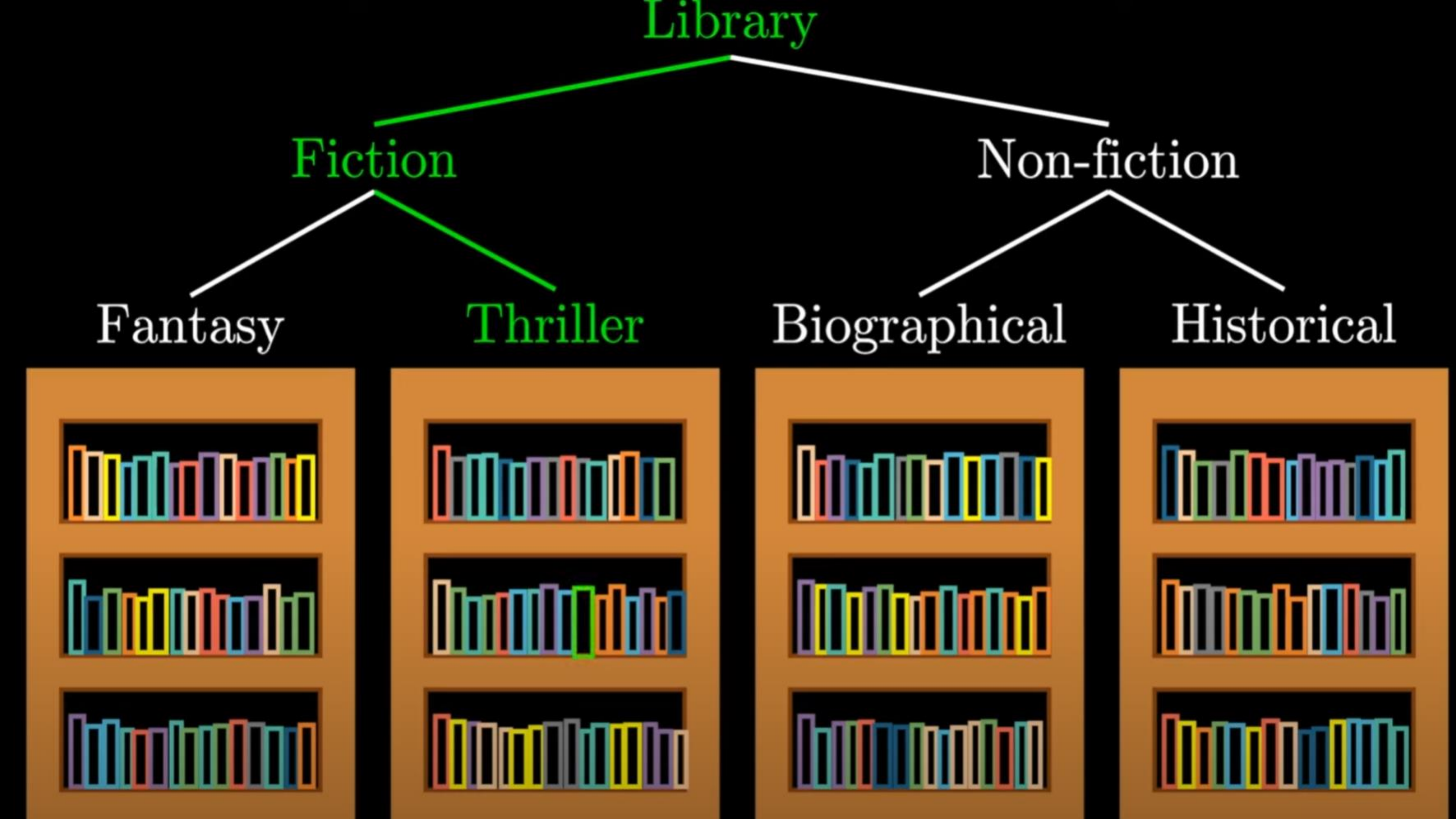

Library
Fiction
Non-fiction
Fantasy
Thriller
Biographical
Historical

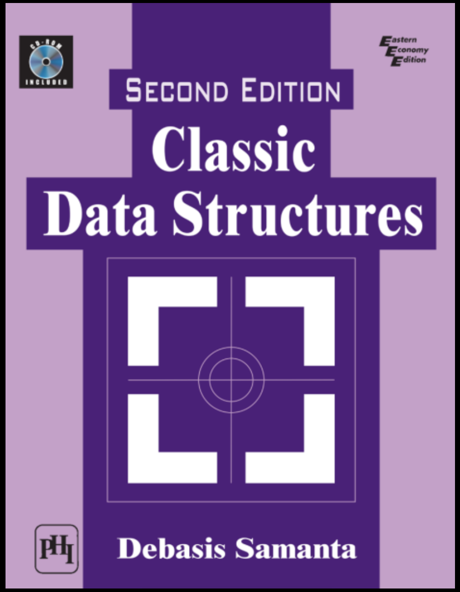
Data Structures and Algorithms

Academic Year 2023-24

Course name	: Data Structures and Algorithms
Quarter	: Q1
Credits	: 2
Instructor	: Sudipta Roy
Instructor Email	: sudipta1.roy@jioinstitute.edu.in
Teaching assistant	: Snehashis (snehashis1.c@jioinstitute.edu.in)

