Course: Advanced Algorithm and Problem Solving

WEEK 1 - Introduction to Advanced Algorithms and Problem Solving

Lemlem Kassa (Ph.D.)

Addis Ababa Science and Technology University, Ethiopia

March ,2025

Course Description

- This course is designed to equip students with a deep understanding of computational problems, focusing on the principles used to design efficient algorithms and the skills to implement and analyze these algorithms using advanced data structures.
- Students will develop robust problem-solving skills necessary for tackling complex computational problems.
- Emphasis will be placed on the efficient utilization of data structures to implement algorithms, as well as the performance analysis and correctness of these algorithms.
- By the end of the course, students will be able to apply algorithmic techniques to solve real-world problems effectively.

Week 1: Introduction to Advanced Algorithms and Problem Solving

Contents

1. Introduction to Algorithm
2. Applications of algorithms
3. Design and Analysis of Algorithms
4. Asymptotic Analysis
5. Types of Time Complexity

Lecture Learning Outcome

- Understand algorithm and its relevance
- Recognize the basic characteristics of algorithm
- Understand different ways of expressing algorithms
- Recognize different application areas of Algorithm
- Understand algorithmic problem-solving process
- Understand the approaches of algorithm design and analysis

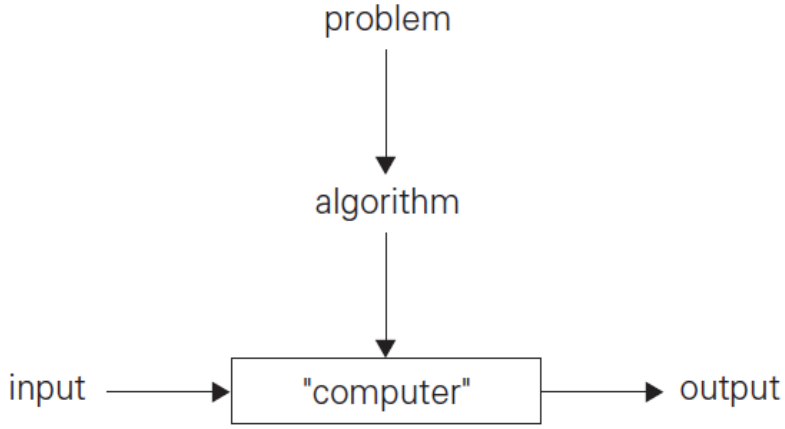
1. Introduction to Algorithm

What are algorithms?

- We need to create a program or set of instructions to solve a real-world problem, which may be considered an instance of a real-world problem.
- A well-defined computational procedure that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output*** in a unit amount of time.
- **Problem:** anything that requires the development of a program or a set of instructions.

Important Problem Types

- (i). Sorting (ii). Searching (iii). String processing (iv). Graph problems (v). Combinatorial problems (vi). Geometric problems (vii). Numerical problems



[1]. Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 3.

1. Introduction to Algorithm ….(Cont'd)

Example :

- Suppose that you need to sort a sequence of numbers into monotonically increasing order. Here is how we formally define the ***sorting problem***:

Input : 31; 41; 59; 26; 41; 58 , a correct sorting algorithm returns as output

Output : a sequence 26; 31; 41; 41; 58; 59. .

- Such an input sequence is called an ***instance*** of the sorting problem.
- ***An Instance of a problem*** consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem

[2]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009, page 5.

1. Introduction to Algorithm ….(Cont'd)

Benefits of Algorithm

- Understand the purpose of the program.
- Diagnose program errors.
- Helps to understand program flow
- Complex programs can be easily written.
- Scalability : It aids in our understanding of scalability. When we have a sizable real-world problem, we must break it down into small steps to analyze it quickly.
- Performance: The real world is challenging to break down into smaller steps. If a problem can be easily divided into smaller steps, the problem is feasible.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, Simplilearn, Dec 30, 2024

1. Introduction to algorithm ….(Cont'd)

Correct Vs incorrect Algorithm

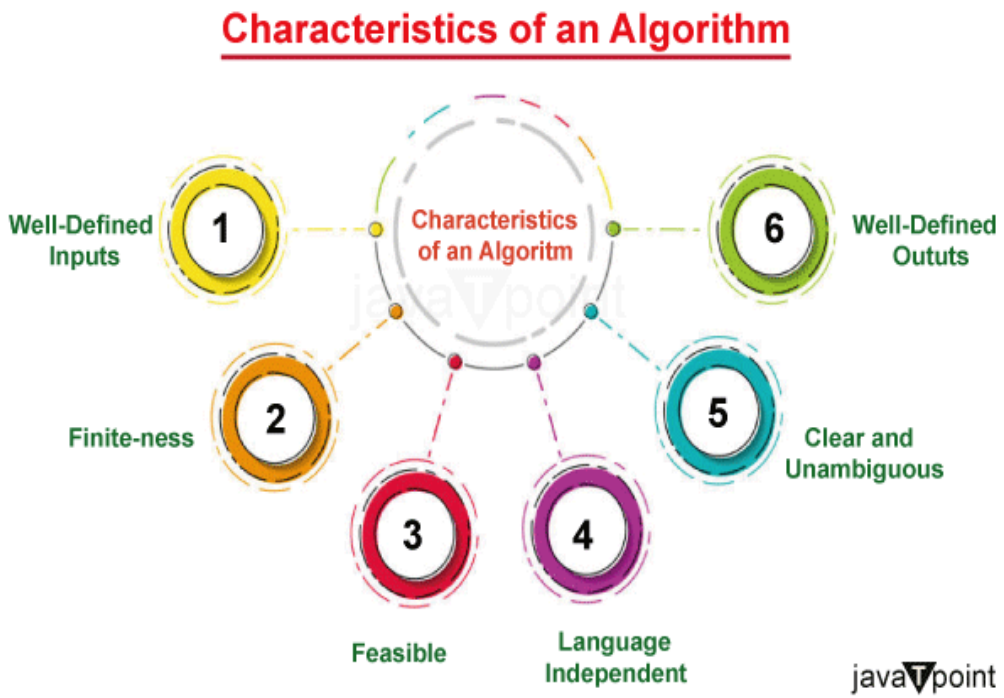
- **Correct Algorithm** : An algorithm for a computational problem is ***correct*** if, for every problem instance provided as input, it ***halts*** finishes its computing in unite time and outputs the correct solution to the problem instance.
  - A correct algorithm ***solves*** the given computational problem.
- **Incorrect algorithm** might not halt at all on some input instances, or it might halt with an incorrect answer.

