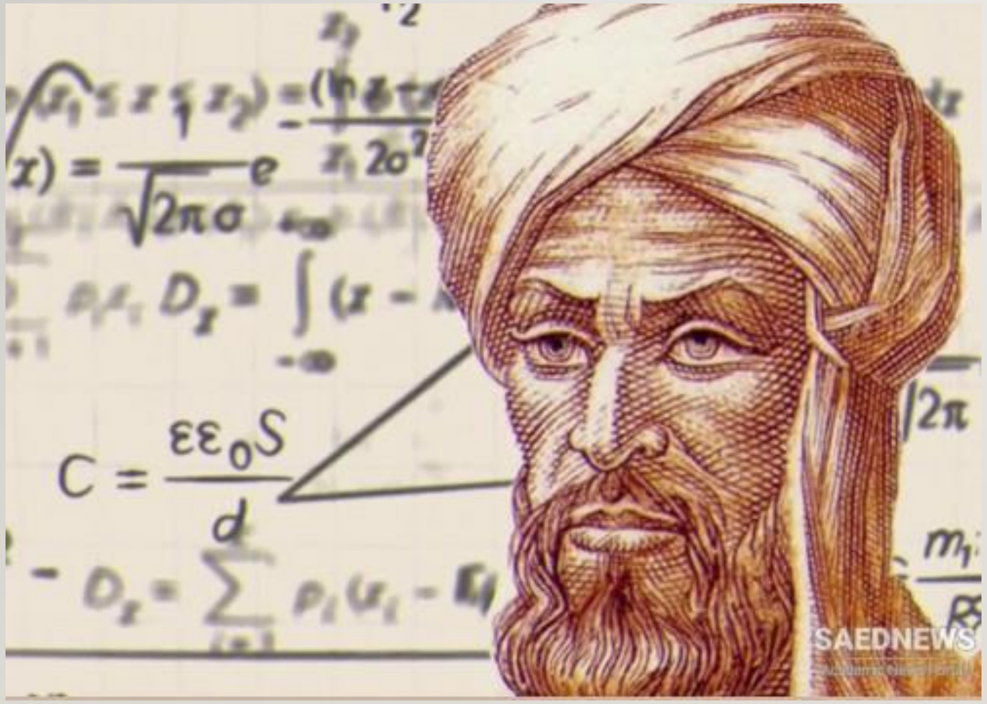
ALGORITHM

DATA STRUCTURES

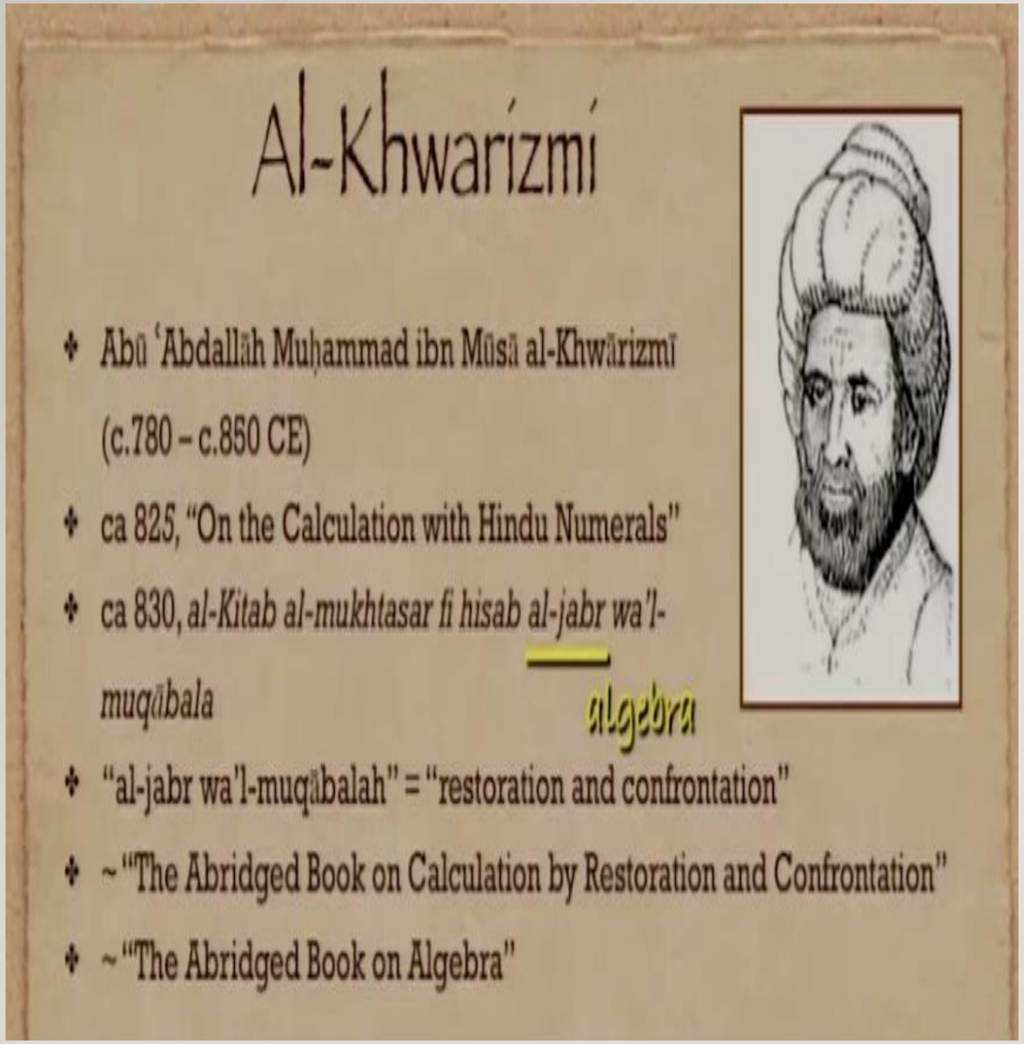
ASIM BALARABE YAZID (asimbyazid@nileuniversity.edu.ng)

NILE UNIVERSITY OF NIGERIA

THE FATHER OF ALGEBRA/ALGORITHM



- **Muḥammad ibn Mūsā al-Khwārizmī**[note 1] (Persian: محمد بن موسی خوارزمی, romanized: *Moḥammad ben Musā Khwārazmi*; c. 780 – c. 850), or **al-Khwarizmi** and formerly Latinized as ***Algorithmi***, was a Persian[6][7][8] polymath who produced vastly influential works in mathematics, astronomy, and geography. Around 820 CE he was appointed as the astronomer and head of the library of the House of Wisdom in Baghdad.[9]:14



Al-Khwarizmi

- Abū ʿAbdallāh Muḥammad ibn Mūsā al-Khwārizmī (c.780 – c.850 CE)
- ca 825, "On the Calculation with Hindu Numerals"
- ca 830, *al-Kitab al-mukhtasar fi hisab al-jabr wa'l-muqābala*

algebra

- "al-jabr wa'l-muqābalah" = "restoration and confrontation"
- ~ "The Abridged Book on Calculation by Restoration and Confrontation"
- ~ "The Abridged Book on Algebra"

- Al-Khwarizmi's popularizing treatise on algebra (*The Compendious Book on Calculation by Completion and Balancing*, c. 813–833 CE[10]:171) presented the first systematic solution of linear and quadratic equations. One of his principal achievements in algebra was his demonstration of how to solve quadratic equations by completing the square, for which he provided geometric justifications.[9]:14 Because he was the first to treat algebra as an independent discipline and introduced the methods of "reduction" and "balancing" (the transposition of subtracted terms to the other side of an equation, that is, the cancellation of like terms on opposite sides of the equation),[11] he has been described as the father[6][12][13] or founder[14][15] of algebra. The term *algebra* itself comes from the title of his book (the word *al-jabr* meaning "completion" or "rejoining").[16] His name gave rise to the terms *algorism* and *algorithm*,[17] as well as Spanish and Portuguese terms *algoritmo*, and Spanish *guarismo*[18] and Portuguese *algarismo* meaning "digit".