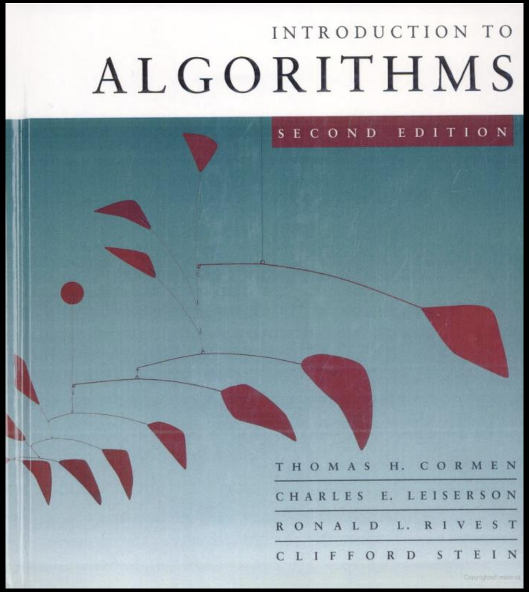
Textbook:

- Debasis Samanta, Classic Data Structures, Prentice Hall India Learning Private Limited, Edition 2nd
- T Cormen, C Leiserson, R Rivest, C Stein, Introduction to Algorithms, 3Ed. MIT Press



Reference book:

- Narasimha Karumanchi, Data Structures and Algorithms Made Easy: Data Structures And Algorithmic Puzzles, Made Easy, 1 January 2016.
- Anany Levitin, Introduction to the Design and Analysis of Algorithms, Pearson Education, EDs 3rd, 26 February 2017
- Kleinberg, Algorithm Design, Pearson Education India, Eds 1st, 1 January 2013
- Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, "Data Structures and Algorithms in Python" WILEY



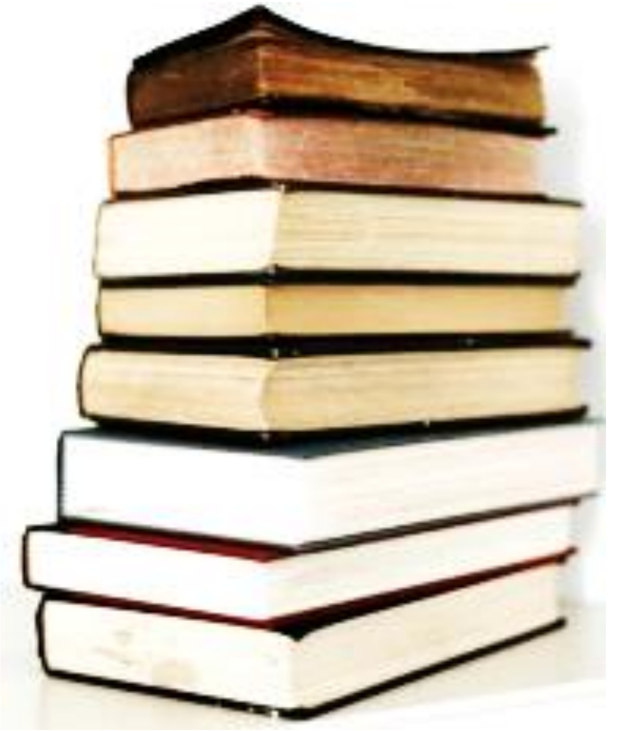
Course Structure and Evaluation:

The grade distribution for this course is as follows:

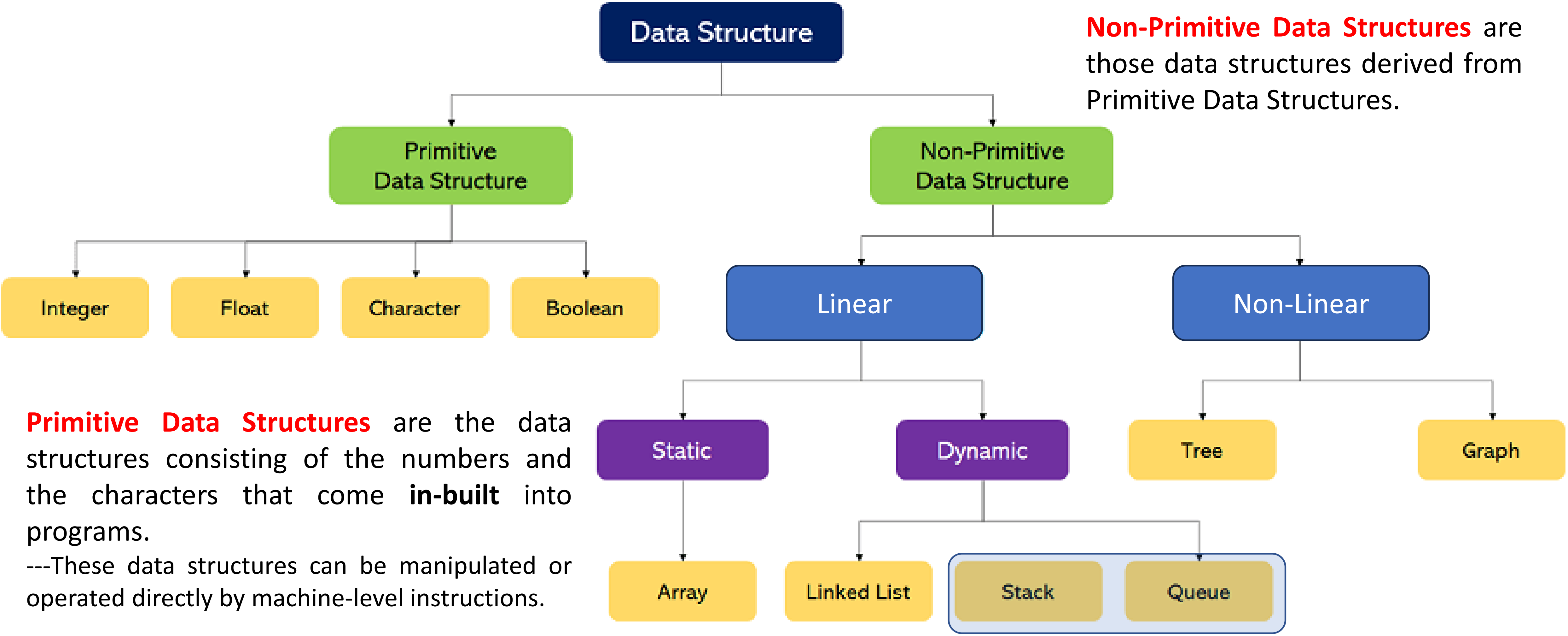
- **Class participation/attendance: 10**
- **Quizzes (top n-1): 10**
- **Lab exercises (20 marks) and home assignments (10): 30 (**Viva and/ or Code
- writing examination for programs will be taken and considered as a performance for LAB. Home assignment marks will be average of n assignments.**)**
- **Final examination: 50 (**Different types of questions will be included such as MCQ, problem solving, long answers and writing fragments of code etc.**)**

Data structure and Algorithm

- **Data**
  - Data is a collection of facts and figures or a set of values or values of a specific format that refers to a single set of item values.
- **Data structure**
  - A data structure is an organization, management, and storage format of data that is usually chosen for efficient access to data. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.
- **Algorithm**
  - An algorithm is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing.



Classification of Data Structures



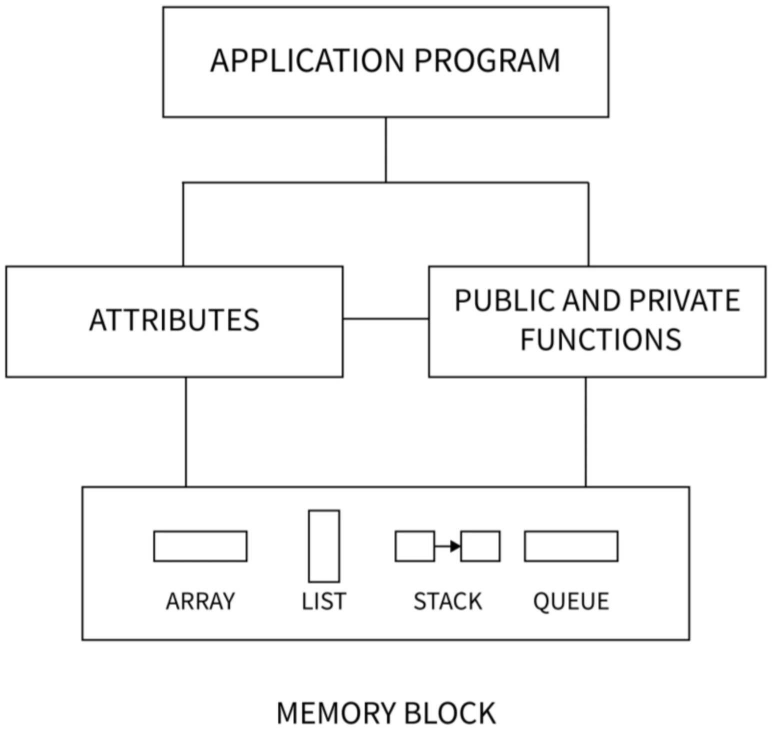
Non-Primitive Data Structures are those data structures derived from Primitive Data Structures.

Primitive Data Structures are the data structures consisting of the numbers and the characters that come **in-built** into programs.

---These data structures can be manipulated or operated directly by machine-level instructions.

Abstract Data type (ADT)

- Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view.
- **Features of ADT:**
  - **Abstraction:** The user does not need to know the implementation of the data structure only essentials are provided.
  - **Better Conceptualization:** ADT gives us a better conceptualization of the real world.
  - **Robust:** The program is robust and has the ability to catch errors.