[2]. Introduction to Algorithms, ThH. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009.

1. Introduction to algorithm ….(Cont'd)

Characteristics of algorithm:- factors that impact an algorithm's effectiveness-:

- **Clearly define the required inputs and Outputs:**
- **Definiteness:** clear and unambiguous;
- **Finiteness:** terminate after a finite number of steps;
- **Well-Defined and correctness:** Every step in the algorithm should be well-defined and straightforward.
- **Effectiveness:** instruction must be sufficiently basic that it can in principle be carried out by a person - must be feasible.
- **Generality :**It should offer a generic fix that manages a variety of inputs.



https://images.app.goo.gl/2zfXX7xAqoLmhEM88
How to Write an Algorithm, Tpoint Tech, 2025

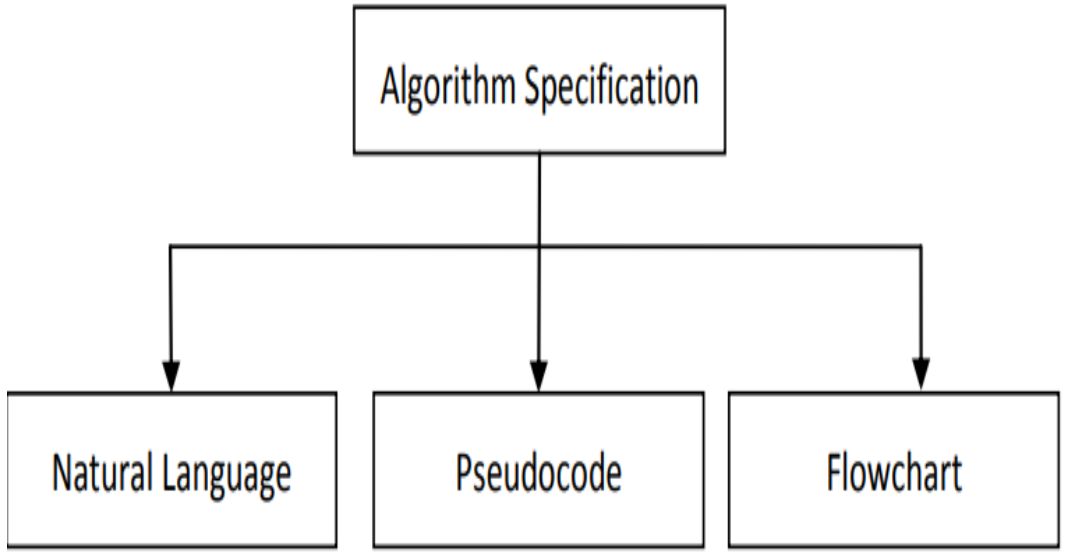
[4] . Algorithm in Data Structure: How to Solve Real Problems with Algorithms, AnalytixLabs, Pritha Bose, Aug 1-2023, https://www.analytixlabs.co.in/blog/algorithm-in-data-structure/

1. Introduction to algorithm ….(Cont'd)

Methods of Specifying an Algorithm

- **Natural Language :**
- A widely acceptable option for representing algorithms. However, using natural language also has certain drawbacks, as it can sometimes be very vague due to a lack of predefined structure.
- **Flow Charts :**A method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps..
- **Pseudocode :** It is a mixture of a natural language and programming language like constructs. It is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.

.



[1]. Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 12.

2. Applications of algorithms

Applications of algorithms in various fields:

1.Data Analysis and Machine Learning

- Algorithms are used in data analysis and machine learning to find patterns in big datasets and forecast outcomes.
  - **E.g.** support vector machines, decision trees, and neural networks, computers can learn from data and improve over time. These techniques are essential for recommendation systems, natural language processing, and picture recognition.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, simplilearn, Dec 30, 2024

2. Applications of algorithms ….(Cont'd)

Cont'd …..Applications of algorithms in various fields:

2. Cryptography & Security

- Cryptography algorithms safeguard data using encryption and decryption techniques, guaranteeing safe data storage and communication.
  - **E.g.** Algorithms such as SHA-256, AES, and RSA are commonly employed in data integrity maintenance, user authentication, and sensitive information security.