CLICK AND WATCH THE VIDEO

https://www.youtube.com/watch?v=oRkNaF0QvnI

WHAT IS ALGORITHMS

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

OPERATIONS

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

CHARACTERISTICS OF AN ALGORITHM

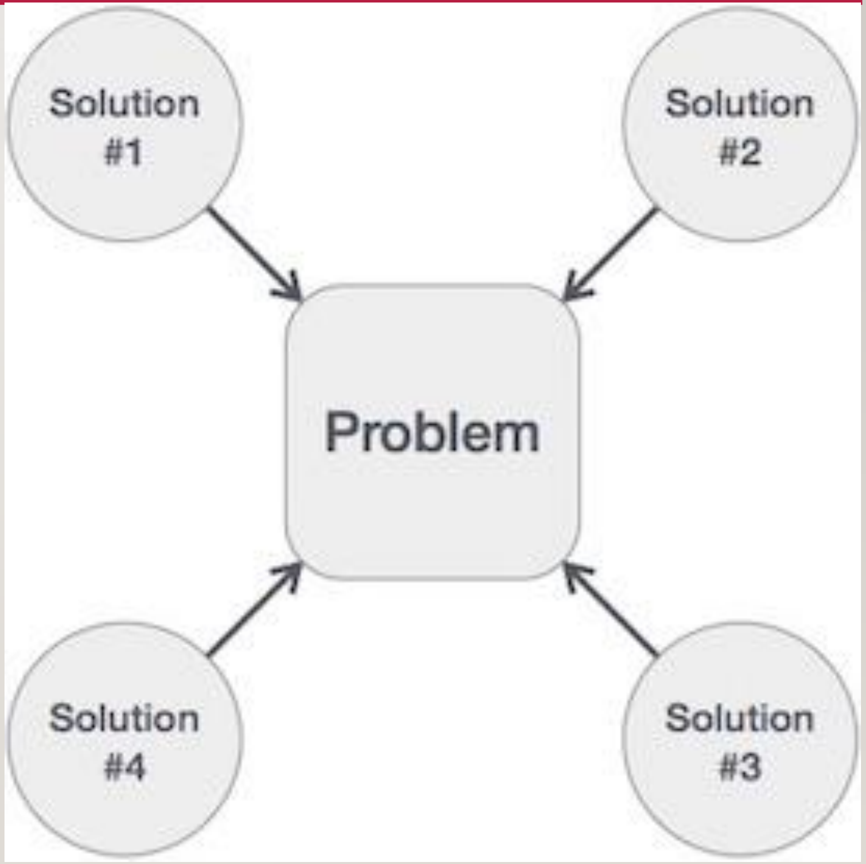
1. **INPUT**: An algorithm should have 0 or more well-defined inputs.
2. **OUTPUT**::An algorithm should have 1 or more well-defined outputs, and should match the desired output
3. **FINITENESS**: Algorithms must terminate after a finite number of steps
4. **CORRECTNESS**: Get the expected result or output
5. **EFFECTIVENESS**: Each step must be simple and should take a finite amount of time
6. **DEFINITENESS/UNAMBIGIOUS**:Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be well defined, clear and must lead to only one meaning.
7. **INDEPENDENT**:An algorithm should have step-by-step directions, which should be independent of any programming code.

WHY DESIGN ALGORITHM

- We design an algorithm to get a solution of a given problem. A problem can be solved in more than one ways.

Hence, many solution algorithms can be derived for a given problem.

The next step is to analyze those proposed solution algorithms and implement the best suitable solution.



