BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Data Structures and Algorithms Design

DSECLZG519

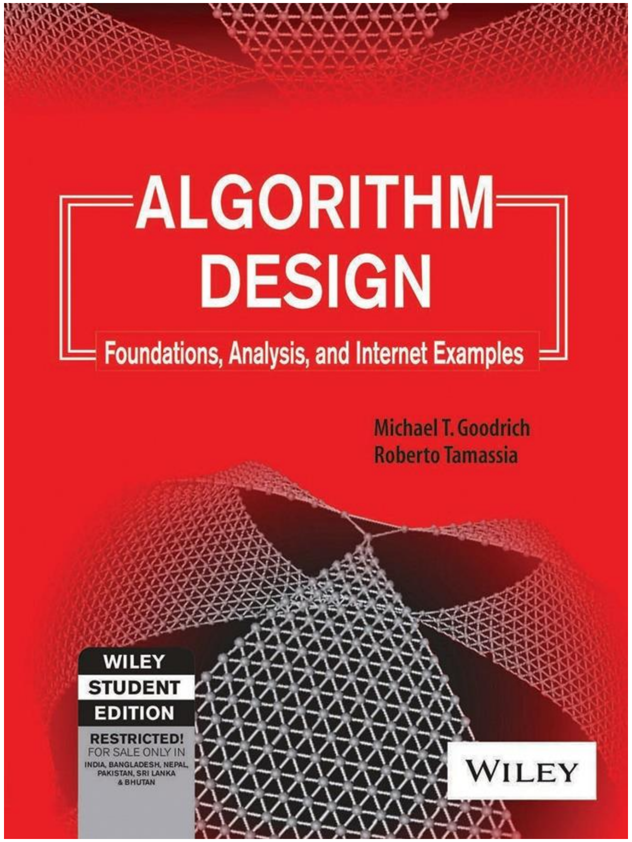
Parthasarathy

DSECLZG519 – CS#1
Introduction to DSECLZG519 & Algorithms

Agenda for CS #1

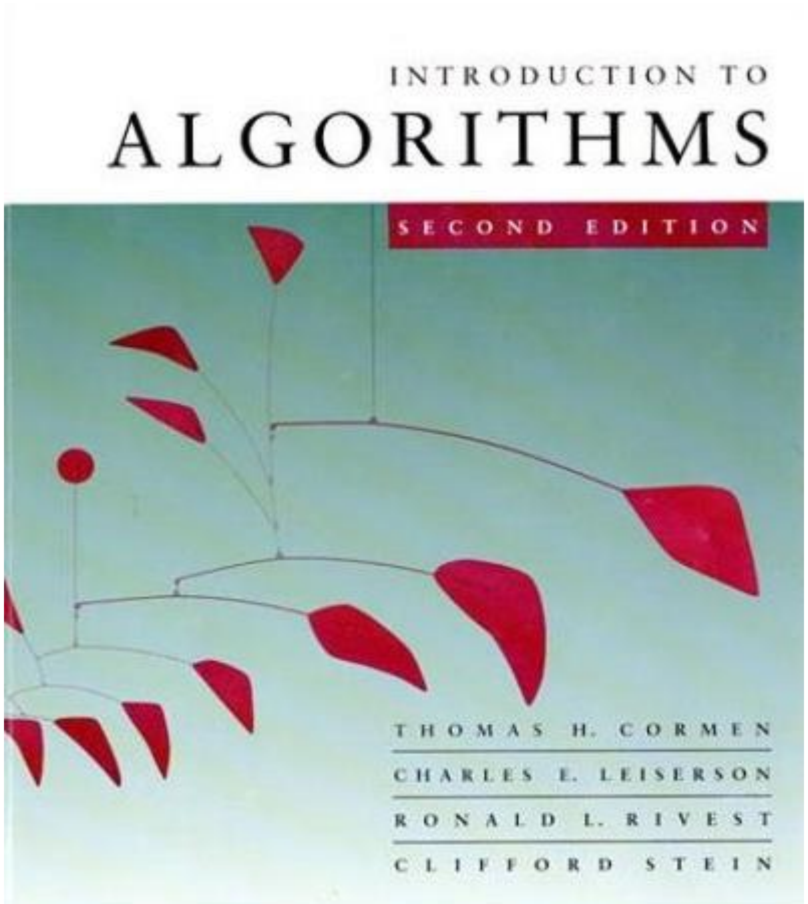
1) Introduction to DSECLZG519
   - Course handout
   - Books & Evaluation components
   - How to make the most out of this course ?
2) Motivation & Synergies between Data Science & DSAD
3) Introduction to Algorithms
   - Notion of an algorithm
   - Properties of an algorithm
   - Phases of program development
   - Why do we have to analyze them ?
4) Analysis of algorithm efficiency
   - Analysis Framework
   - Pseudocode & Counting primitives
   - Order of Growth
   - Asymptotic Notations
5) Q&A!

Text Book



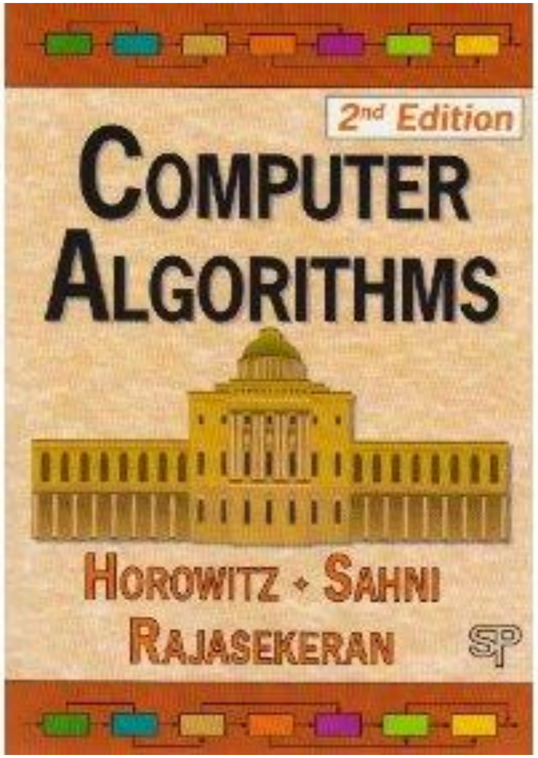
Michael T. Goodrich and Roberto Tamassia: *Algorithm Design: Foundations, Analysis and Internet examples* (John Wiley &Sons, Inc., 2002)

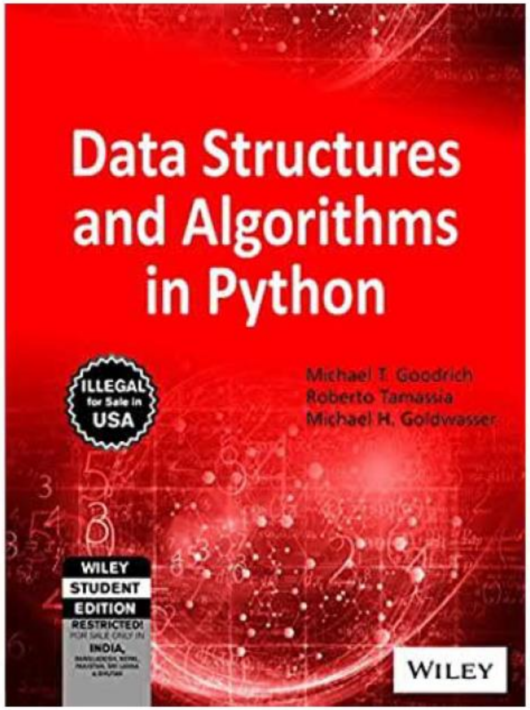
Reference Books



Also known as CLRS book

Reference Books





Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran. **Computer Algorithms**

Michael T. Goodrich and Roberto Tamassia and Goldwasser: *Data structures and Algorithms in Python* (John Wiley &Sons)

Ground Rules!

