Unit-1

Introduction to Data Structure

Computer Engineering Department SVIT Vasad

✓ Outline

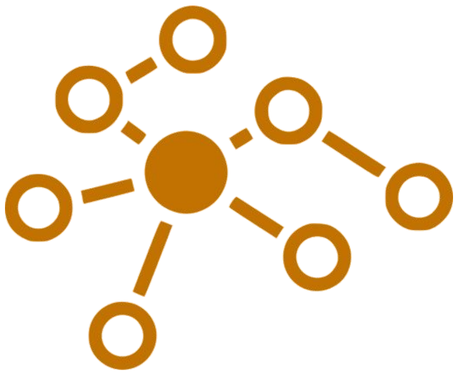
- Data Management concepts
- Data types
  - Primitive
  - Non-primitive
- Types of Data Structures
  - Linear Data Structures
  - Non Linear Data Structures
- Performance Analysis and Measurement
- Time analysis of algorithms
- Space analysis of algorithms

What is Data?

- **Data** is the raw / basic fact or entity that is utilized in calculation or manipulation.
- There are two different **types of data** **Numeric** data and **Alphanumeric** data.
- When a programmer collects such type of data for **processing**, he/she would require **to store data in computer's main memory**.
- **Information** is Processed Data. When collected data is processed then it becomes meaningful information with respect to context / requirement.
- The process of storing data items in computer's main memory is called ***representation.***
- **Data** to be processed must be **organized in a particular fashion**, these organization leads to structuring of data, and hence the mission to study the Data Structures.

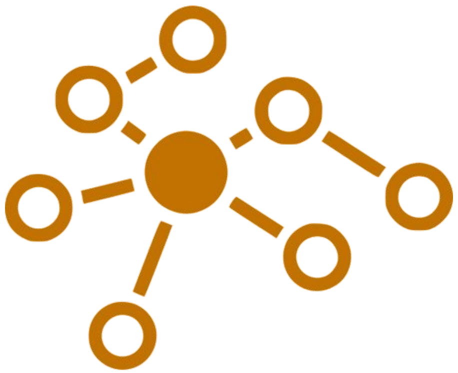
Data Structure Definitions

- A data structure is a **way of storing data** in a computer so that it can be used **efficiently** and it will allow the most **efficient algorithm** to be used
- Data Structure is a way of collecting and **organising data** in such a way that we can **perform operations** on these data in an **effective way**
- Data Structure is a **representation of the logical relationship** existing between individual elements of data.
- A Data structure is a way of organizing all data items that considers **not only the elements stored but also their relationship** to each other.
- A data structure is a class of data that can characterized by its organization and the operations that are defined on it. Hence
  - **Data Structure= Organized Data + Allowed Operations**



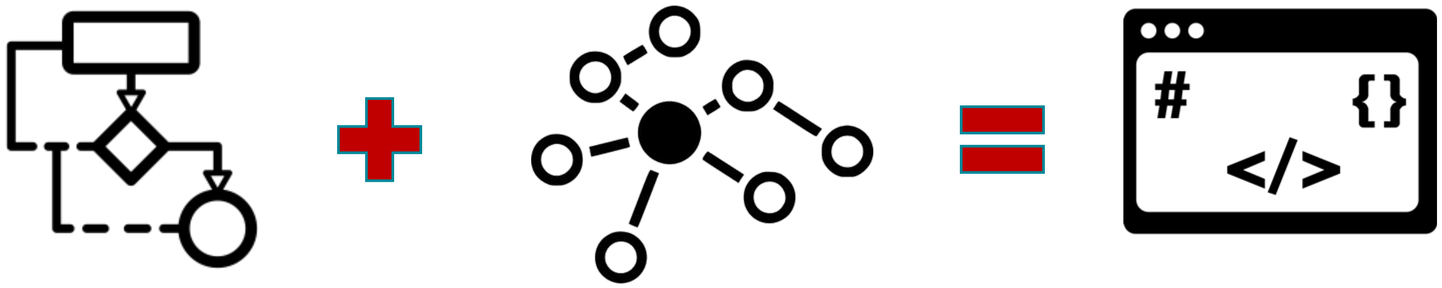
What is Data Structure?