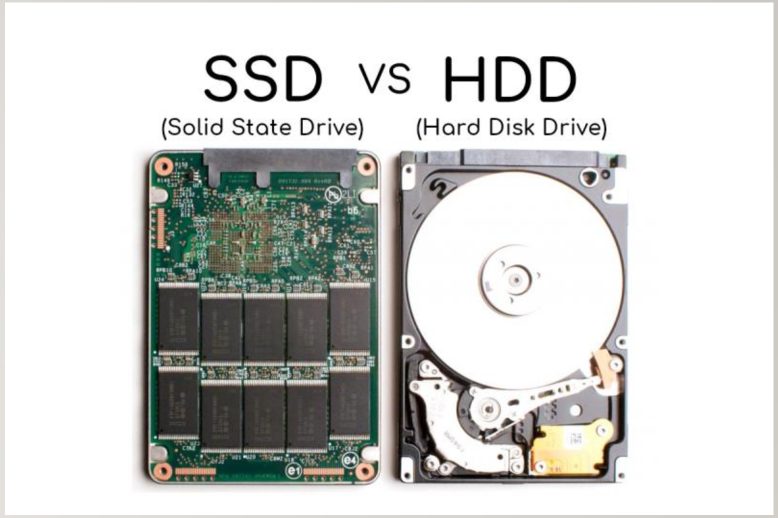
WHEN WE TALK ABOUT CAR A PERFORMANCE WHAT DO YOU THINK OF?

INTRODUCTION

- Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance.