- Be Regular 😅
- Mentally present – Observe!! Listen!! 😉
- Keep your questions for the Q&A section …
- Use the [Discussion]() Forum in Canvas effectively
- Solve the exercises regularly!
- Go an extra mile ☺

$$\boxed{1^{365} = 1}$$

$$\boxed{1.01^{365} = 37.8}$$

Motivation & DSE – DSAD ?

Aspiring Data Scientist and allied areas ?

- Awesome! Even in such a role, you would be creating solutions that inevitably have code !!
- When you want to code → You need to have a strong understanding of Data structures and algorithms.
- Data Structures and Algorithms Knowledge give us the ability to improve our solution to the problem and the ability to write much better and efficient code.
- But most importantly, it helps to build problem solving mindset …
- Thus, learning Data Structures and Algorithms can be a major learning curve for any computer science / data science student.

What is an Program ?

Algorithm

An algorithm is a *step-by-step procedure* for solving a problem in a finite amount of time.

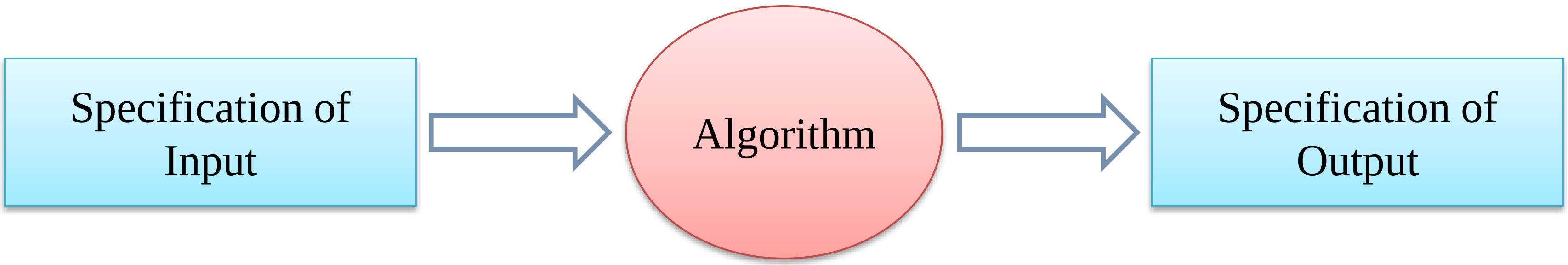
Data Structures

Is a *systematic way of organizing and accessing data, so that data can be used efficiently*

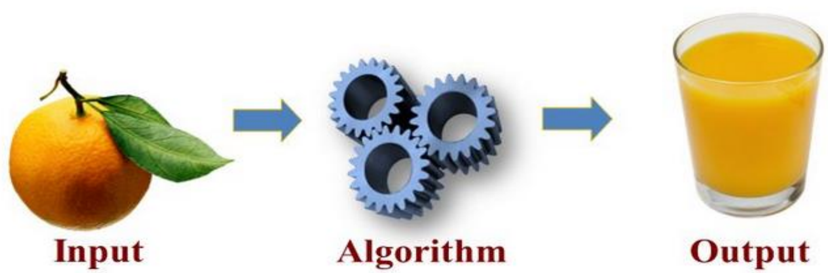
Algorithms + Data Structures = Program

An algorithm is defined as a finite sequence of unambiguous instructions followed to accomplish a given task.

Algorithmic Solution



- Algorithm describes actions on the input instance.
- Infinitely many correct algorithm for the same problem.
- Infinite number of input instances satisfying the specification.



Properties of an Algorithm

- Input : Each algorithm should have zero or more inputs
- Output : The algorithm should produce correct results. Atleast one output has to be produced
- Definiteness : Each instruction should be clear and unambiguous
- Effectiveness : The instructions should be simple and should transform the given input to the desired output.
- Finiteness : The algorithm must terminate after a finite sequence of instructions