- ***Data Structure*** is a representation of the logical relationship existing between individual elements of data.
- In other words, a data structure is a **way of organizing all data items** that **considers** not only the **elements stored** but also their **relationship to each other.**
- We can also define data structure as a **mathematical or logical model** of a particular **organization** of **data items.**
- Data Structure mainly specifies the following four things
    - Organization of Data
    - Accessing Methods
    - Degree of Associativity
    - Processing alternatives for information



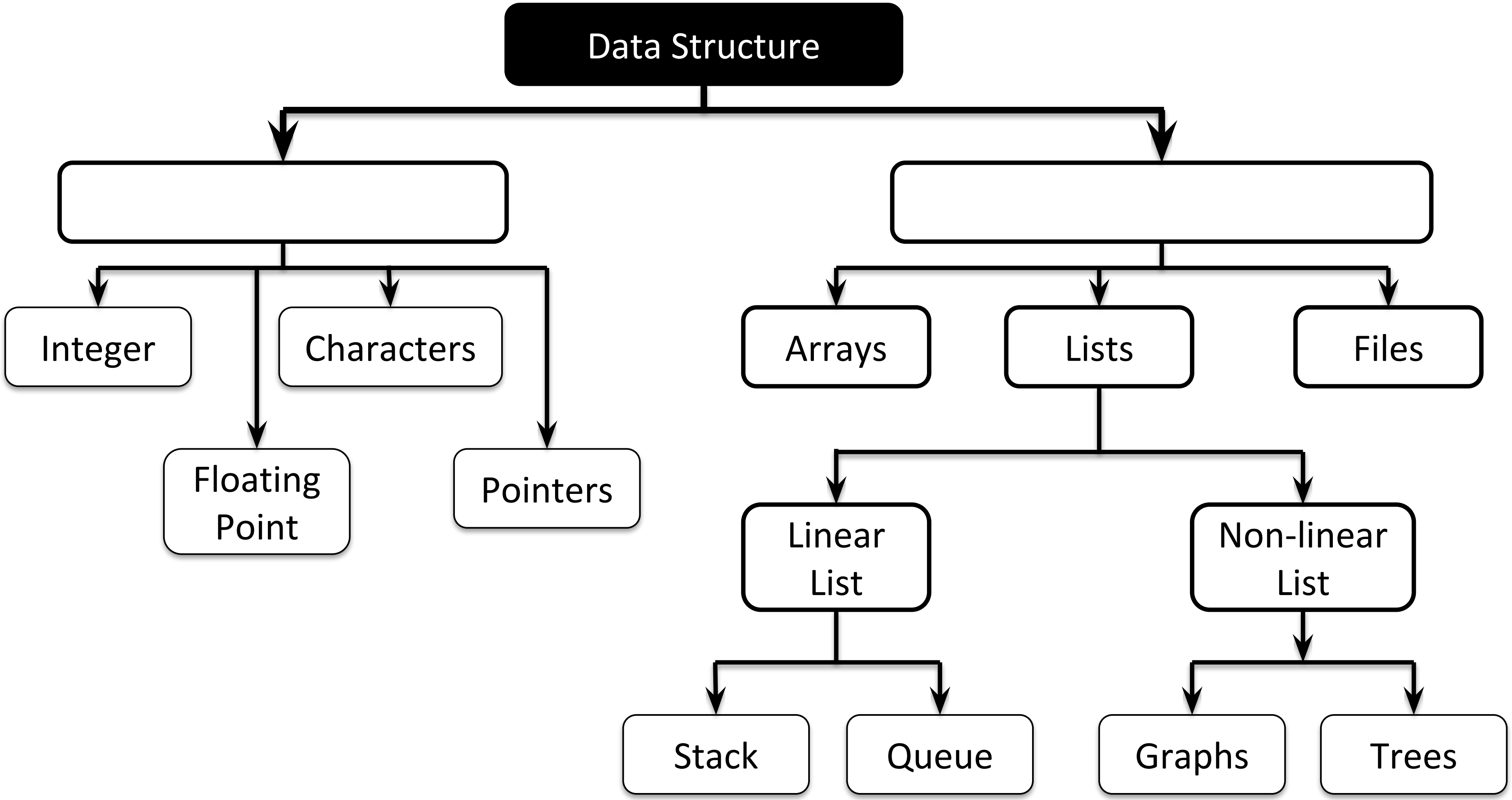
What is Data Structure? Cont..