Array

Polynomial and Sparse Metrics

Polynomials using Arrays

Arrays:

- Basic data structure
- May store any type of elements

Polynomials: defined by a list of **coefficients** and **exponents-** *degree* of polynomial = **the largest exponent in a polynomial**

$$p(x) = a_1 x^{e_1} + \ldots + a_n x^{e_n}$$

$2x^3 - 6x^2 + 2x - 1$

1	**2**	**-6**	**2**					
0	1		……………					9

Python

```python
# 2x3 - 6x2 + 2x - 1 for x = 3
poly = [2, -6, 2, -1]
x = 3
n = len(poly)
# Declaring the result
result = 0
# Running a for loop to traverse through the
list
for i in range(n):

    # Declaring the variable Sum
    Sum = poly[i]

    # Running a for loop to multiply x (n-i-1)
    # times to the current coefficient
    for j in range(n - i - 1):
        Sum = Sum * x

    # Adding the sum to the result
    result = result + Sum

# Printing the result
print(result)
```

User input

```python
a=[]
n=int(input("Number of elements in array:"))
for i in range(0,n):
l=int(input())
a.append(l)
print(a)
```

What is the problem ?

$2x^{30} - 6x^2 + 2x - 1$

Polynomial

- Use one global array to store all polynomials

$A(X)=2X^{1000}+1$
$B(X)=X^4+10X^3+3X^2+1$

This also helps in addition

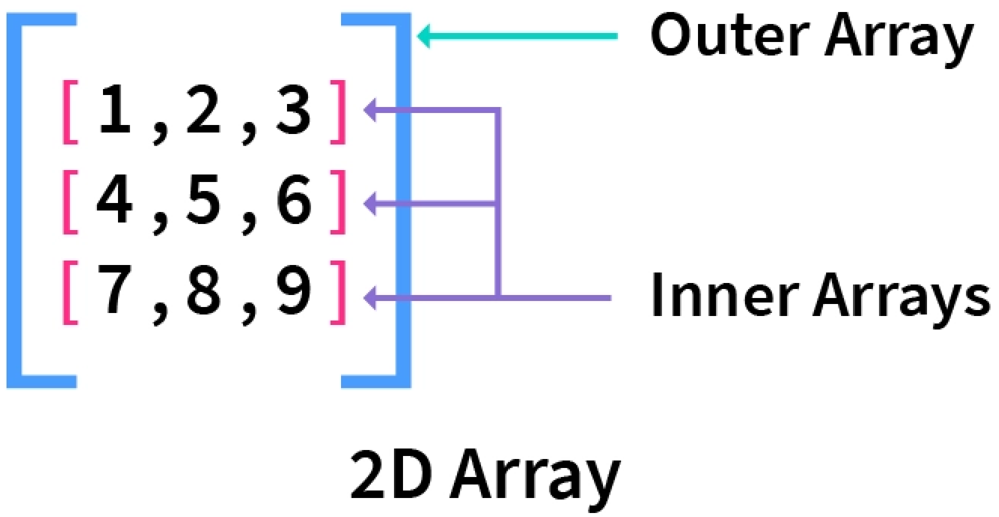
	Start A	*finish A*	*start B*			*finish B*	
coef	2	1	1	10	3	1	
exp	1000	0	4	3	2	0	
	0	1	2	3	4	5	6

$\underbrace{\text{0 … 1}}_{A(X)=2X^{1000}+1}$ $\underbrace{\text{2 … 5}}_{B(X)=X^4+10X^3+3X^2+1}$

2D Array
or
two array

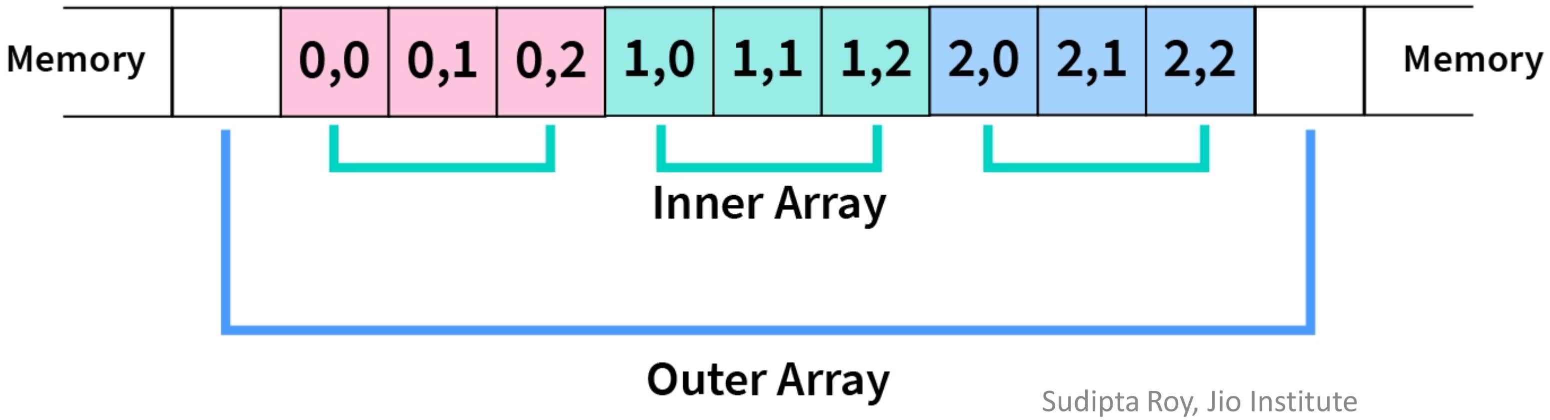
You can also use Structure for this

2D Array matrix



row,col

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2



2D Array

```c
int b[2][3];
int i,j,num;
printf("Enter elements into 2-D array: ");
for(i=0;i<2;i++)
{
        for(j=0;j<3;j++)
        {
                scanf("%d" , &b[i][j]);
        }
}
```

```python
rows, cols = (5, 5)

arr = [[0 for i in range(cols)] for j in range(rows)]

# again in this new array lets change
# the first element of the first row
# to 1 and print the array
arr[0][0] = 1

for row in arr:
        print(row)
```