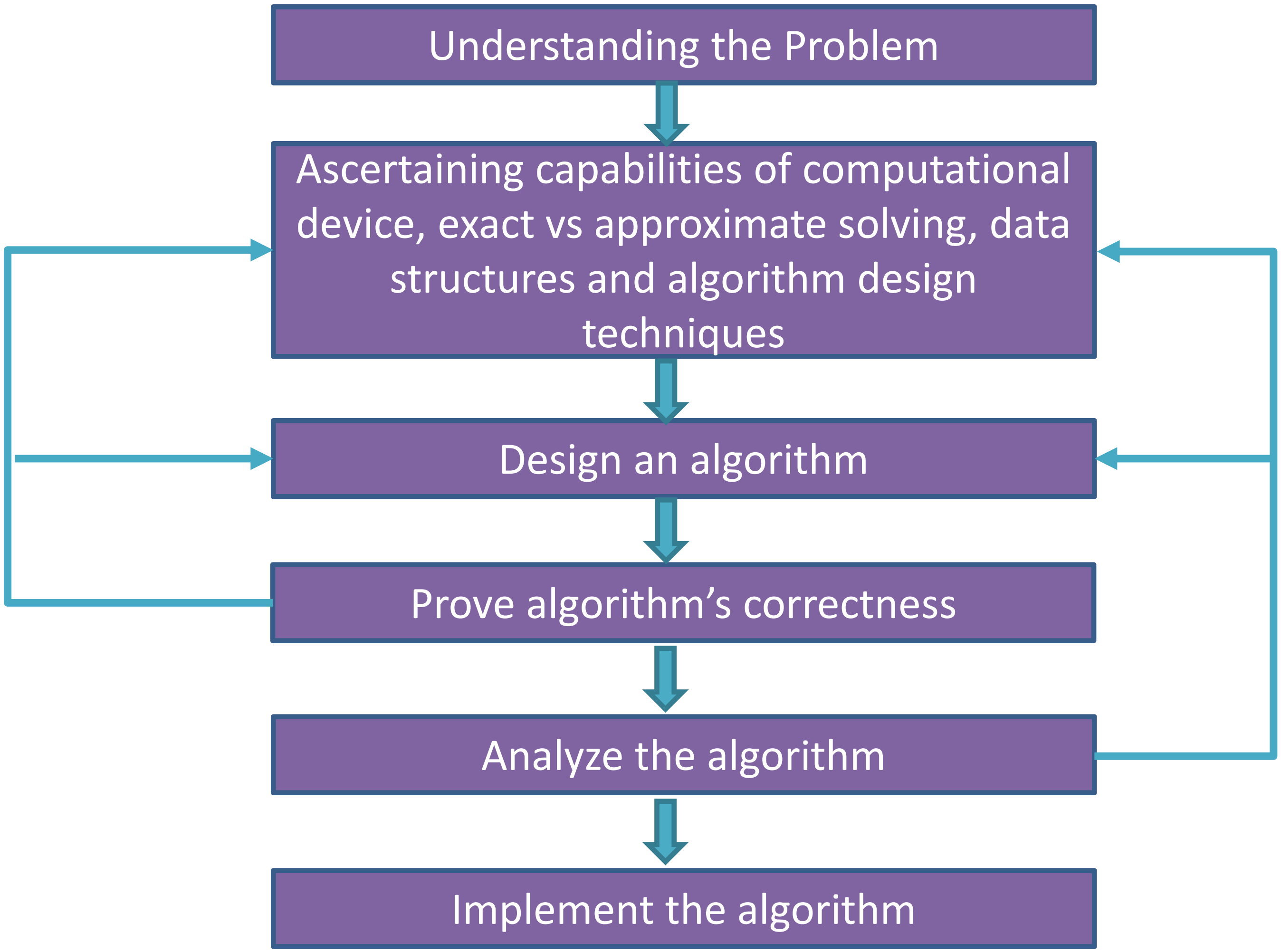
Example of an Algorithm

- Let us first take an example of a real-life situation for creating algorithm. Here is the algorithm for making tea!

1. Put the teabag in a cup.
2. Fill the kettle with water.
3. Boil the water in the kettle.
4. Pour some of the boiled water into the cup.
5. Add milk to the cup.
6. Add sugar to the cup.
7. Stir the tea.
8. Drink the tea.

- Some steps like 5,6 can be interchanged but some like 3,8 cannot be interchanged.

Phases of Program Development



Data Structures Outlook

Name of array (Note that all elements of this array have the same name, **c**)

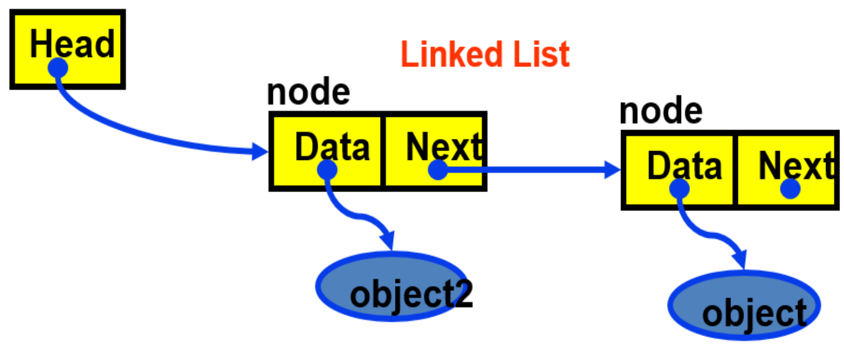
Array

Linearly Ordered Set

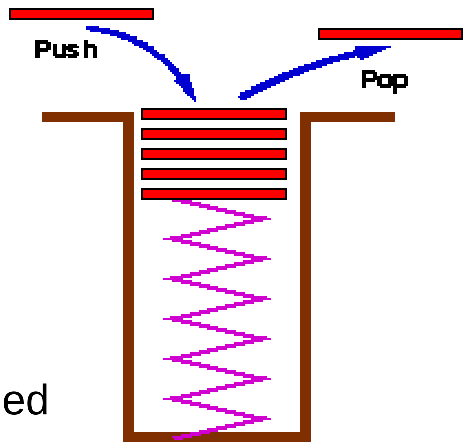
c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array **c**

*A **data structure** is a particular way of organizing data in a computer so that it can be used effectively.*



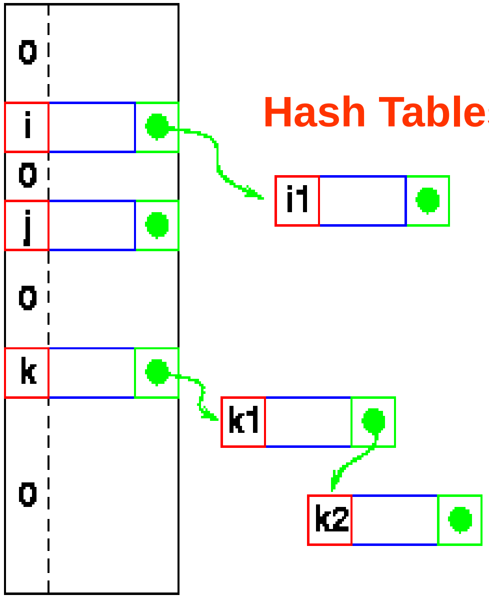
Data Structures Outlook



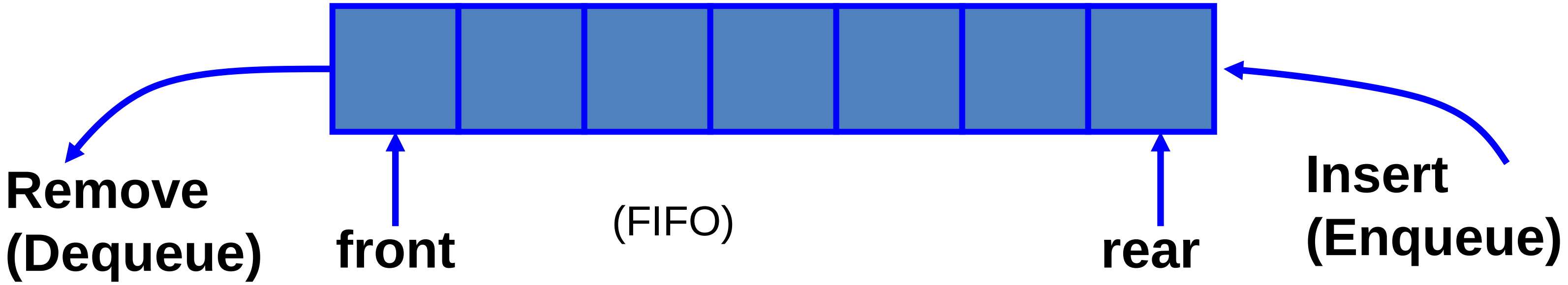
Stack

Set with delete operation specified (LIFO)