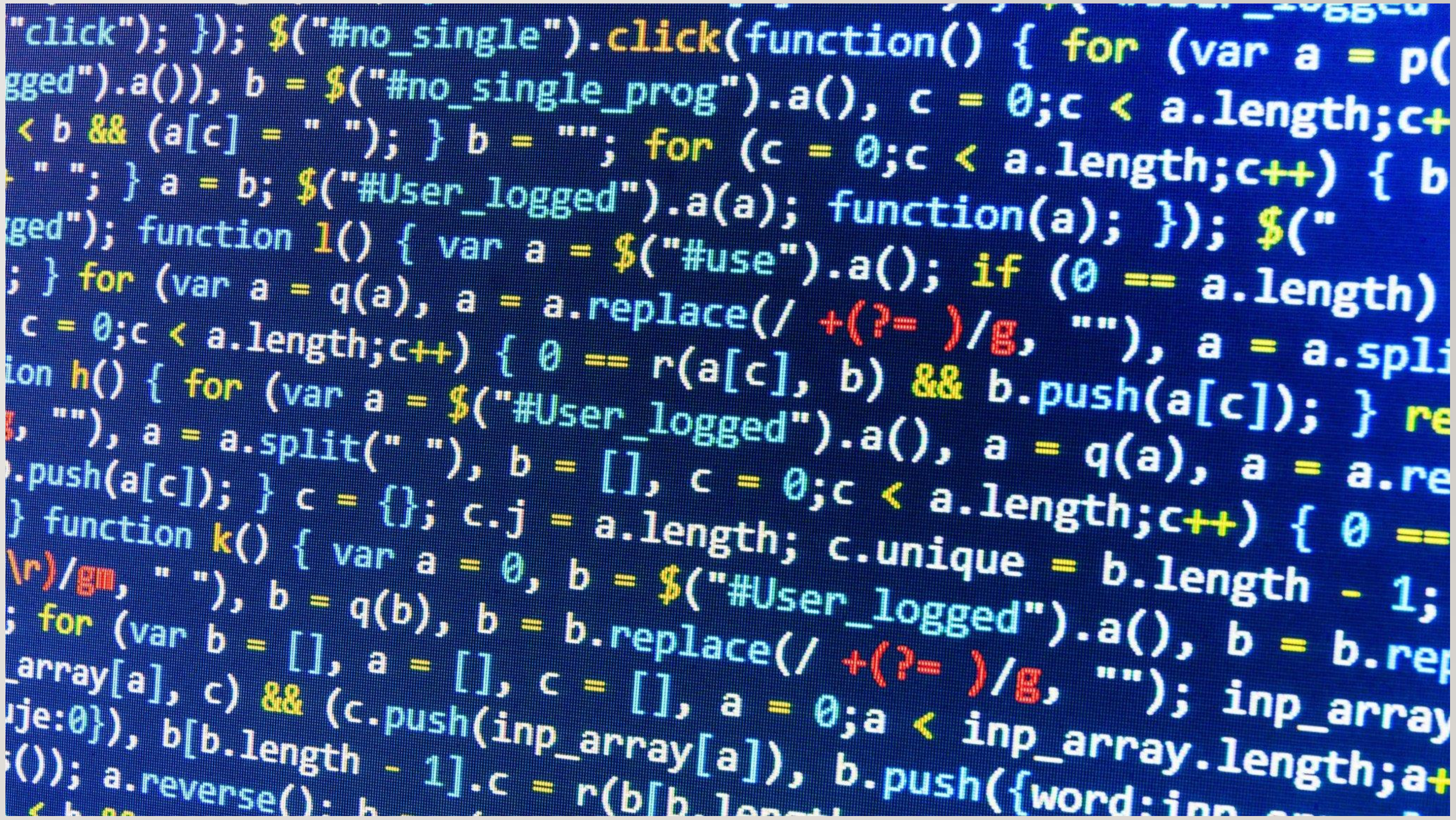
A POSTERIORI ANALYSIS

This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.



A PRIORI ANALYSIS

This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.



DIFFERENCE BETWEEN A POSTERIORI ANALYSIS AND A PRIORI ANALYSIS:

A Posteriori analysis	A priori analysis
Posteriori analysis is a relative analysis.	Piori analysis is an absolute analysis.
It is dependent on language of compiler and type of hardware.	It is independent of language of compiler and types of hardware.
It will give exact answer.	It will give approximate answer.
It doesn't use asymptotic notations to represent the time complexity of an algorithm.	It uses the asymptotic notations to represent how much time the algorithm will take in order to complete its execution.
The time complexity of an algorithm using a posteriori analysis differ from system to system.	The time complexity of an algorithm using a priori analysis is same for every system.
If the time taken by the algorithm is less, then the credit will go to compiler and hardware.	If the program running faster, credit goes to the programmer.

TIME COMPLEXITY

- WHEN WE SAY TIME WE DO NOT MEAN REAL-TIME (HOUR MINUTES & SECONDS)
- WE ARE REFERRING TO COMPUTER TIME COMMONLY KNOWN AS ASYMPTOTIC

- Complexity of an algorithm is analyzed in two perspectives: **Time** and **Space**.
- **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

TIME COMPLEXITY

- It’s a function describing the amount of time required to run an algorithm in terms of the size of the input. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.

SPACE COMPLEXITY

NOTE

- Space complexity is sometimes ignored because the space used is minimal and/or obvious, however sometimes it becomes as important an issue as time.

TYPES OF CASE SCENERIOS

- Usually, the time required by an algorithm falls under three types −
- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

- Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

NOTE:

- Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

ASYMPTOTIC NOTATIONS

- Execution time of an algorithm depends on the instruction set, processor speed, disk I/O speed, etc. Hence, we estimate the efficiency of an algorithm asymptotically.
- Time function of an algorithm is represented by **T(n)**, where **n** is the input size.