2. Applications of algorithms ….(Cont'd)

Cont'd …..Applications of algorithms in various fields:

3. Information Retrieval and Search Engines

- Search Engines can efficiently index and retrieve pertinent information using algorithms such as PageRank and Hummingbird.
- By prioritizing web pages according to their significance and relevancy, these algorithms assist users in locating the most relevant information available online.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, simplilearn, Dec 30, 2024

2. Applications of algorithms ….(Cont'd)

Cont'd …..Applications of algorithms in various fields:

4. **Optimization problems**

- Optimization methods are utilized to select the optimal answer from various options. In various industries, including banking, engineering, logistics, sophisticated issues are resolved using methods like gradient descent, linear programming, and genetic algorithms.
- These algorithms increase productivity, reduce expenses, and optimize resources for operations and decision-making.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, simplilearn, Dec 30, 2024

2. Applications of algorithms ….(Cont'd)

Cont'd …..Applications of algorithms in various fields:

5. Genomics and medical diagnostics

- Due to their ability to analyze medical images, forecast disease outbreaks, and recognize genetic changes, algorithms are indispensable in medical diagnostics.
- Algorithms help to speed up the sequencing and interpretation of genomic data, improving biotechnology and genomics research.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, Simplilearn, Dec 30, 2024

3. Design and Analysis of Algorithms

Random Access Memory (RAM) model of computation

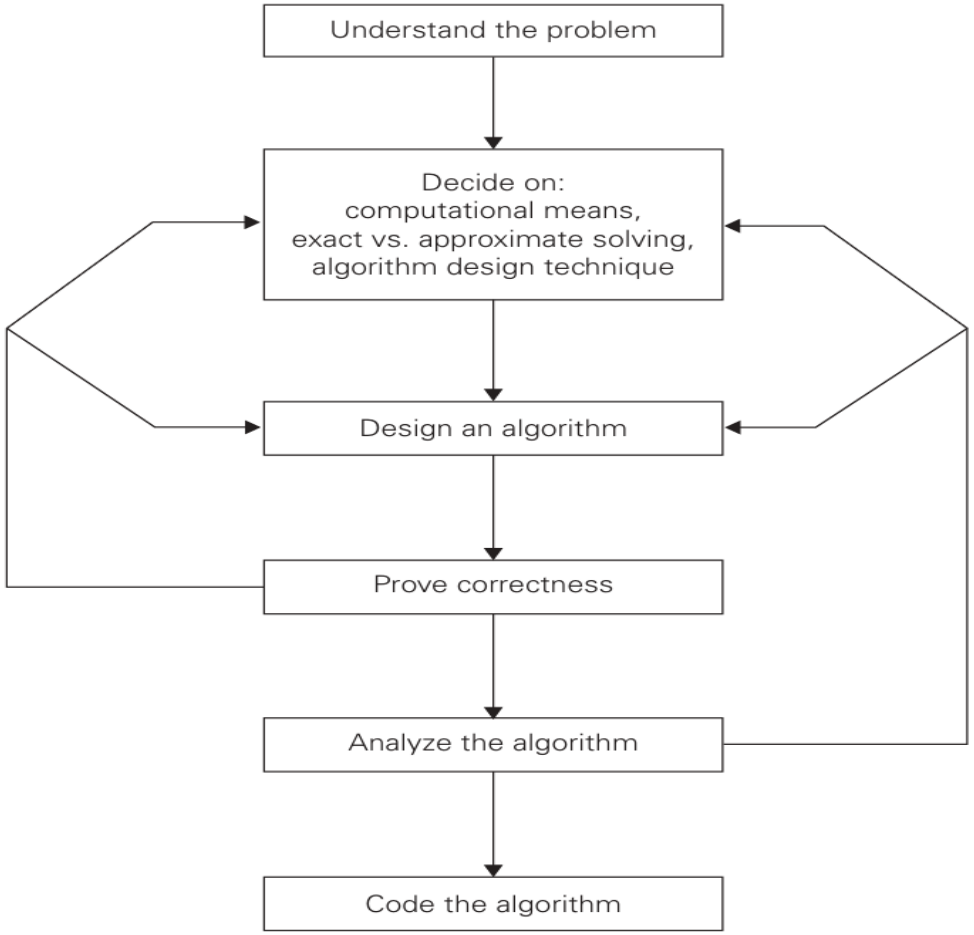
- Its central assumption is that instructions are executed one after another, one operation at a time.
- Accordingly, algorithms designed to be executed on such machines are called sequential algorithms.
- The central assumption of the RAM model does not hold for some newer computers that can execute operations concurrently, i.e., in parallel.
- Still, studying the classic techniques for design and analys of algorithms under the RAM model remains the cornerstone of algorithmics for the predictable future.

- Machine-independent algorithm design depends up on a hypothetical computer called **RAM**
- Machines changes so frequently, thus we measure the running time of an algorithm by counting up the number of steps it takes on a given problem instant

[2]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009, Page 23.

3. Design and Analysis of Algorithms ….(Cont'd)

Algorithmic problem-solving process



A) Understanding the Problem

- Before designing an algorithm is to understand completely the problem given.
  - It helps to understand how such an algorithm works and to know its strengths and weaknesses, especially if we have to choose among several available algorithms.

❖ **Remember that a correct algorithm is not one that works most of the time, but one that works correctly for all legitimate inputs.**

[1].Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 10.

3. Design and Analysis of Algorithms ….(Cont'd)

B) Decision to choose between solving the problem exactly or solving it approximately.
Exact algorithm and Approximation algorithm. Why ?

- **First,** there are important problems that simply cannot be solved exactly for most of their instances; examples include extracting square roots, and evaluating definite integrals.
- **Second,** available algorithms for solving a problem exactly can be unacceptably slow because of the problem's intrinsic complexity.
- **Third,** an approximation algorithm can be a part of a more sophisticated algorithm that solves a problem exactly.

[1].Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 10.