Hash Tables

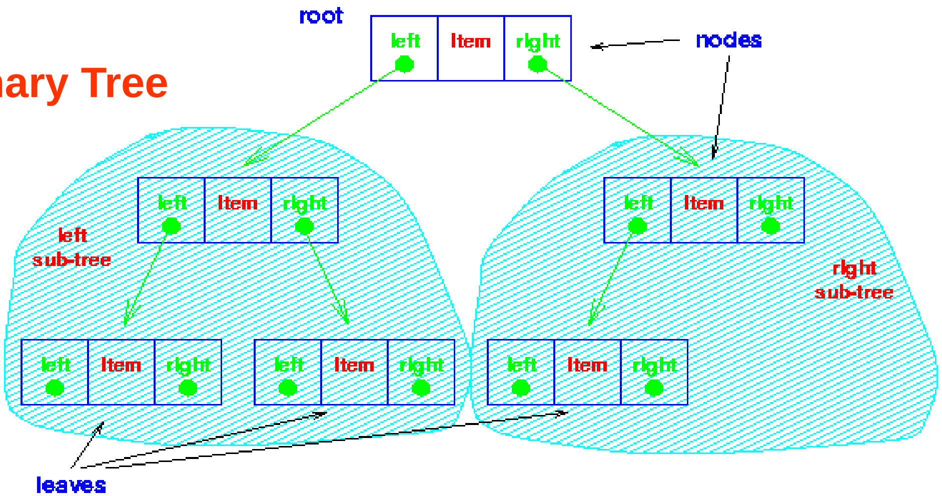


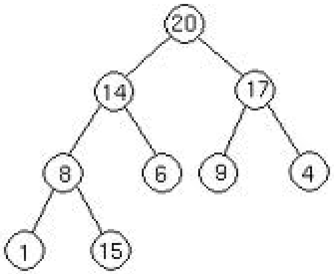
Queue



Data Structures Outlook

Binary Tree

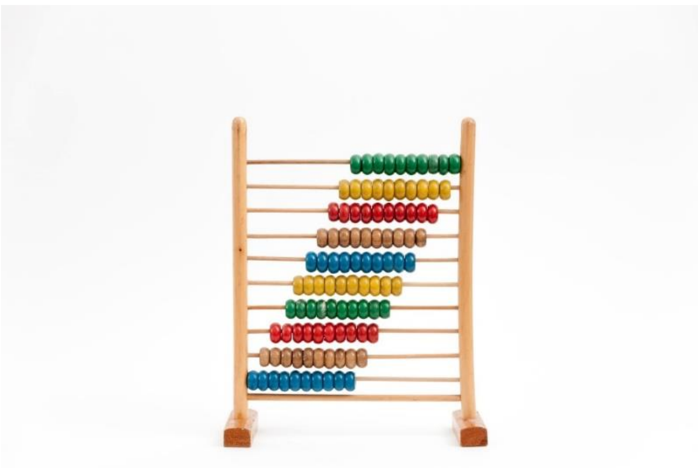




Heap

AVL Tree and others …

Algorithm Techniques – Brute Force

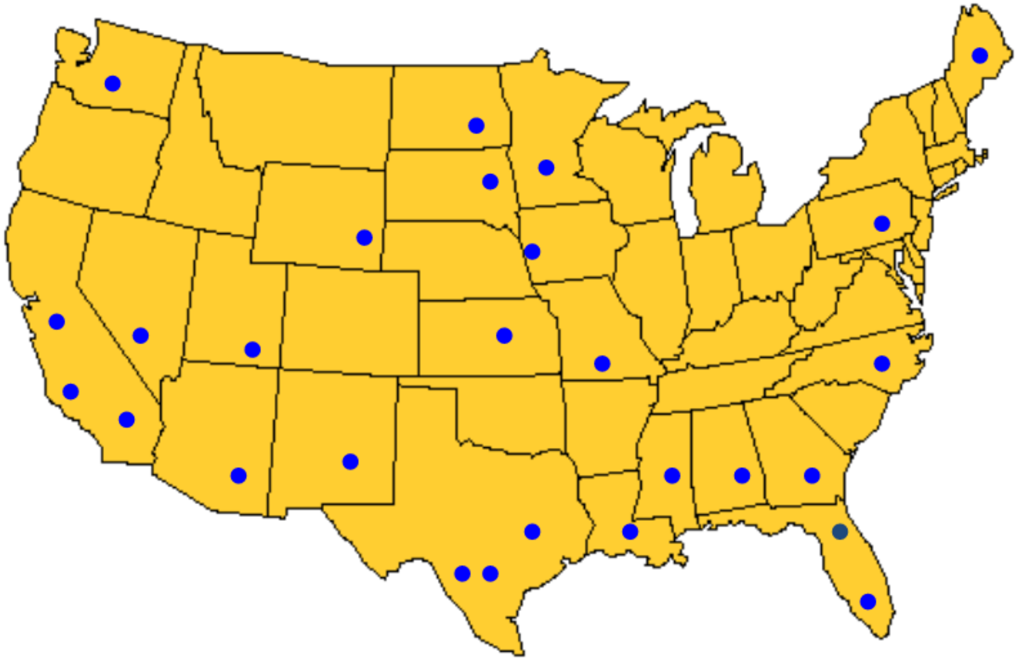


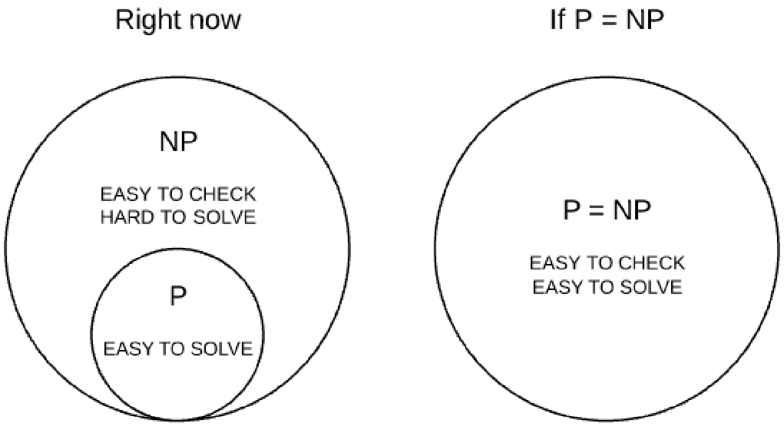
Brute Force Algorithms are exactly what they sound like – straightforward methods of solving a problem that rely on sheer computing power and trying every possibility rather than advanced techniques to improve efficiency.

Algorithm Techniques Outlook

- Brute Force
- Greedy Method
  - Knapsack
  - MST
  - Dijkstra's
- Divide & Conquer
  - Merge Sort
  - Quick Sort
  - Integer Multiplication Problem
- Dynamic Programming
  - Matrix Chain Multiplication
  - Floyd – Warshall's
- …
- …

P, NP, NP-Complete, NP Hard





Progress Check!

1) Introduction to DSECLZG519

- *Course handout*
- *Books & Evaluation components*
- *How to make the most out of this course ?*

2) Motivation & Synergies between Data Science & DSAD

3) Introduction to Algorithms

- Notion of an algorithm
- Properties of an algorithm
- Phases of program development
- Why do we have to analyze them ?

4) Analysis of algorithm efficiency

- Analysis Framework
- Pseudocode & Counting primitives
- Order of Growth
- Asymptotic Notations

5) Q&A!

Who’s the champion?

Which Algorithm is better ?

Who is topper in DSAD?

Algorithm 1:

Sort *A* into decreasing order

Output *A*[1].

Algorithm 2:

```
int i;
int m = A[1];
for (i = 2; i <= n; i ++)
          A[i] > m)
          m = A[i];
return m;
```

Which is better?