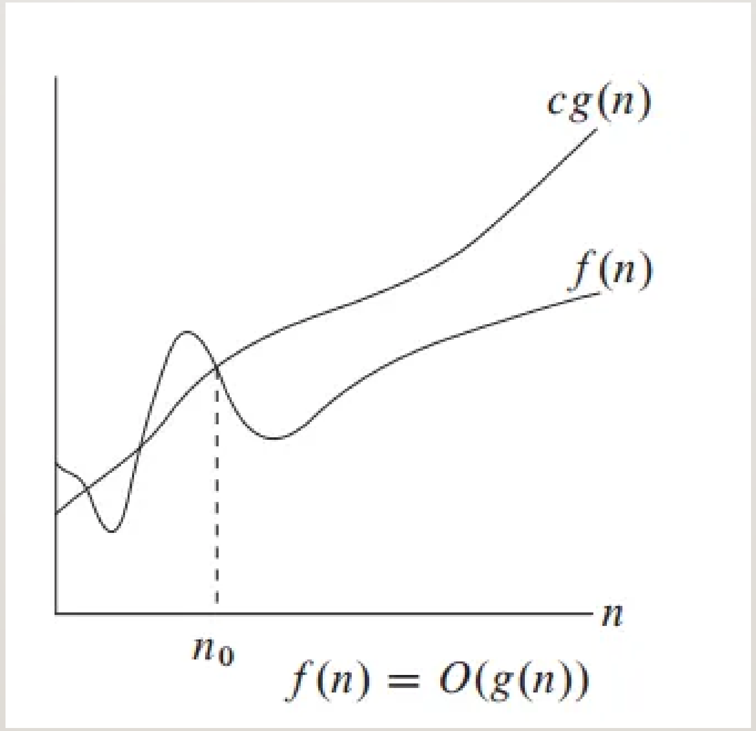
ASYMPTOTIC NOTATIONS

- Different types of asymptotic notations are used to represent the complexity of an algorithm. Following asymptotic notations are used to calculate the running time complexity of an algorithm.
- **O** − Big Oh
- **Ω** − Big omega
- **θ** − Big theta

BIG OH NOTATION, O

- The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.
-

BIG OH NOTATION, O



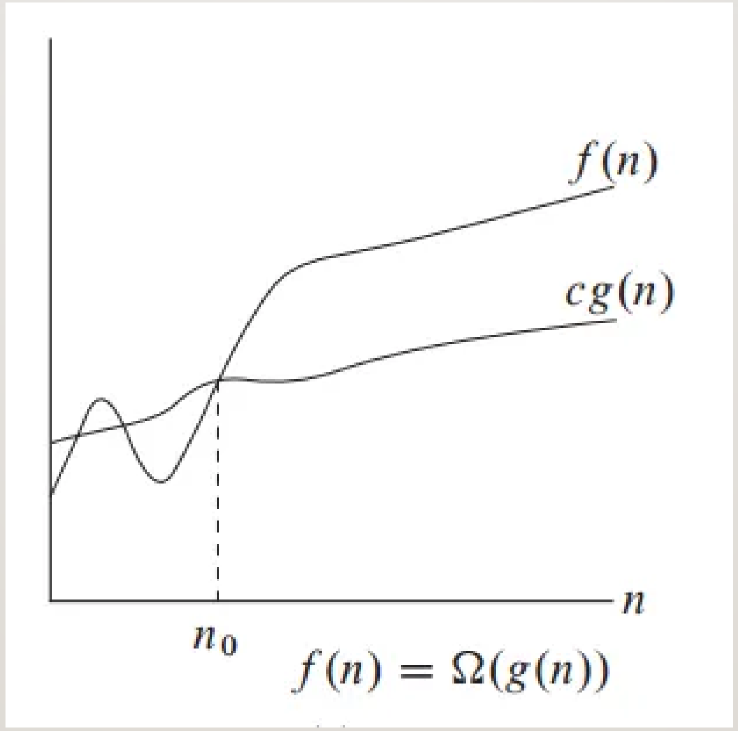
For a function ***f*(n)**

O(*f*(n)) = { *g*(n) : there exists c > 0 and n_0 such that *f*(n) ≤ c.*g*(n) for all n > n_0. }