Sparse Matrices

$$
\begin{array}{c}
\\ \text{row0} \\ \text{row1} \\ \text{row2} \\ \text{row3} \\ \text{row4} \\ \text{row5}
\end{array}
\begin{array}{c}
\begin{array}{cccccc} \text{col1} & \text{col2} & \text{col3} & \text{col4} & \text{col5} & \text{col6} \end{array} \\
\begin{bmatrix}
15 & 0 & 0 & 22 & 0 & -15 \\
0 & 11 & 3 & 0 & 0 & 0 \\
0 & 0 & 0 & -6 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
91 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 28 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

- An n x n matrix may be stored as an n x n array.
- This takes $O(n^2)$ space.
- The example structured sparse matrices may be mapped into a 1D array so that a mapping function can be used to locate an element quickly; the space required by the 1D array is less than that required by an n x n array.

Sparse Matrices

Count -- **row and column** and **non-zero element** in the matrix

$$\begin{array}{c} \\ \text{row 1} \\ \text{row 2} \\ \text{row 3} \\ \text{row 4} \\ \text{row 5} \end{array} \begin{array}{c} \begin{array}{ccc} \text{col 1} & \text{col 2} & \text{col 3} \end{array} \\ \begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{bmatrix} \end{array}$$

5*3

15/15

Not sparse matrix

$$\begin{array}{c} \\ \text{row0} \\ \text{row1} \\ \text{row2} \\ \text{row3} \\ \text{row4} \\ \text{row5} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{col1} & \text{col2} & \text{col3} & \text{col4} & \text{col5} & \text{col6} \end{array} \\ \begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

6*6

8/36

data structure?

sparse matrix

	row	*col*	*value*
a[0]	***6***	***6***	***8***
[1]	*0*	*0*	*15*
[2]	*0*	*3*	*22*
[3]	*0*	*5*	*-15*
[4]	*1*	*1*	*11*
[5]	*1*	*2*	*3*
[6]	*2*	*3*	*-6*
[7]	*4*	*0*	*91*
[8]	*5*	*2*	*28*

Sparse Matrix Operations

- Transpose of a sparse matrix.
- What is the transpose of a matrix?

Row<--> column and then sort based on 1st column

	row	col	value
a[0]	**6**	**6**	**8**
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

$\xrightarrow{\text{transpose}}$

	row	col	value
b[0]	**6**	**6**	**8**
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

Sparse matrix in C

```
 // number of columns in compactMatrix (counter) must be equal to
//number of non - zero elements in sparseMatrix

    int compactMatrix[3][counter];
     // Making of new matrix
    int k = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (sparseMatrix[i][j] != 0)
            {
                compactMatrix[0][k] = i;
                compactMatrix[1][k] = j;
                compactMatrix[2][k] = sparseMatrix[i][j];
                k++;
            }

    for (int i=0; i<3; i++)
    {
        for (int j=0; j< counter; j++)
            printf("%d ", compactMatrix[i][j]);
         printf("\n");
    }
    return 0;
}
```

```
#include <stdio.h>
 void main ()
{
    static int array[10][10];
    int i, j, m, n;
    int counter = 0;
     printf("Enter the order of the matix \n");
    printf("Enter the number of rows in array: ");
    scanf("%d", &m);
    printf("Enter the number of columns in array: ");
    scanf("%d", &n);

    printf("Enter the co-efficients of the matix: \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &array[i][j]);
            if (array[i][j] == 0)
            {
                ++counter;
            }
        }
    }
    if (counter > ((m * n) / 2))
    {
        printf("The given matrix is sparse matrix \n");
    }
    else
        printf("The given matrix is not a sparse matrix \n");
    printf("There are %d number of zeros", counter);
}
```

```
import numpy as np
input_matrix = np.array([[16, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 5], [0, 0, 0, 0]])
print("The input matrix is:")
print(input_matrix)

sparse_matrix = []

rows, cols = input_matrix.shape
for i in range(rows):
        for j in range(cols):
                if input_matrix[i][j] != 0:
                        triplet = [i, j, input_matrix[i][j]]
                        sparse_matrix.append(triplet)
print("The sparse matrix is:")
print(sparse_matrix)
```

What wrong in this code ?