3. Design and Analysis of Algorithms ….(Cont'd)

C) Algorithm Design Techniques

- An algorithm design technique (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning algorithm design technique:
    - Provide guidance for designing algorithms for new problems, i.e., problems for which there is no known satisfactory algorithm.
    - Make it possible to classify algorithms according to an underlying design idea; therefore, they can serve as a natural way to both categorize and study algorithms.
    - **Example :** Divide and Conquer, Greedy Algorithms ,etc

[1].Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 10.

3. Design and Analysis of Algorithms ….(Cont'd)

D) Designing an Algorithm and Data Structures

- While the algorithm design techniques do provide a powerful set of general approaches to algorithmic problem solving, designing an algorithm for a particular problem may still be a challenging task
- Data structures remain crucially important for both design and analysis of algorithms.
- **Example:-** Divide and conquer is a common approach. Divide the problem into a number of subproblems that are smaller instances of the same problem.

[1].Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 10.

3. Design and Analysis of Algorithms ….(Cont'd)

E) Proving an Algorithm's Correctness

- We have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.
- For an approximation algorithm, we usually would like to be able to show that the error produced by the algorithm does not exceed a predefined limit.

[1].Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012, Page 10.

3. Design and Analysis of Algorithms ….(Cont'd)

Analyzing an Algorithm

- For an algorithm the most important is efficiency. In fact, there are two kinds of algorithm efficiency. They are:
  - **Time efficiency**, indicating how fast the algorithm runs, and
  - **Space efficiency**, indicating how much extra memory it uses.
- The efficiency of an algorithm is determined by measuring both time efficiency and space efficiency. So factors to analyze an algorithm are:
  - Time efficiency of an algorithm
  - Space efficiency of an algorithm
  - Simplicity of an algorithm
  - Generality of an algorithm

[5]. Difference Between Time Complexity and Space Complexity, Shiksha, Anshuman Singh , March 28, 2024, https://www.shiksha.com.

3. Design and Analysis of Algorithms ….(Cont'd)

- In the analysis of algorithms, understanding the performance under various circumstances is crucial. This is where the concepts of **Best, Worst, and Average Case Complexity** come into play.
- Analysis of an algorithm is the same thing as estimating the efficiency of the algorithm.
- There are two fundamental parameters based on which we can analyze the algorithm -→ Space and Time Complexity.

[5]. Difference Between Time Complexity and Space Complexity, Shiksha, Anshuman Singh , March 28, 2024, https://www.shiksha.com.

3. Design and Analysis of Algorithms ….(Cont'd)

- An algorithm is analyzed using **Time Complexity** and **Space Complexity.**
- Writing an efficient algorithm help to consume the minimum amount of time for processing the logic.
- Issues:
  - **Correctness** – *Does it work as advertised?*
  - **Time Efficiency** – *Are time requirements minimized?*
  - **Space Efficiency** – *Are space requirements minimized?*
  - **Optimality** – *Do we have the best balance between minimizing time and space?*

[5]. Difference Between Time Complexity and Space Complexity, Shiksha, Anshuman Singh , March 28, 2024, https://www.shiksha.com.

3. Design and Analysis of Algorithms ….(Cont'd)

- **Time Complexity analysis** a technique to measure how long an algorithm would take to complete given an input of size n; independent of the *machine, language, and compiler.*
- While complexity is usually in terms of time, it is also analyzed in terms of space i.e. algorithm's memory requirements.

Why Complexity Analysis is Required?

- It gives an estimated time and space required to execute a program.
- It is used for comparing different algorithms for different input sizes.
- It helps to determine the difficulty of a problem.

[5]. Difference Between Time Complexity and Space Complexity, Shiksha, Anshuman Singh , March 28, 2024, https://www.shiksha.com.

3. Design and Analysis of Algorithms ….(Cont'd)

How is Time Complexity Calculated?

a) Using Big O Notation

- Time complexity is most commonly expressed using Big O notation, which describes the upper bound of time required by an algorithm in terms of input size.

b) Counting Primitive Operations

- Time complexity can be calculated by counting the number of primitive operations an algorithm performs in relation to its input size.

c) Understanding Algorithm Structure

- Different algorithm structures - loops, series, and recursive calls, have distinct impacts on time complexity, which can be systematically analyzed.

3. Design and Analysis of Algorithms ….(Cont'd)

d) Worst, Best, and Average Case

- Time complexity can vary depending on the worst-case, best-case, and average-case input scenarios, which is kept in mind when calculating it.

e) Considering Constants and Lower-Order Terms

- While calculating time complexity, constants and lower-order terms are usually ignored as they become insignificant for large inputs.

[6]. Time Complexity: Importance & Best Practices, BotPenguin, https://botpenguin.com/glossary/time-complexity

3. Design and Analysis of Algorithms ….(Cont'd)

Bast practices to reduce time complexity:

- **Using Efficient Data Structures**
  - Selecting the right data structures for a task can drastically improve an algorithm's time complexity. For example, hash tables can make searching operations faster.
- **Avoiding Unnecessary Computations**
  - Time complexity can be lowered by avoiding unnecessary computations and making use of pre-computed values wherever possible.

[6]. Time Complexity: Importance & Best Practices, BotPenguin, https://botpenguin.com/glossary/time-complexity

3. Design and Analysis of Algorithms ….(Cont'd)

Efficient Use of Loops

- Nested loops can dramatically increase time complexity. Hence, they are used judiciously and only when necessary.

Utilizing Divide and Conquer

- The divide-and-conquer strategy often leads to more efficient solutions as it breaks the problem into smaller, more manageable parts.