What is good algorithm?

- Resources Used
  - Running time (Lesser the running time, better the algorithm)
  - Space used (Lesser the space used, better the algorithm)

- Resource Usage
  - Measured proportional to (input) size

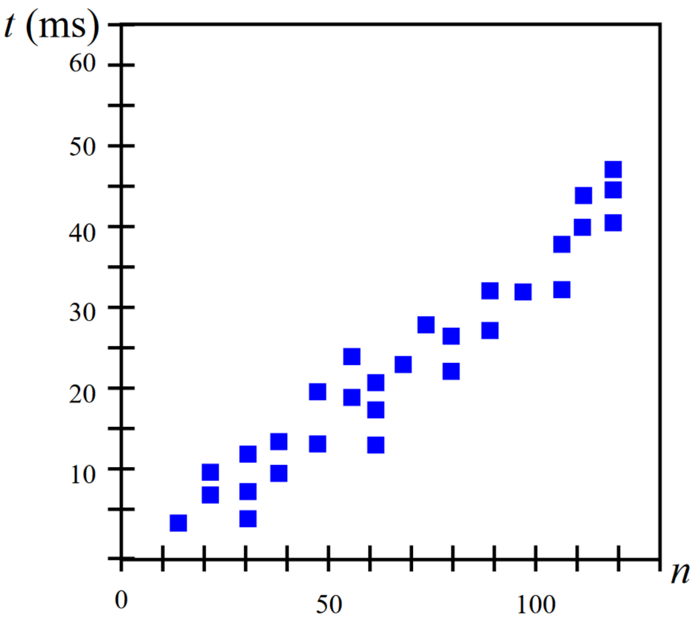
Analysis Framework

- ✓ "Better" = more efficient
- ✓ **Time**
- ✓ **Space**

How should we measure the running time of an algorithm?

Experimental Study

- Write a program that implements the algorithm
- Run the program with data sets of varying size and composition.
- Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time.
- The result maybe something similar to this :



Beyond Experimental Studies

Experimental studies have several limitations:

- It is necessary to implement and test the algorithm in order to determine its running time.
- Experiments can be done only on a limited set of inputs, and may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments should be used.
- Analytical Model to analyze algorithm - We will now develop a general methodology for analyzing the running time of algorithms that :
  - Uses a high-level description of the algorithm instead of testing one of its implementations.
  - Takes into account all possible inputs.
  - Allows one to evaluate the efficiency of any algorithm in a way that is independent from the hardware and software environment.

How to Analyze time complexity ?

Running time depends on:

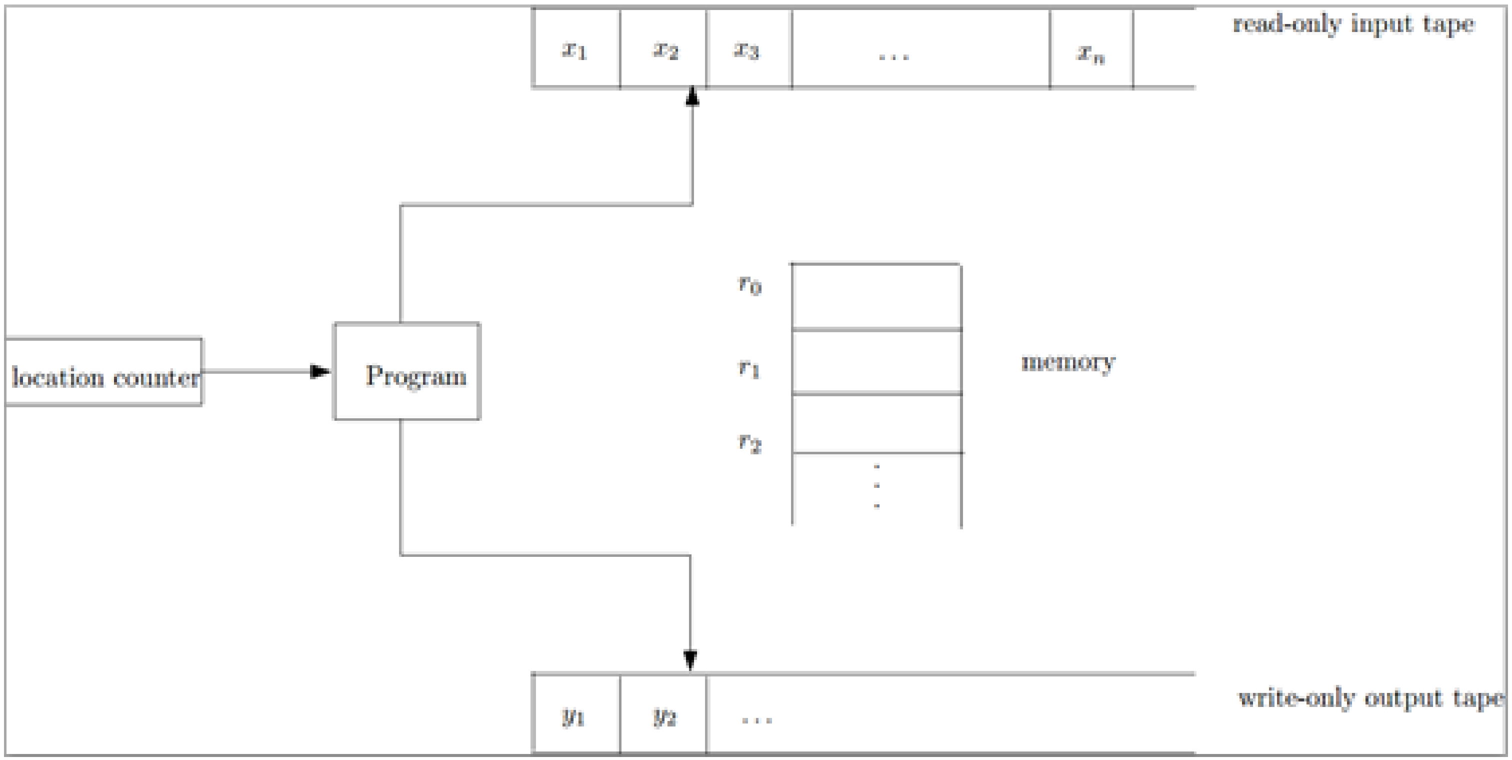
- Single vs multi processor
- Read/write speed to memory
- 32 bit or 64 bit architecture
- Input given to algorithm

A function which defines rate of growth of time w.r.t input!

Model machine: Random Access Machine model

- Algorithms can be measured in a machine-independent way using the **Random Access Machine (RAM) model.**
- This model assumes a single processor.
- In the RAM model, instructions are executed one after the other, with no concurrent operations.
- This model of computation is an abstraction that allows us to compare algorithms on the basis of performance.
- The assumptions made in the RAM model to accomplish this are:
  - Each simple operation takes 1 time step.
  - Loops and subroutines are not simple operations.
  - Each memory access takes one time step, and there is no shortage of memory (we assume we have unbounded memory).