- The **representation** of a particular data **structure in the memory** of a computer is called ***Storage Structure.***
- The storage structure **representation in auxiliary memory** is called as ***File Structure.***



Algorithm **Data Structure** **Program**

Classification of Data Structure



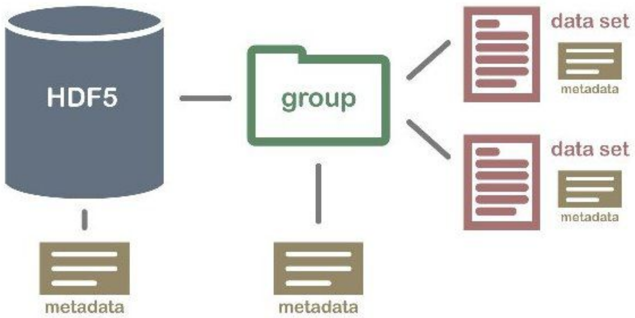
Primitive / Non-primitive data structures

- **Primitive data structures**
  - Primitive data structures are basic structures and are directly operated upon by machine instructions.
  - ***Integers***, ***floats***, ***character*** and ***pointers*** are examples of primitive data structures.
- **Non primitive data structure**
  - These are derived from primitive data structures.
  - The non-primitive data structures emphasize on structuring of a group of homogeneous or heterogeneous data items.
  - Examples of Non-primitive data type are ***Array***, ***List***, and ***File***.

Non primitive Data Structure

- **Array:** An array is a fixed-size sequenced collection of elements of the same data type.
- **List:** An ordered set containing variable number of elements is called as Lists.
- **File:** A file is a collection of logically related information. It can be viewed as a large list of records consisting of various fields.

Array



File

List

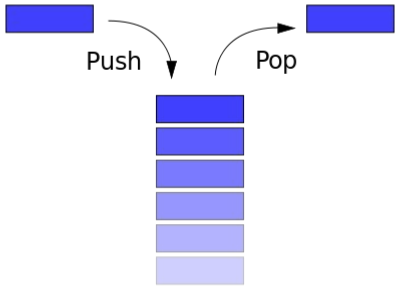
Linear / Non-Linear data structure

Linear data structures

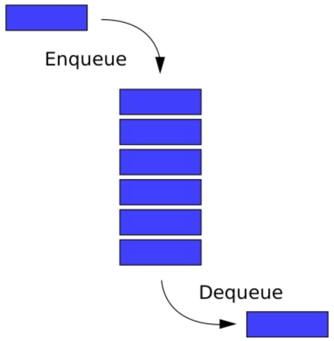
- A data structure is said to be Linear, if its elements are connected in linear fashion by means of logically or in sequence memory locations.
- Examples of Linear Data Structure are ***Stack*** and ***Queue***.

Nonlinear data structures

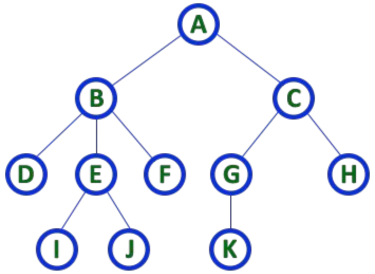
- Nonlinear data structures are those data structure in which data items are not arranged in a sequence.
- Examples of Non-linear Data Structure are ***Tree*** and ***Graph*****.**