- **Implementing Caching/Memoization**
  - By implementing caching or memoization techniques, repeated operations can be avoided, thus reducing overall time complexity

[6]. Time Complexity: Importance & Best Practices, BotPenguin, https://botpenguin.com/glossary/time-complexity

4. Asymptotic Analysis

- The study of the variations in the performance of the algorithm with the change in the order of the input size is called **Asymptotic Analysis**.
- **Asymptotic notations** : mathematical notations to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
    - In other words, it defines the mathematical limits of its run-time performance.
- Using the asymptotic analysis, we can easily conclude the
    - average-case, best-case, and worst-case scenario of an algorithm.

[2]. Introduction to Algorithms, MIT Press, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2022. Page 53.

4. Asymptotic Analysis …(cont'd)

- There are mainly three asymptotic notations for the complexity analysis of algorithms.

  They are as follows:

  - **Big-Oh (O) Notation** describes the worst-case scenarios.
  - **Big-Omega (Ω) Notation** reveals the best-case run time.
  - **Big-Theta (Θ) Notation** encapsulates the extremes and provides a tight and consistent range (average).

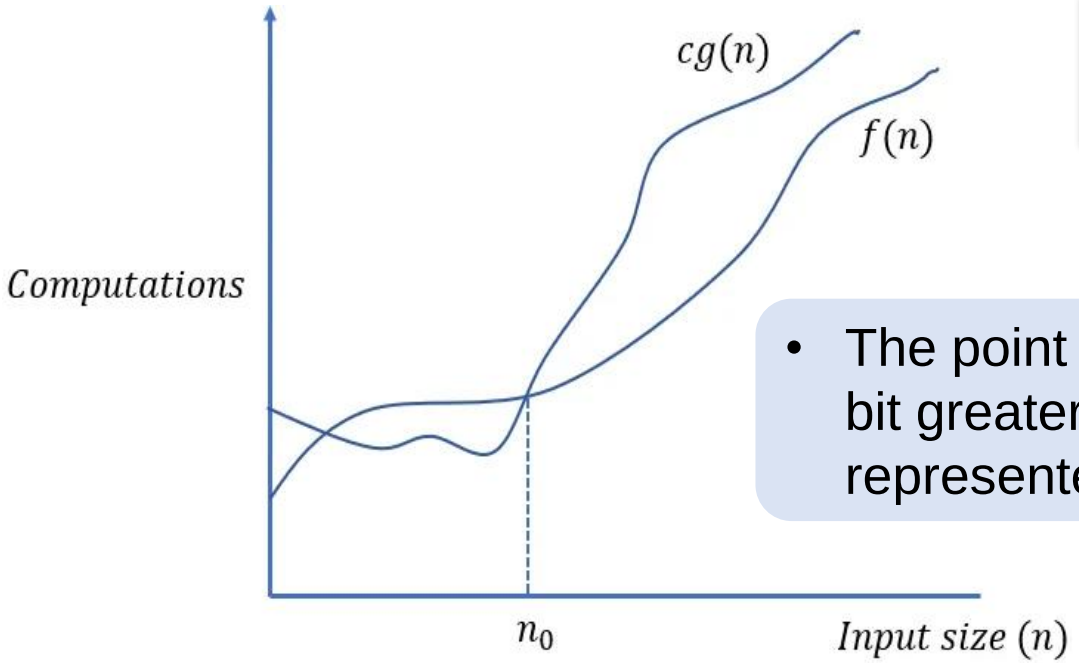
- **Note:** Ignore machine dependent constants, otherwise impossible to verify and to compare algorithms

[2]. Introduction to Algorithms, MIT Press, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2022. Page 53.

4. Asymptotic Analysis …(cont'd)

Big-Oh (O) Notation

- Moreover, Big-Oh notation is used to describe the **asymptotic upper bound** of an algorithm, **i.e. its worst-case scenario.**
- Finding a function which is slightly greater than our function f(n)
- An algorithm with a running time of f(n) is equivalent to O(g(n)) if and only if there are constants c and n0 such that the below expression is satisfied.



$$0 \leq f(n) \leq cg(n) \quad for\ n \geq n_0$$

- The point from which g(n) is always a bit greater than f(n) is threshold for f(n) represented as n_0

4. Asymptotic Analysis …(cont'd)

- Big-Oh represents the asymptotically tight ***upper bound of a function f(n).***

$$0 \leq f(n) \leq cg(n) \quad for\ n \geq n_0$$

- The reason that it is asymptotic is because we are only interested in large values of n, as it is for these values that the run time affects the computer's performance.
- Therefore, the inconsistencies present below n0 are ignored and we search for a function *c.g(n)*, which *f(n)* will tend to.

Example

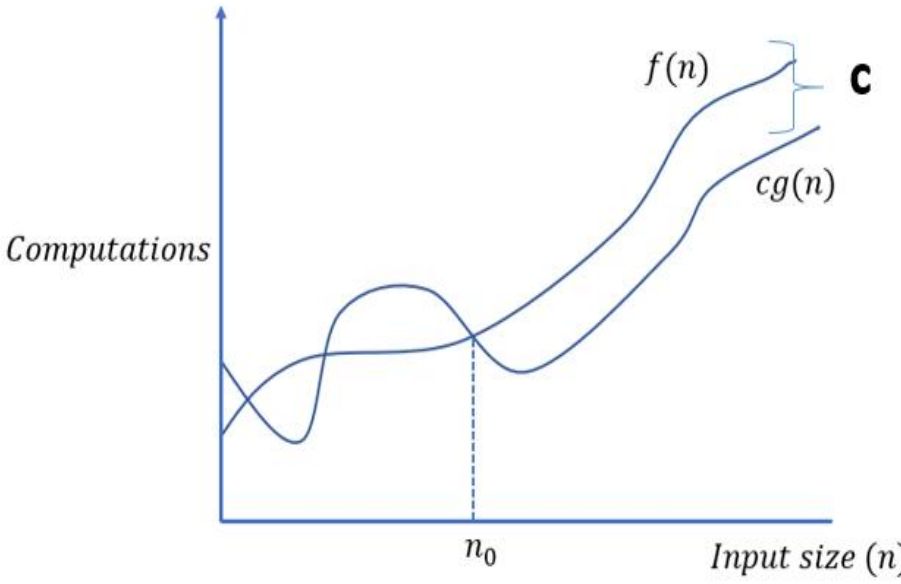
- Let's Find the upper bound of the function **f(n) = n² + 2.**
- Use the formal Definition **f(n)<=c.g(n)**