Asymptotic Notations

An analogy

Say we go through a drive-thru. We drive-in. We order. We pay. We receive our food. We drive out.
The drive-thru is the function.

```
function DriveThru(car)
 {
        order;
        pay;
        food;
        return car;
 }
```

If a drive-thru serves 1,000 cars in a day, it returns 1,000 cars. Every car goes through the same three steps: order, pay, and food.

Why do some cars get through the drive-thru faster than others?

As:

- A car can hold N amount of people.
- The time it takes a car to get through the drive-thru is dependent on how many people are in the car. Think of the car as an array of objects.

Asymptotic Complexity

- **Describes behavior of function in the limit.**
- Running time of an algorithm as a function of input size *n* **for large *n***.
- Expressed using only the **highest-order term** in the expression for the exact running time.
- Written using ***Asymptotic Notation.*** Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.
- Asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine-specific constants.

Asymptotic Notation

Θ, O, Ω, o, ω

- Defined for functions over the natural numbers.
  - **Ex:** $f(n) = \Theta(n^2)$.
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The **notations describe different rate-of-growth relations between the defining function and the defined set of functions**.

O-notation

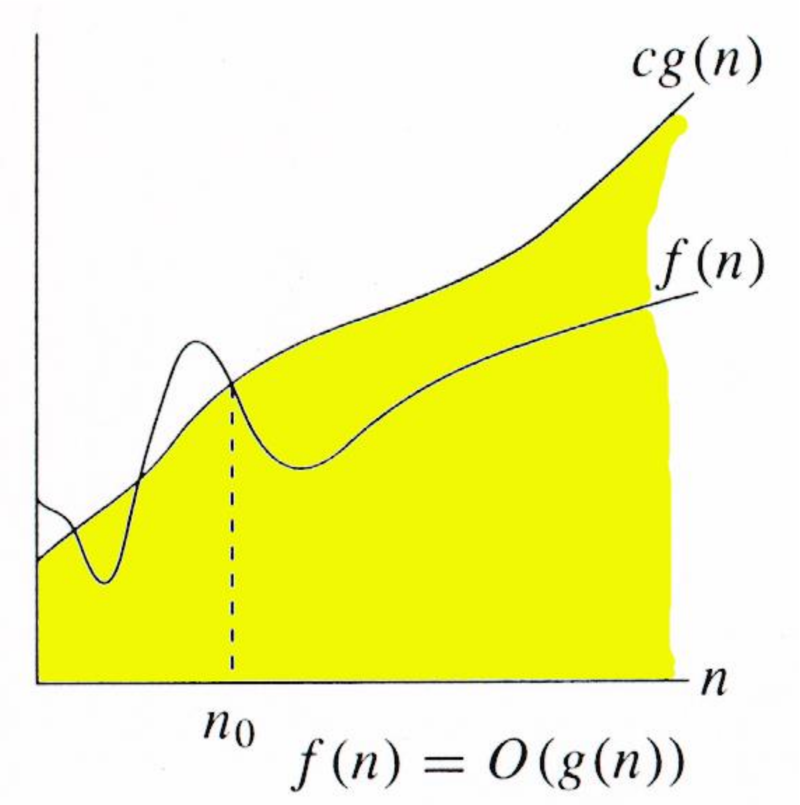
For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n)$: $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n) \}$

Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

$g(n)$ is an *asymptotic upper bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.
$\Theta(g(n)) \subset O(g(n))$.



Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.

Let's have an example

N	N!	2^N
1		
2		
3		
4		
5		

$N_0 = ?$

$C = ?$

F_n ?

G_n ?

Examples

$O(g(n)) = \{f(n)$: $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n)$ $\}$

Consider the following f(n) and g(n)...
f(n) = 3n + 2, g(n) = n
If we want to represent **f(n)** as **O(g(n))** then it must satisfy **f(n) <= C g(n)** for all values of **C > 0** and **n_0>= 1**

f(n) <= C g(n) $\Rightarrow$3n + 2 <= C n
Above condition is always TRUE for all values of **C = 4** and **n >= 2**.

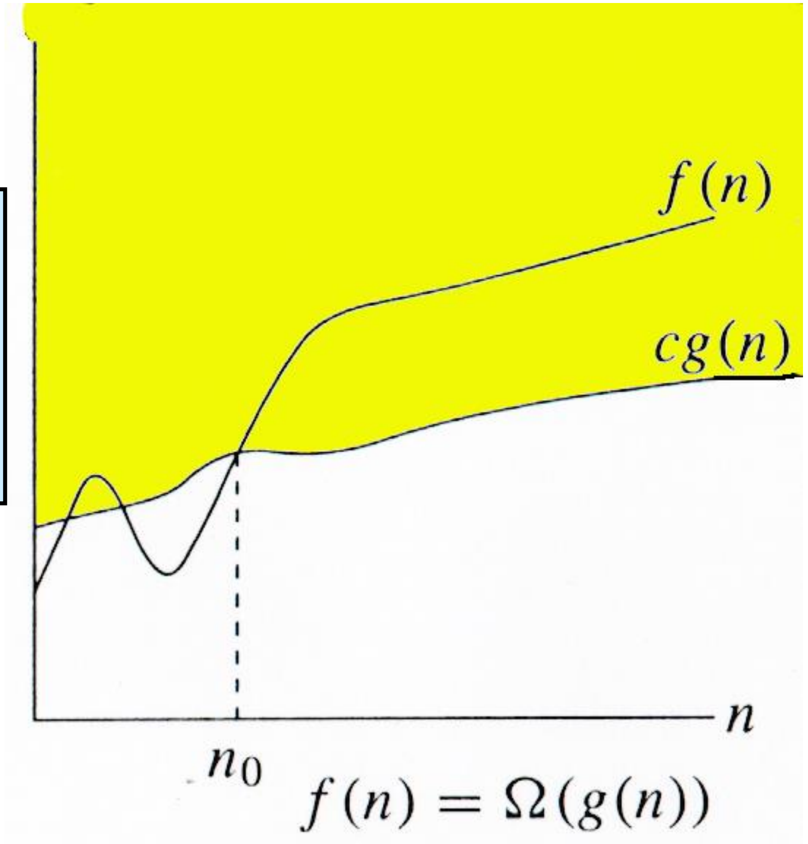
By using Big - Oh notation we can represent the time complexity as follows...
3n + 2 = O(n)