Model machine: Random Access Machine model



- ◆Time complexity (running time) = number of instructions executed
- ◆Space complexity = the number of memory cells accessed

Pseudo-code

A mixture of natural language and high level programming concepts that describes the main ideas behind a generic implementation of a data structure and algorithms.

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

0	1	2	3	4
10	20	30	5	7

Example: find max element of an array

Algorithm ***arrayMax***(***A***, ***n***)
  Input array ***A*** of ***n*** integers
  Output maximum element of ***A***

  currentMax ← ***A***[0]
  for ***i*** ← 1 **to** ***n*** – 1 **do**
    if *A*[*i*] > ***currentMax*** **then**
      currentMax ← ***A***[***i***]
  return ***currentMax***

Pseudo-code (Some Guidelines)

- Is structured than usual text but less formal than a programming language.
- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method declaration
  - **Algorithm** ***method*** (***arg*** [, ***arg***…])
- Expressions

  Use standard mathematical symbols to describe numeric and Boolean expressions.

  $\leftarrow$ Assignment (like = in C / Java / Python)

  $=$ Equality testing (like == in C / Java / Python)

  n^2 Superscripts and other mathematical formatting allowed

Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely *independent* from the programming language
- Assumed to take a constant amount of time in the RAM model

Examples:

- Evaluating an expression
- Assigning a value to a variable
- Indexing into an array
- Calling a method
- Returning from a method

Analyzing Pseudo-code (By Counting) / Counting Primitives

1. For each line of pseudocode, count the number of primitive operations in it.
2. Pay attention to the word "*primitive*" here; sorting an array is not a primitive operation.
3. Multiply this count with the number of times this line is executed.
4. Sum up over all lines.

Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

Printing each element of an array

```
i=0;
while (i<n)
{
        print a[i];
        i++;
}
```

Counting Primitive Operations

Printing each element of an array

```
i=0;
while (i<n)
{
        print a[i];
        i++;
}
```

→ One Initialization of i
→ n+1 comparisons
→ n array indexing operations
→ n invocations of print
→ n increments of I

So, we write T(n)=1+n+1+n+n+n
T(n)=4n+2

Counting Primitives

We can also use a tabular way for counting primitives as below:

Algorithm ArraySum(A, n)	#Operations	Remarks
(1) Sum = A [0]	2	Indexing , Assignment
(2) i = 1	1	Assignment
(3) while (i<n)	n	Comparison
(a) Sum = Sum + A[i]	3 (n-1)	(n-1) times indexing, addition and assignment
(b) i = i + 1	2 (n-1)	(n-1) times addition and assignment
(4) return Sum	1	1 times returning

Add the Operations Column = $2 + 1 + n + 3n - 3 + 2n - 2 + 1$

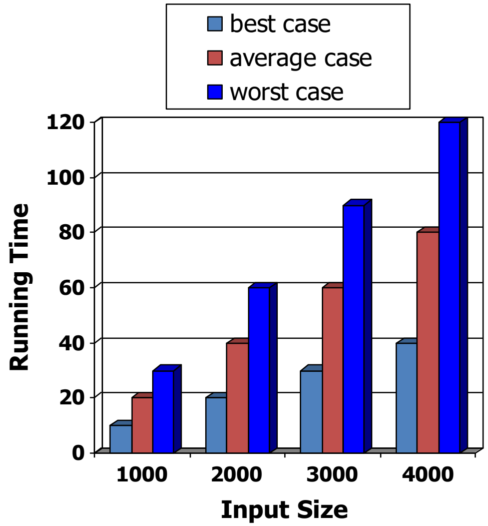
$$= 6n - 1$$

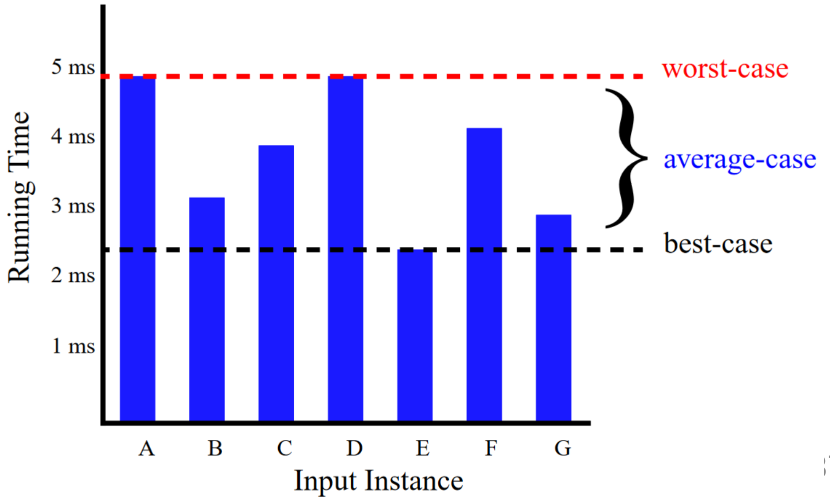
Progress Check!

1) *Introduction to DSECLZG519*
   - *Course handout*
   - *Books & Evaluation components*
   - *How to make the most out of this course ?*
2) *Motivation & Synergies between Data Science & DSAD*
3) *Introduction to Algorithms*
   - *Notion of an algorithm*
   - *Properties of an algorithm*
   - *Phases of program development*
   - *Why do we have to analyze them ?*
4) *Analysis of algorithm efficiency*
   - *Analysis Framework*
   - *Pseudocode & Counting primitives*
   - Order of Growth
   - Asymptotic Notations
5) Q&A!

Running Time

- What is best, average, worst case running of a problem?
- An algorithm may run faster on certain data sets than on others.
- Average case time is often difficult to determine – why?
- We focus on the worst case running time.
  - Easier to analyze and best to bet
  - Crucial to applications such as games, finance and robotics
  - Performing well in worst case means it would perform well in normal input too!





Example

To find the sum of two numbers

Sum(a,b)

{

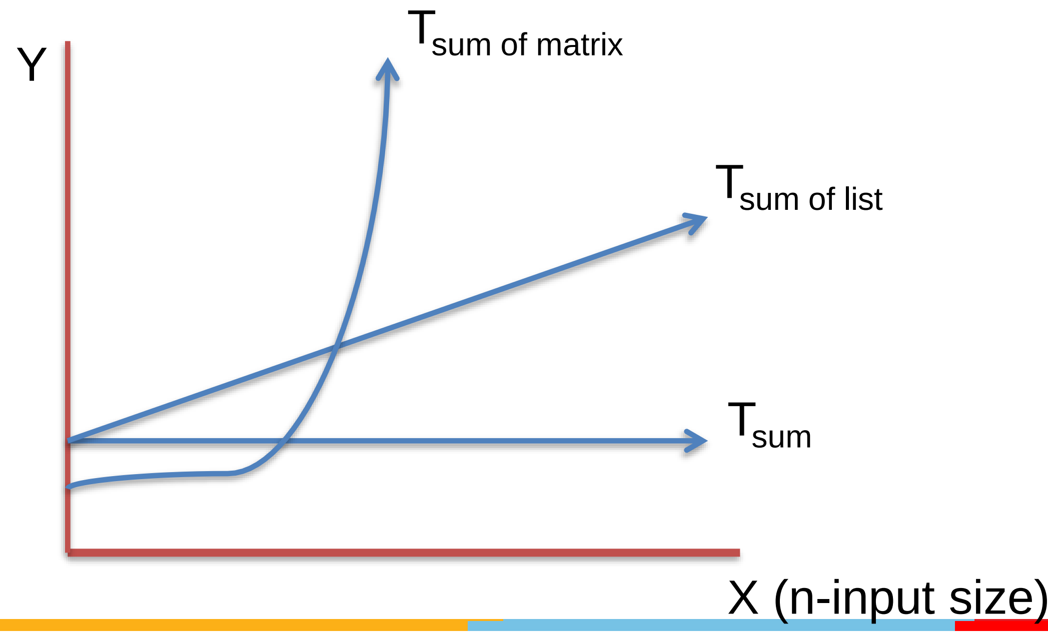
Return a+b

}

T_{sum} = 2 {it’s a constant time algorithm)