n² + 2<= c.(n²) // try for c=2 , and n>=1
n² + 2<= 2.(n²)
n² + 2<= O(n²), // for c=2 , and n>=1 however, O(logn) also true , but we should always find the closest function

4. Asymptotic Analysis …(cont'd)

Big-Omega (Ω) Notation

- Big-Omega notation is used to describe the **asymptotic lower bound** of an algorithm, i.e. **its best-case scenario.**
- An algorithm with a running time of f(n) is equivalent to Ω(g(n)) if and only if there are constants c and n0 such that the below expression is satisfied.



$$0 \leq cg(n) \leq f(n) \quad for\ n \geq n_0$$

- Big-Omega only describes the rate of growth at large input sizes.
- Therefore, for inputs above n0, cg(n) is defined as the asymptotic tight lower bound of f(n) with a time complexity of Ω(g(n)).

[2]. Introduction to Algorithms, MIT Press, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2022. Page 44.

4. Asymptotic Analysis …(cont'd)

Big-Omega (Ω) Notation …(cont'd)

Example

- Find the **lower bound** of the function $f(n) = 4n^2 - 3n + 2$.

$$0 \leq cg(n) \leq f(n)$$
$$0 \leq cn^2 \leq 4n^2 - 3n + 2 \text{ and } c = 4 \text{ so } n_0 = 2$$
$$\therefore 4n^2 - 3n + 2 = \Omega(n^2) \text{ with } c = 4 \text{ and } n_0 = 2$$

- Ω(logn) also make the condition true, however the closest function is $\Omega(n^2)$

- With c = 4, the inequality is only true if n0 = 2 or above.
- With this we have all the necessary information to describe the asymptotic lower bound of the function.

[2]. Introduction to Algorithms, MIT Press, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2022. Page 44.

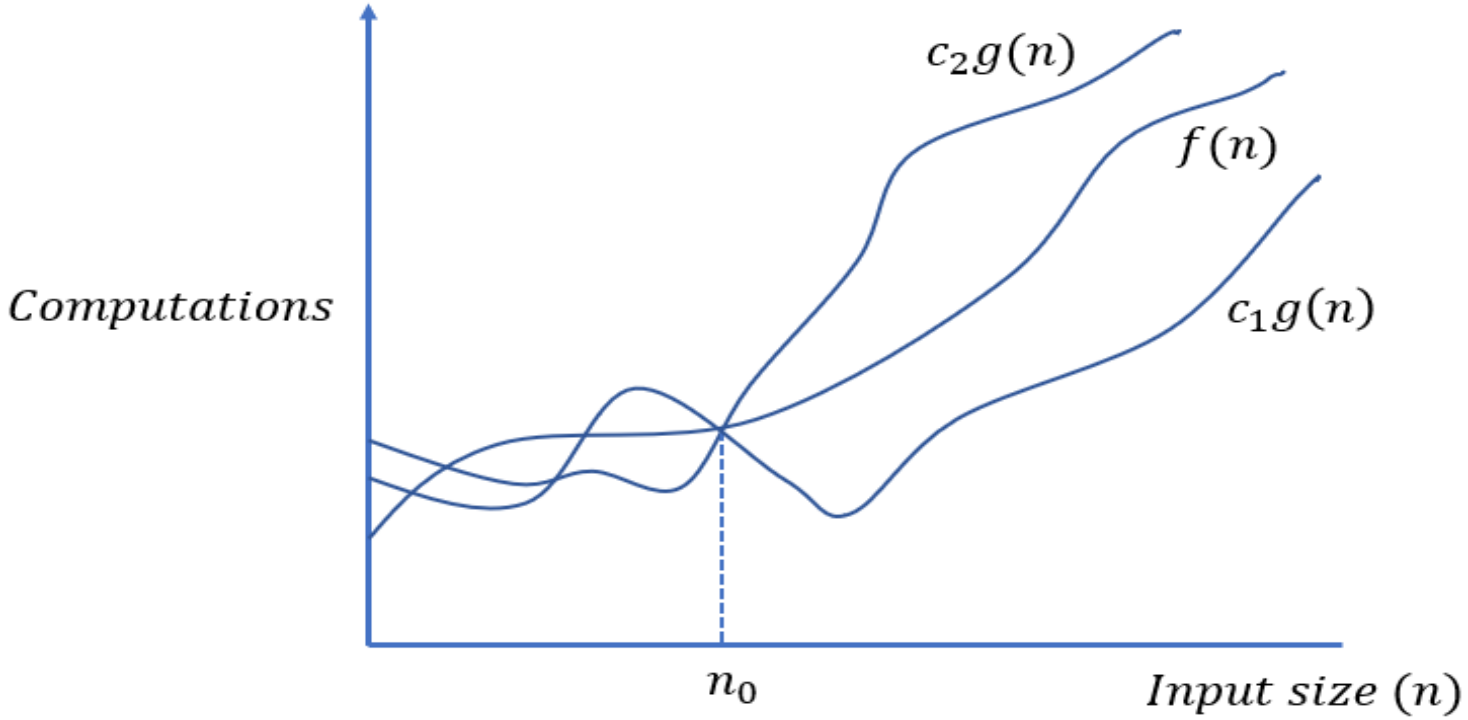
4. Asymptotic Analysis …(cont'd)