OMEGA NOTATION, Ω

- The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

OMEGA NOTATION, Ω



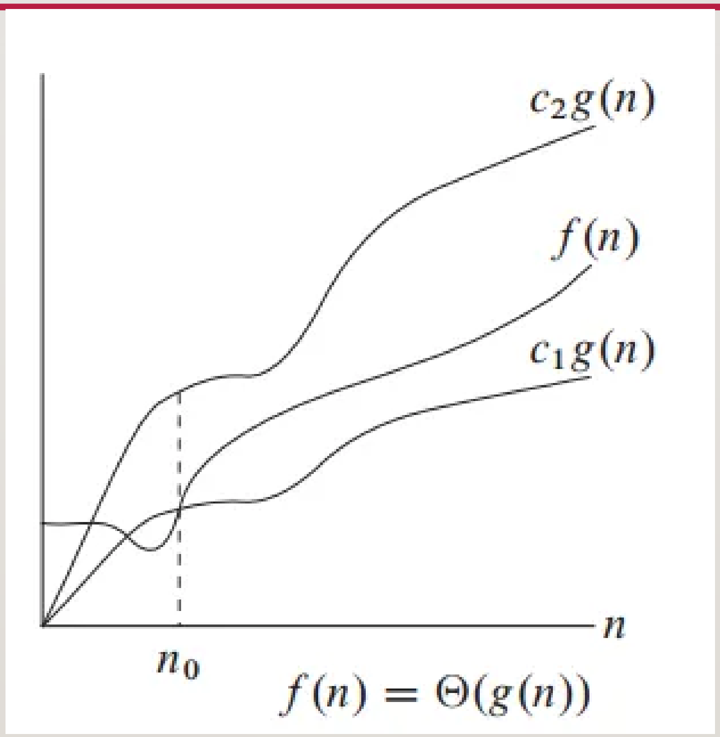
For a function ***f*****(n)**

Ω(*f*(n)) ≥ { *g*(n) : there exists c > 0 and n_0 such that *g*(n) ≤ c.*f*(n) for all n > n_0. }

THETA NOTATION, Θ

- The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −

THETA NOTATION, Θ



$\theta(f(n))$ = { $g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0$. }

COMMON ASYMPTOTIC NOTATIONS

Following is a list of some common asymptotic notations −

constant	−	O(1)
logarithmic	−	O(log n)
linear	−	O(n)
n log n	−	O(n log n)
quadratic	−	$O(n^2)$
cubic	−	$O(n^3)$
polynomial	−	$n^{O(1)}$
exponential	−	$2^{O(n)}$

GREEDY ALGORITHM

- An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.
- Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

EXAMPLE: COUNTING COINS

- This problem is to count to a desired value by choosing the least possible coins and the greedy approach forces the algorithm to pick the largest possible coin. If we are provided coins of ₦1, ₦2, ₦5 and ₦10 and we are asked to count ₦18 then the greedy procedure will be −
- **1** − Select one ₦10 coin, the remaining count is 8
- **2** − Then select one ₦5 coin, the remaining count is 3
- **3** − Then select one ₦2 coin, the remaining count is 1
- **4** − And finally, the selection of one ₦1 coins solves the problem

- Though, it seems to be working fine, for this count we need to pick only 4 coins. But if we slightly change the problem then the same approach may not be able to produce the same optimum result.
- For the currency system, where we have coins of 1, 7, 10 value, counting coins for value 18 will be absolutely optimum but for count like 15, it may use more coins than necessary. For example, the greedy approach will use 10 + 1 + 1 + 1 + 1 + 1, total 6 coins. Whereas the same problem could be solved by using only 3 coins (7 + 7 + 1)
- Hence, we may conclude that the greedy approach picks an immediate optimized solution and may fail where global optimization is a major concern.