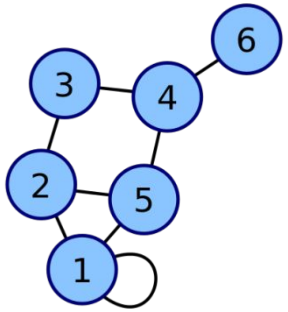
Stack



Queue



Tree



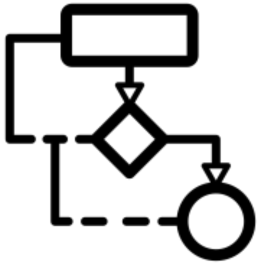
Graph

Operations of Data Structure

- **Create**: It results in reserving memory for program elements.
- **Destroy**: It destroys memory space allocated for specified data structure.
- **Selection**: It deals with accessing a particular data within a data structure.
- **Updation**: It updates or modifies the data in the data structure.
- **Searching**: It finds the presence of desired data item in the list of data items.
- **Sorting**: It is a process of arranging all data items in a data structure in a particular order.
- **Merging**: It is a process of combining the data items of two different sorted list into a single sorted list.
- **Splitting**: It is a process of partitioning single list to multiple list.
- **Traversal**: It is a process of visiting each and every node of a list in systematic manner.

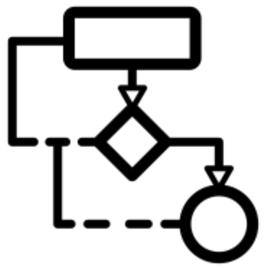
Algorithms - Introduction

- An Algorithm may be defined as a **finite sequence of instructions each of which has a clear meaning** and can be performed with a finite amount of effort in a finite length of time.
- The word algorithm originates from the Arabic word **Algorism** which is linked to the name of the Arabic Mathematician **Al Khwarizmi**.
- Al Khwarizmi is considered to be the **first algorithm designer for adding numbers**.
- The **efficiency** or **performance** of an algorithm relates to the resources required by it, such as **how quickly it will run**, or **how much computer memory it will use.**



Properties of an Algorithm

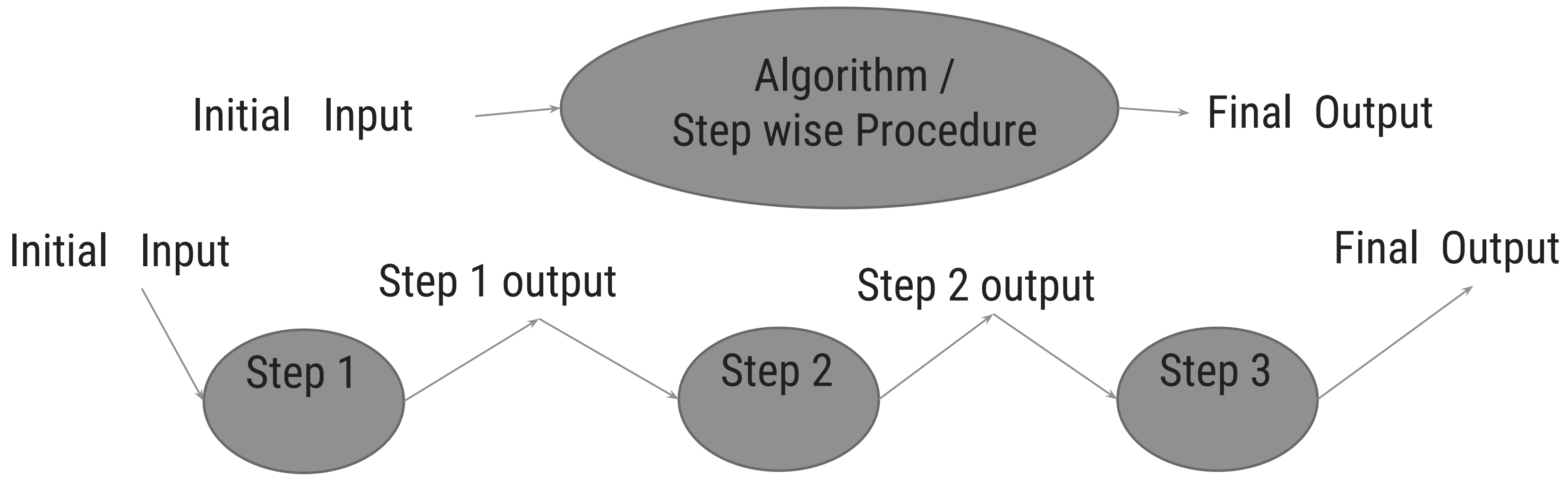
- **Finiteness**
  - An algorithm must terminate after finite number of steps.
- **Definiteness**
  - The steps of the algorithm must be precisely defined.
- **Generality**
  - An algorithm must be generic enough to solve all problems of a particular class.
- **Effectiveness**
  - The operations of the algorithm must be basic enough to be put down on pencil and paper.