Big-Theta (Θ) Notation

- This notation describes **BOTH** the upper and lower bound of a function and is often referred to as the **average time complexity.**
- The purpose of this notation is to provide a **tight** and **consistent** bound that represents the lower and upper bound of a function (algorithm) in a single notation.
- If a function has a time complexity of Θ(n), it means that **both** the worst and best case are Ω(n) and O(n) respectively.
- Formal definition : **f(n) = Θ(g(n)), where c1, c2, n>0**

4. Asymptotic Analysis …(cont'd)

- **Big-Theta notation** is described mathematically as follows (a combination of **BOTH** the upper and lower bound of a function).



$$c_1g(n) \leq f(n) \leq c_2g(n)$$

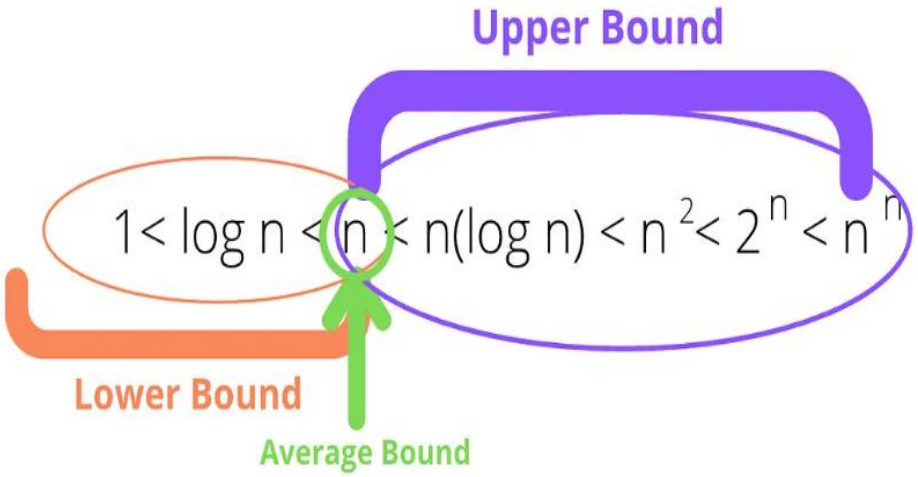
[2]. Introduction to Algorithms, MIT Press, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2022. Page 53.

4. Asymptotic Analysis …(cont'd)

Example: Big-Theta (Θ) Notation

- Find the Θ bound of **f(n) = 3n²- n**.

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$
$$c_1 g(n) \leq 3n^2 - n \leq c_2 g(n)$$
$$c_1 n^2 \leq 3n^2 - n \leq c_2 n^2$$
$$\therefore 3n^2 - n = \Theta(n^2) \; with \; c_1 = 2, c_2 = 3 \; and \; n_0 = 1$$



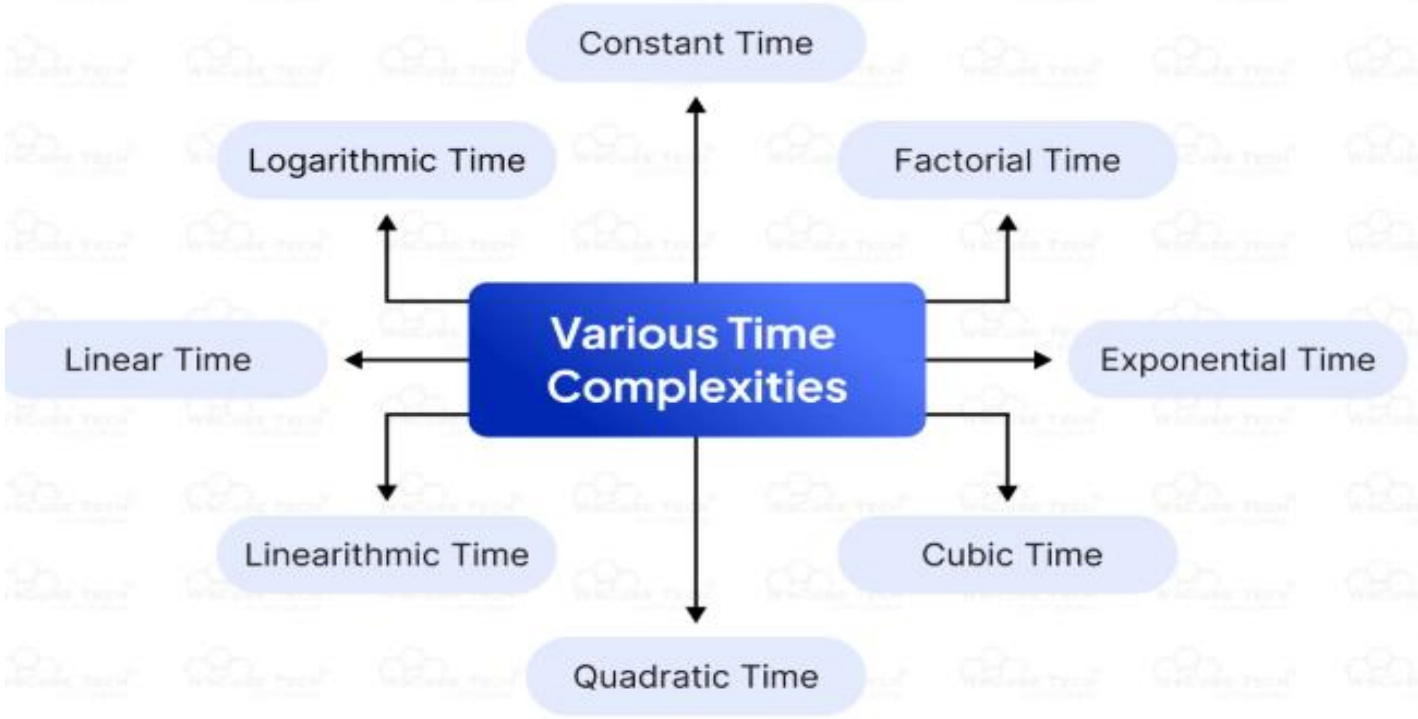
Asymptotic Notations, Hatice Karatay, Medium, December 25, 2021https://haticekaraty.medium.com/asymptotic-notations-16984e16e666