- Any linear *function an* + *b* is in $O(n^2)$/ O(n).
- Show that $3n^3=O(n^4)$ for appropriate *c* and n_0.

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ positive constants c and $n_0,$ such that $\forall n \geq n_0,$
we have $0 \leq cg(n) \leq f(n)\}$



Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm.

Example

$\Omega(g(n)) = \{f(n) :\ \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$

Consider the following f(n) and g(n)...
f(n) = 3n + 2, g(n) = n
If we want to represent **f(n)** as **Ω(g(n))** then it must satisfy **f(n) >= C g(n)** for all values of **C > 0** and **n_0>= 1**

f(n) >= C g(n) ⇒3n + 2 >= C n
Above condition is always TRUE for all values of **C = 1** and **n >= 1**.

By using Big - Omega notation we can represent the time complexity as follows...
3n + 2 = Ω(n)

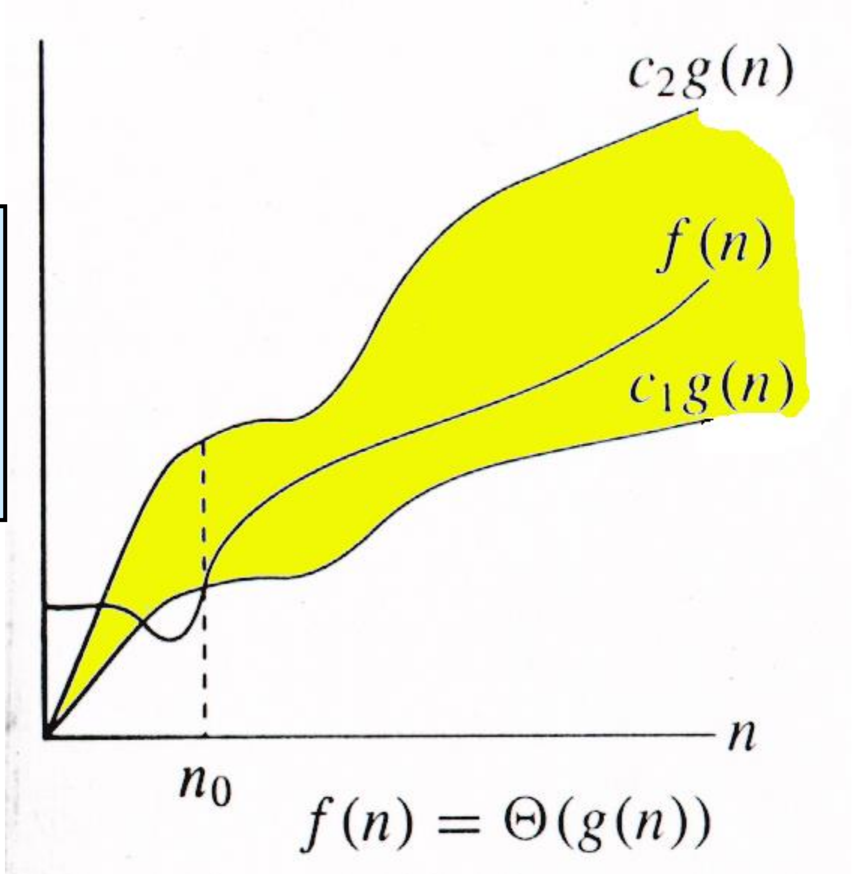
- $\sqrt{n} = \Omega(\lg n)$. Choose c and n_0.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n) :$ $\exists$ **positive constants c_1, c_2, and n_0, such that** $\forall n \geq n_0,$

we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\}$



Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.
I'll accept either…

$f(n)$ and $g(n)$ are nonnegative, for large n.

Example

$\Theta(g(n)) = \{ f(n) :$ $\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Consider the following f(n) and g(n)...
f(n) = 3n + 2, g(n) = n
If we want to represent f(n) as Θ(g(n)) then it must satisfy C_1 g(n) <= f(n) <= C_2 g(n) for all values of $C_1 > 0$, $C_2 > 0$ and $n_0 >= 1$

C_1 g(n) <= f(n) <= C_2 g(n) $\Rightarrow C_1$ n <= 3n + 2 <= C_2 n
Above condition is always TRUE for all values of **C_1 = 1, C_2 = 4** and **n >= 2**.