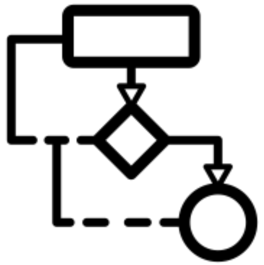


Properties of an Algorithm

Input-Output

- The algorithm must have certain initial and precise inputs, and outputs that may be generated both at its intermediate and final steps





Time and space analysis of algorithms

- Sometimes, there are more than one way to solve a problem.
- We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem.
- While analyzing an algorithm, we mostly consider time complexity and space complexity.
- ***Time complexity*** of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
- ***Space complexity*** of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.
- Time & space complexity depends on lots of things like hardware, operating system, processors, etc.
- However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm.

Calculate Time Complexity of Algorithm

- **Time Complexity** is most commonly **estimated** by **counting** the **number of elementary functions performed** by the algorithm.

- It can be computer either by an…

- **Empirical or Posteriori approach**
  - This approach calls for implementing the complete algorithm and executes them on a computer for various instances of the problem.
- **Theoretical or Apriori Approach**
  - This approach calls for mathematically determining the resources such as time and space needed by the algorithm, as a function of parameter related to the instances of the problem considered.

Calculate Time Complexity of Algorithm

- Apriori analysis computed the efficiency of the program as a function of the total frequency count of the statements comprising the program.

- **Example: Let us estimate the frequency count of the statement x = x+2 occurring in the following three program segments A, B and C.**

Total Frequency Count of Program Segment A

• Program Statements	• Frequency Count
………………………	
x = x+ 2	1
………………………	
Total Frequency Count	1

Time Complexity of Program Segment A is **O(1)**.

Total Frequency Count of Program Segment B

• Program Statements	• Frequency Count
………………………	
for k = 1 to n do	(n+1)
x = x+ 2;	n
end	n
………………………	………………………
Total Frequency Count	3n+1

Time Complexity of Program Segment B is **O(n)**.

Total Frequency Count of Program Segment C

• Program Statements	• Frequency Count
………………………	
for j = 1 to n do	(n+1)
for k = 1 to n do	(n+1)n
x = x+ 2;	n^2
end	n^2
end	n
………………………	………………………
Total Frequency Count	$3n^2+3n+1$

Time Complexity of Program Segment C is **O(n^2)**.

Calculate Time Complexity of Algorithm

- **Time Complexity** is most commonly **estimated** by **counting** the **number of elementary functions performed** by the algorithm.
- Since the algorithm's performance may vary with different types of input data,
  - hence for an algorithm we usually use the **worst-case Time complexity** of an algorithm because that is the maximum time taken for any input size.