Example (Cont..,)

1) $T_{sum} = K \rightarrow$ all the function of form some constant O(1)
2) $T_{sum\ of\ list} = cn + c' \rightarrow$ all linear function O(n) [linear function]
3) $T_{sum\ of\ matrix} = an^2 + bn + c \rightarrow O(n^2)$ [set of all the function]



Time complexity Analysis- Some General Rules.

We analyze time complexity

a) Very large input-size

b) Worst case scenario

$T(n) = n^3 + 3n^2 + 4n + 2$

$\approx n^3 \ (n \rightarrow \alpha)$

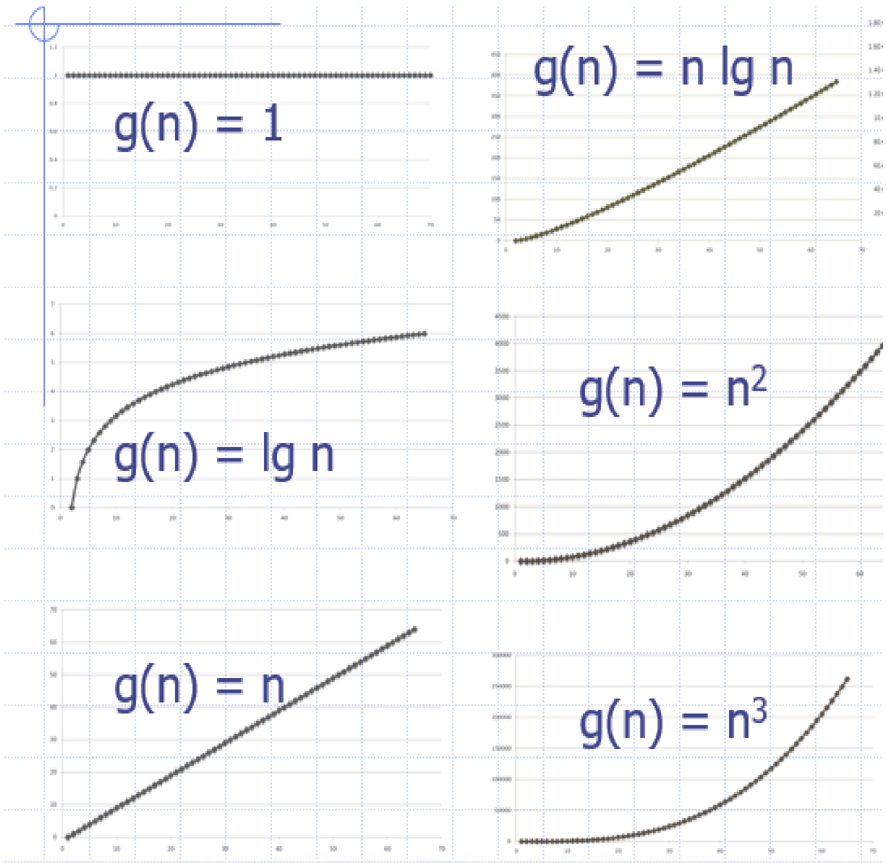
$c.n^3 \quad O(n^3)$

Rule: a) drop lower order terms

b) drop constant multiplier

$T(n) = 17n^4 + 3n^3 + 4n + 8 = O(n^4)$

$T(n) = 16n + \log n = O(n)$



Time complexity Analysis- Some General Rules.

Eliminate low order terms

$4n + 5 \Rightarrow 4n$

$0.5\, n \log n - 2n + 7 \Rightarrow 0.5\, n \log n$

$2^n + n^3 + 3n \Rightarrow 2^n$

Eliminate constant coefficients

$4n \Rightarrow n$

$0.5\, n \log n \Rightarrow n \log n$

Order of Growth

- We expect our algorithms to work faster for all values of N. Some algorithms execute faster for smaller values of N. But, as the value of N increases, they tend to be very slow.
- So, the behavior of some algorithms changes with increase in value of N.
- This change in behavior of the algorithm and algorithm's efficiency can be analyzed by considering the highest value of N.
- Order of growth :

As N increases order of growth increases

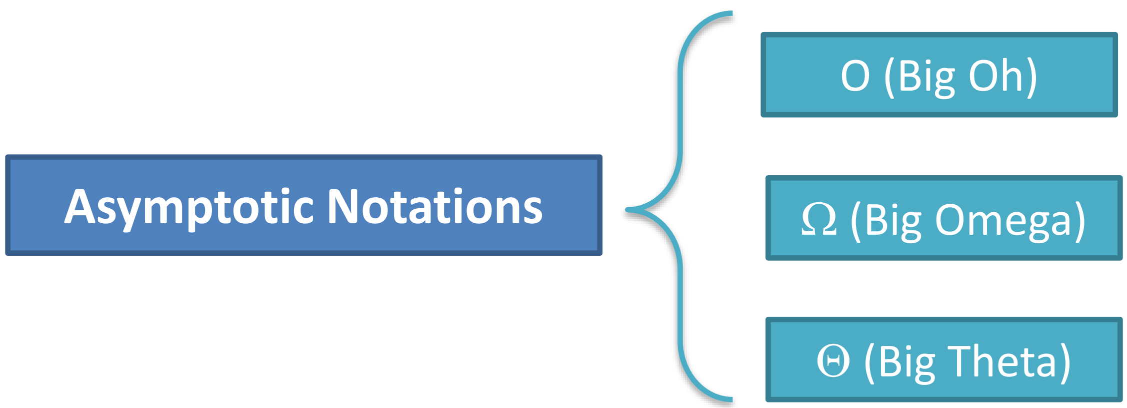
N	log N	N	N log N	N^2	N^3	2^N	N!
1	0	1	0	1	1	2	1
2	1	2	2	4	8	4	2
4	2	4	8	16	64	16	24
8	3	8	24	64	512	256	40320
16	4	16	64	256	4096	65536	high
32	5	32	160	1024	32768	4294967296	very high

$$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$$

As n Increases, order of Growth increases …

Asymptotic Notations

- Asymptotic Notations are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases.
- This is also known as an algorithm's growth rate.
- Asymptotic notations are the notations used to express the order of growth of an algorithm and it can be used to compare two algorithms with respect to their efficiency.