- In this example, we did exactly the same as in the Big-Oh and Big-Omega notation examples.
- Additionally, as the lower and upper bound shared the same time complexity, Big-Theta notation was used.

5. Types of Time Complexity

Different Types of Time Complexities

- The following are the primary types of time complexities that commonly appear in algorithm analysis:



[7]. Time Complexity , WSCubetech. https://deen3evddmddt.cloudfront.net/uploads/content-images/types-of-time-complexities.webp

5. Types of Time Complexity …Cont'd

1. Constant Time: O(1)

- An **algorithm** is said to have a constant time complexity when the time to complete does not depend on the size of the input data.

Examples: Accessing any element in an array by index. -→ X= arr[i]

2. Logarithmic Time: O(log n)

- Logarithmic time complexity occurs when the algorithm reduces the problem size by a factor with each step. Binary search is a classic example of logarithmic time complexity,

3**. Quadratic Time: O(n²) :**Algorithms with quadratic time complexity are generally characterized by two nested loops. **Examples:** Simple sorting algorithms like **bubble sort, insertion sort**, and **selection sort**.

[8]. Algorithm Design, Jon Kleinberg and Éva Tardos, Pearson, 2006, Page 46.

5. Types of Time Complexity …Cont'd

4. Cubic Time: $O(n^3)$

- Cubic time complexity appears in algorithms involving three nested loops.
- This is generally seen in more complex mathematical computations such as matrix multiplication or solving three-body problems in physics.

5. Factorial Time: O(n!)

- Factorial time complexity grows extremely fast with the increase in input size and is generally unsustainable even for relatively small n.
- This type of complexity is common in algorithms that generate all possible permutations of a dataset, such as solving the **traveling salesman problem** via a brute-force approach.

[8]. Algorithm Design, Jon Kleinberg and Éva Tardos, Pearson, 2006, Page 46.

5. Types of Time Complexity …Cont'd

Growth of function

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

- The running time of an algorithm is often expressed as a function of *n* *such as* $O(n), O(n^2)$
  - **n is** Linear time. Grows proportionally with *n*.

Key Points

- The choice of algorithm depends heavily on the input size n.
- For large n, algorithms with lower time complexity (e.g., $O(\lg n)$, $O(n)$, $O(n \lg n)$) are essential.
- The table demonstrates why exponential ($O(2^n)$) and factorial ($O(n!)$) algorithms are rarely used for large datasets.

[8] Algorithm Design, Jon Kleinberg and Éva Tardos, Pearson, 2006, Page34.

Summary

- Algorithm is a well-defined computational procedure that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output*** in a unit amount of time.
- Algorithm has several benefits such as helps to Understand the purpose of the program, diagnose program errors, understand program flow , understanding of scalability and Performance.
- Characteristics of algorithm impact an algorithm's effectiveness such as Clearly of inputs and Outputs, Definiteness, Finiteness, Well-Defined and correctness: well-defined and straightforward, Effectiveness and generality.
- There are mainly three asymptotic notations for the complexity analysis of algorithms. Such as Big-Oh (O) describes the worst-case scenarios, Big-Omega (Ω) Notation reveals the best-case run time. And Big-Theta (Θ) Notation.
- The choice of algorithm depends on heavily on the input size N
- The primary types of time complexities that commonly appear in algorithm analysis: such as, Constant Time: O(1), Logarithmic Time: O(log n), etc.

References

[1]. Introduction to the design & analysis of algorithms, Levitin A. Pearson,2012.

[2]. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, MIT Press, 2009.

[3]. What is an Algorithm? Algorithm Definition for Computer Science Beginners, Haroon Ahamed Kitthu, simplilearn, Dec 30, 2024. https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm

[4] Algorithm in Data Structure: How to Solve Real Problems with Algorithms, AnalytixLabs, Pritha Bose, Aug 1-2023, https://www.analytixlabs.co.in/blog/algorithm-in-data-structure/

[5]. Difference Between Time Complexity and Space Complexity, Shiksha, Anshuman Singh , March 28, 2024, https://www.shiksha.com.

[6]. Time Complexity: Importance & Best Practices, BotPenguin, https://botpenguin.com/glossary/time-complexity.

[7] Time Complexity , WSCubetech. https://deen3evddmddt.cloudfront.net/uploads/content-images/types-of-time-complexities.webp

[8] Algorithm Design, Jon Kleinberg and Éva Tardos, Pearson, 2006, Page34.

Thank you