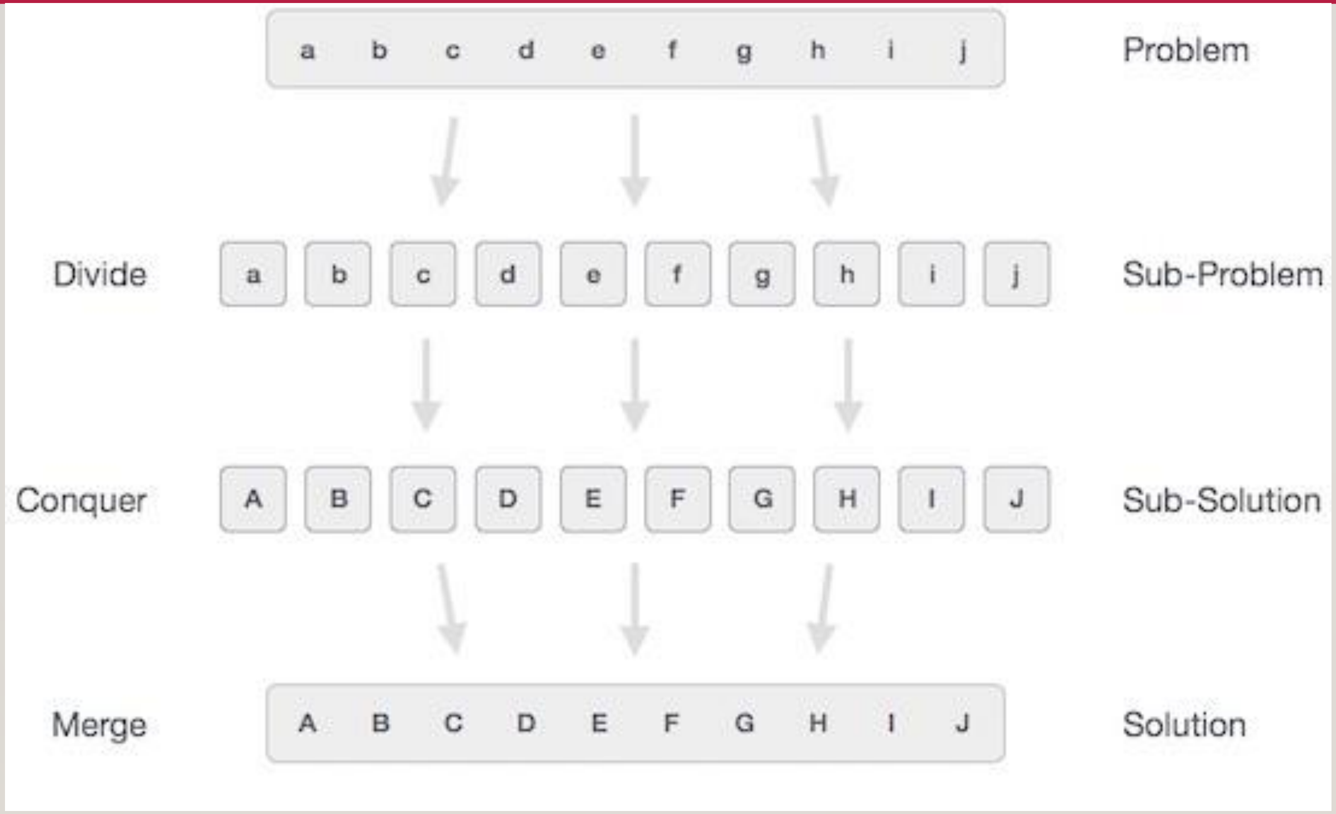
Most networking algorithms use the greedy approach. Here is a list of few of them −

1. Travelling Salesman Problem
2. Prim's Minimal Spanning Tree Algorithm
3. Kruskal's Minimal Spanning Tree Algorithm
4. Dijkstra's Minimal Spanning Tree Algorithm
5. Graph - Map Coloring
6. Graph - Vertex Cover
7. Knapsack Problem
8. Job Scheduling Problem

DIVIDE AND CONQUER

- In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

DIVIDE AND CONQUER



DIVIDE AND CONQUER

- The following computer algorithms are based on **divide-and-conquer** programming approach −
- Merge Sort
- Quick Sort
- Binary Search
- Strassen's Matrix Multiplication
- Closest pair (points)

DYNAMIC PROGRAMMING

- Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.

DYNAMIC PROGRAMMING

- Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.

So we can say that −

1. The problem should be able to be divided into smaller overlapping sub-problem.
2. An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
3. Dynamic algorithms use Memoization.

COMPARISON

- In contrast to greedy algorithms, where local optimization is addressed, dynamic algorithms are motivated for an overall optimization of the problem.

- The following computer problems can be solved using dynamic programming approach −
- Fibonacci number series
- Knapsack problem
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall
- Shortest path by Dijkstra
- Project scheduling

ASSIGNMENT: WATCH THIS VIDEO ON HARVARD CS50

- https://youtu.be/lFPedSR9wNU

NEXT CLASS

C