Calculating Time Complexity

- Calculate Time Complexity of Sum of elements of List (One dimensional Array)

A is array, n is no of elements in array

```
SumOfList(A,n)
{
Line 1   total = 0
Line 2   for i = 0 to n-1
Line 3     total = total + A[i]
Line 4   return total
}
```

Line	Cost	No of Times
[redacted]		
[redacted]		
[redacted]		
[redacted]		

TSumOfList = 1 + 2 (n+1) + 2n + 1

= 4n + 4

We can neglate constant 4

= n

Time complexity of given algorithm is ***n*** unit time

- **Big oh(O):**

- **$f(n) = O(g(n))$** (read as f of n is big oh of g of n), if there exists a positive integer n0 and a positive number c such that **$|f(n)| \le c\,|g(n)|$ for all $n \ge n0$**.

f(n)	g(n)	
$16n^3 + 45n^2 + 12n$	n^3	$f(n) = O(n^3)$
$34n - 40$	n	$f(n) = O(n)$
50	1	$f(n) = O(1)$

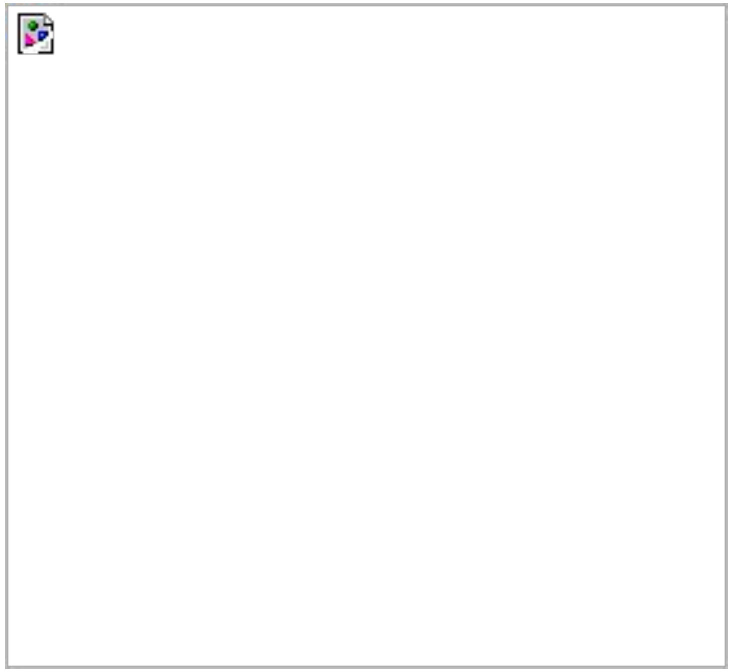
- Here g(n) is the upper bound of the function f(n).

Asymptotic Notation

- **Omega(Ω):**

- **f(n) = Ω(g(n))** (read as f of n is omega of g of n), if there exists a positive integer n0 and a positive number c such that **$|f(n)| \geq c\ |g(n)|$ for all $n \geq n0$**.

f(n)	g(n)	
$16n^3 + 8n^2 + 2$	n^3	$f(n) = \Omega\ (n^3)$
$24n + 9$	n	$f(n) = \Omega\ (n)$



- Here g(n) is the upper bound of the function f(n).

Asymptotic Notation

- **Theta(Θ) :**

- **$f(n) = \Theta(g(n))$** (read as f of n is theta of g of n), if there exists a positive integer n0 and two positive constants c1 and c2 such that **$c_1 |g(n)| \le |f(n)| \le c_2 |g(n)|$ for all $n \ge n_0$**.

f(n)	g(n)	
$16n^3 + 30n^2 - 90$	n^2	$f(n) = \Theta(n^2)$
$7.\ 2^n + 30n$	2^n	$f(n) = \Theta(2^n)$

- The function g(n) is both an upper bound and a lower bound for the function f(n) for all values of n, $n \ge n_0$.

Time Complexity

Complexity	Notation	Description
Constant	$O(1)$	Constant number of operations, not depending on the input data size.
Logarithmic	$O(\log n)$	Number of operations proportional of log(n) where n is the size of the input data.
Linear	$O(n)$	Number of operations proportional to the input data size.
Quadratic	$O(n^2)$	Number of operations proportional to the square of the size of the input data.
Cubic	$O(n^3)$	Number of operations proportional to the cube of the size of the input data.
Exponential	$O(2^n)$ $O(k^n)$ $O(n!)$	Exponential number of operations, fast growing.

Thank You