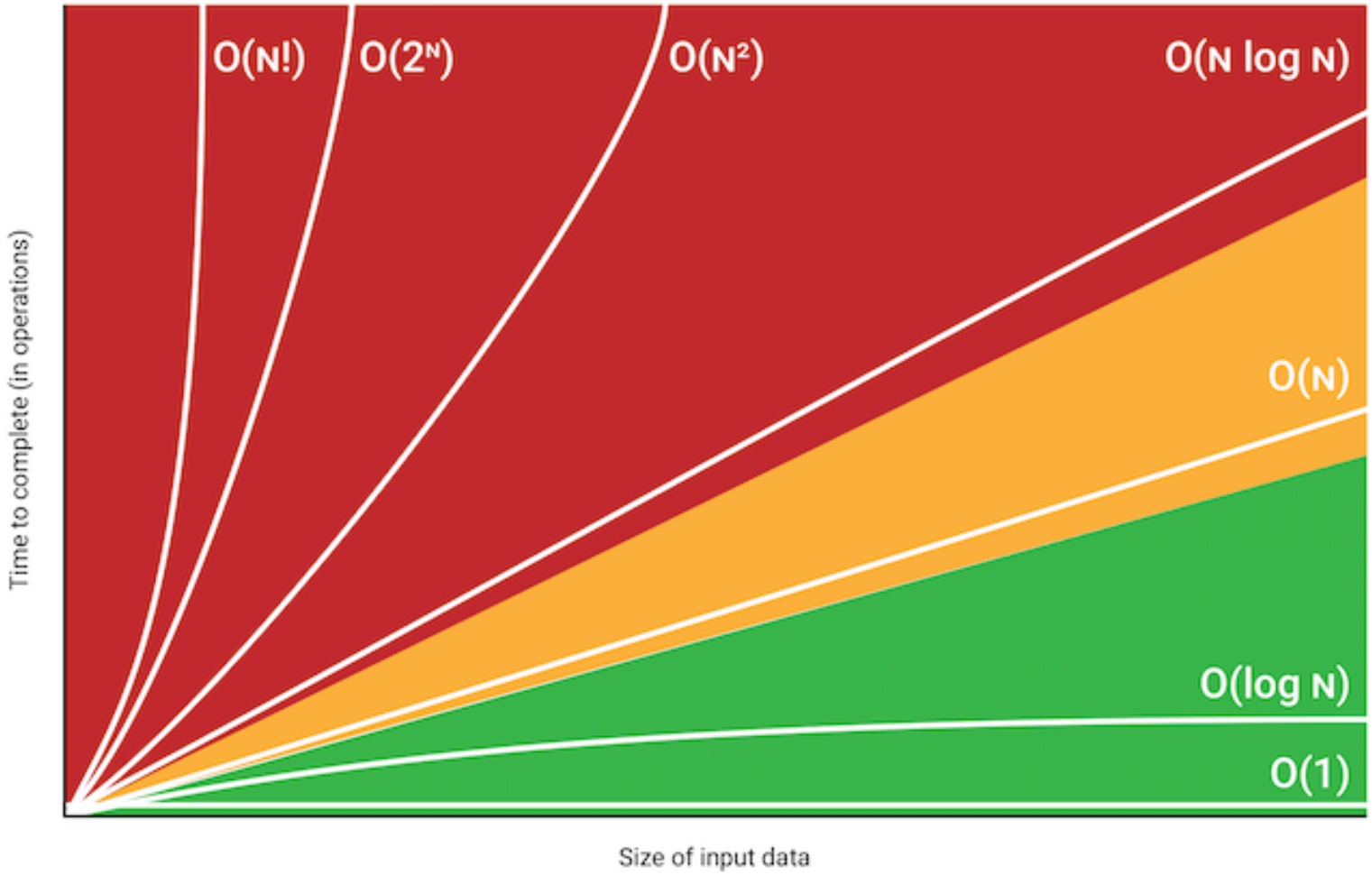
g(n) is O(h(n))

then

f(n) is O(h(n))

7.3 Big-O notation contd.

Big-O and Growth Rate Contd.



7.3 Big-O notation contd.

Big-O Summary

Notation	Type	Examples	Description
O(1)	Constant	Hash table access	Remains constant regardless of the size of the data set
O(log n)	Logarithmic	Binary search of a sorted table	Increases by a constant. If n doubles, the time to perform increases by a constant, smaller than n amount
O(< n)	Sublinear	Search using parallel processing	Performs at less than linear and more than logarithmic levels
O(n)	Linear	Finding an item in an unsorted list	Increases in proportion to n. If n doubles, the time to perform doubles
O(n log(n))	n log(n)	Quicksort, Merge Sort	Increases at a multiple of a constant
$O(n^2)$	Quadratic	Bubble sort	Increases in proportion to the product of n*n
$O(c^n)$	Exponential	Travelling salesman problem solved using dynamic programming	Increases based on the exponent n of a constant c
O(n!)	Factorial	Travelling salesman problem solved using brute force	Increases in proportion to the product of all numbers included (e.g., 1*2*3*4...)

Summary

The Running Time Analysis of an Algorithm	The process of determining how processing time increases with respect to the input time.
The Rate of Growth of an Algorithm	The rate at which the running time increases as a function of input size.
Big-O Notation	Big-O notation is a mathematical notation that gives the upper bound on the rate of growth of an algorithm.