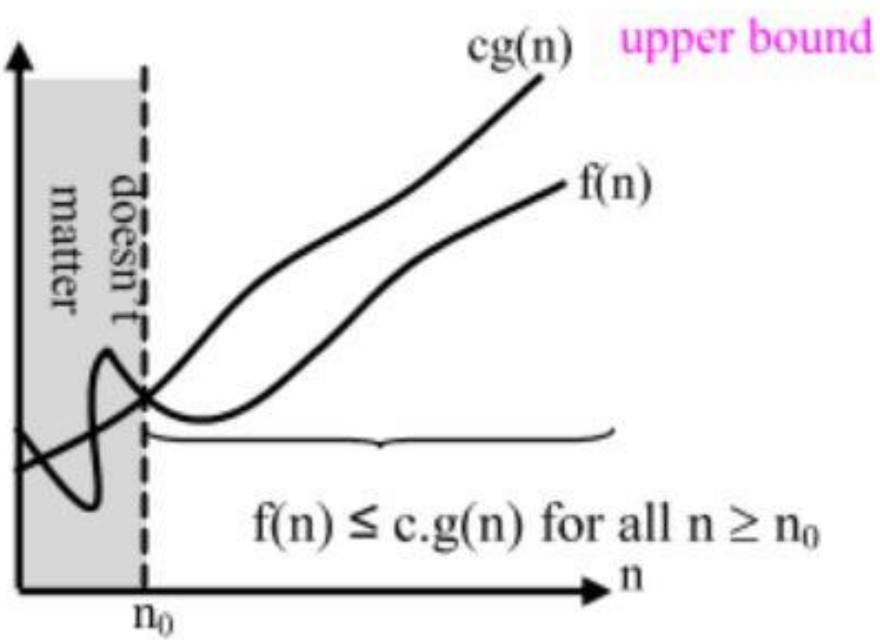


Asymptotic Notation

- O notation: asymptotic "less than": **Upper Bound**
  - $f(n)=O(g(n))$ implies: $f(n)$ "$\leq$" $g(n)$
- Ω notation: asymptotic "greater than": **Lower Bound**
  - $f(n)=\Omega(g(n))$ implies: $f(n)$ "$\geq$" $g(n)$
- Θ notation: asymptotic "equality": **Average Bound**
  - $f(n)=\Theta(g(n))$ implies: $f(n)$ "$=$" $g(n)$

Big - Oh Notation

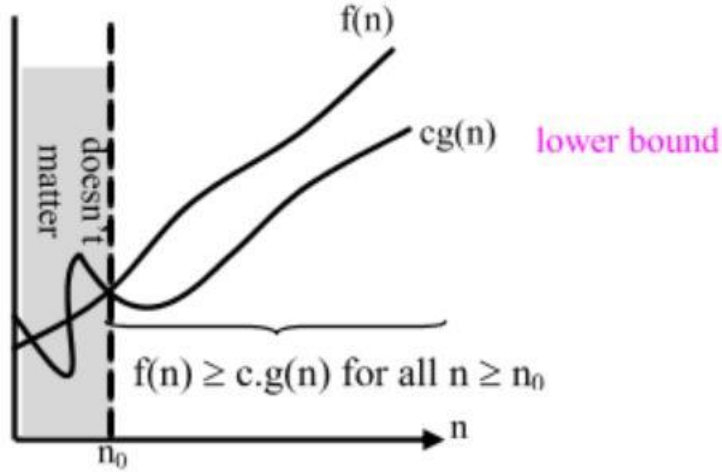
- Big – Oh is the formal method of expressing the upper bound of an algorithm's running time.
- It is a **measure of the longest amount of time** it could possibly take for the algorithm to complete.



Big Oh denotes the **worst** case complexity !!

Relatives of Big-Oh – Big Omega ($\Omega(n)$)

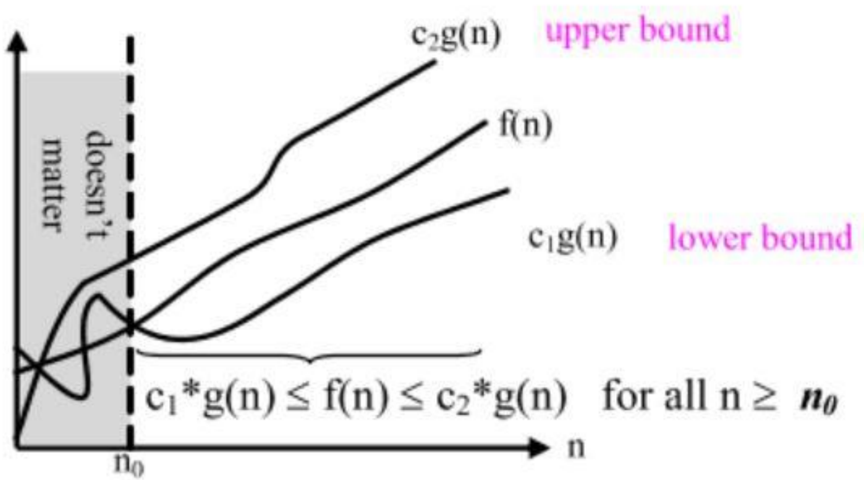
- Big – Omega is the formal method of expressing the lower bound of an algorithm's running time.
- In general, the lower bound implies that below this time the algorithm cannot perform better.
- It is a **measure of the least amount of time** it could possibly take for the algorithm to complete.



Big Omega denotes the **best** case complexity !!

Relatives of Big-Oh – Big Theta (Θ(n))

- Big – theta is the formal method of expressing both the lower bound & upper bound of an algorithm's running time.
- The upper bound on f(n) indicates that function f(n) will not consume more than the specified time c_2*g(n) & the lower bound on f(n) indicates that function f(n) in the best case will consume atleast the specified time c_1*g(n)



Big Theta denotes the **average** case complexity !!

Summary for CS #1

1) Introduction to DSECLZG519
   - Courseware
   - Books & Evaluation components
   - How to make the most out of this course ?
2) Motivation & Synergies between Data Science & DSAD
3) Introduction to Algorithms
   - Notion of an algorithm
   - Properties of an algorithm
   - Phases of program development
   - Outline of data structures & Algorithm design strategies
   - Why do we have to analyze them ?
4) Analysis of algorithm efficiency
   - Analysis Framework
   - Pseudocode & Counting primitives
   - Asymptotic Notations
5) Exercises, Q&A

Exercises

Write the pseudocode for below and count the primitives:

- Minimum element in an array
- Maximum of 3 numbers
- Write code to count how many elements are even in the given list.
- Linear Search – Finding if an element is present in an array.

State if the below are true/False , Fill in the blanks

- Is $T(n) = 9n^4 + 876n = O(n^4)$?
- Is $T(n) = 9n^4 + 876n = O(n^3)$?
- Is $T(n) = 9n^4 + 876n = O(n^{27})$?

- $T(n) = n^2 + 100n = O(?)$
- $T(n) = 3n + 32n^3 + 767249999n^2 = O(?)$

We will explore more on Algorithm analysis in the next class …

Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in