By using Big - Theta notation we can represent the time compexity as follows...
3n + 2 = Θ(n)

Example

$\Theta(g(n))$ = { $f(n)$: $\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$}

- $10n^2 - 3n = \Theta(n^2)$
  - What constants for n_0, c_1, and c_2 will work?
  - Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.

Homework

Example

$\Theta(g(n))$ = {$f(n)$: $\exists$ positive constants c_1, c_2, and n_0,
such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$}

- Is $3n^3 \in \Theta(n^4)$??
- How about $2^{2n} \in \Theta(2^n)$??

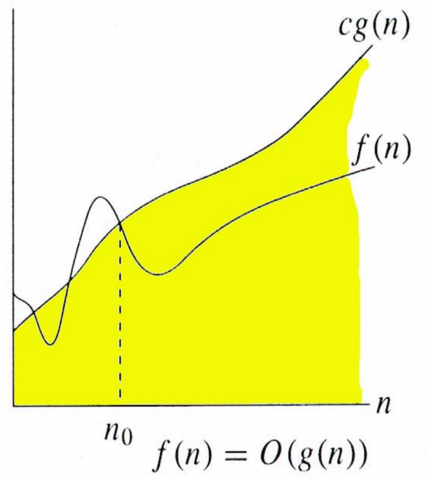
Relations Between O, Θ, Ω

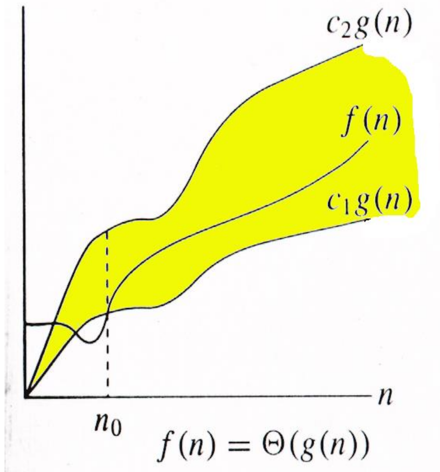
Theorem : For any two functions $g(n)$ and $f(n)$,

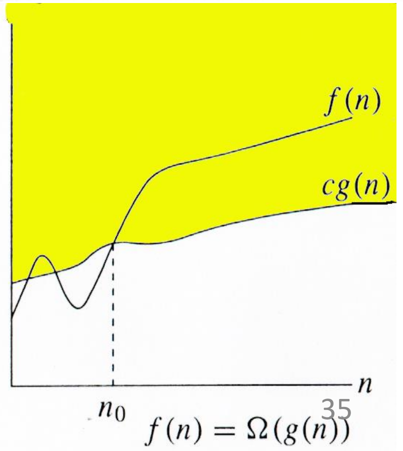
$f(n) = \Theta(g(n))$ iff

$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- I.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.







Asymptotic Notation in Equations

- Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- For example,

  $4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$

  $= 4n^3 + \Theta(n^2) = \Theta(n^3)$. **How to interpret?**
- In equations, $\Theta(f(n))$ always stands for an ***anonymous function*** $g(n) \in \Theta(f(n))$
  - In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

o-notation (Little o)

For a given function $g(n)$, the set little-o:

$o(g(n)) = \{f(n)$: $\forall$ $c > 0$, $\exists$ $n_0 > 0$ such that $\forall$ $n \geq n_0$,
we have $0 \leq f(n) < cg(n)\}$.

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = 0$$

$g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Observe the difference in this definition from previous ones.

ω–notation (ω)

For a given function $g(n)$, the set little-omega:

$\omega(g(n))$ = {$f(n)$: $\forall\, c > 0$, $\exists\, n_0 > 0$ such that $\forall\, n \geq n_0$, we have $0 \leq cg(n) < f(n)$}.

$f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = \infty.$$

$g(n)$ is a ***lower bound*** for $f(n)$ that is not asymptotically tight.

Comparison of Functions [Summary]

$$f \leftrightarrow g \approx a \leftrightarrow b$$

$$f(n) = O(g(n)) \approx a \leq b$$

$$f(n) = \Omega(g(n)) \approx a \geq b$$

$$f(n) = \Theta(g(n)) \approx a = b$$

$$f(n) = o(g(n)) \approx a < b$$

$$f(n) = \omega(g(n)) \approx a > b$$

Properties

- **Transitivity**

$$f(n) = \Theta(g(n)) \ \& \ g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \ \& \ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o\,(g(n)) \ \& \ g(n) = o\,(h(n)) \Rightarrow f(n) = o\,(h(n))$$
$$f(n) = \omega(g(n)) \ \& \ g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

- **Reflexivity**

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

- **Symmetry**

$$f(n) = \Theta(g(n)) \text{ iff } g(n) = \Theta(f(n))$$

- **Complementarity**

$$f(n) = O(g(n)) \text{ iff } g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \text{ iff } g(n) = \omega((f(n))$$

- $f(n)$ is
  - **monotonically increasing**
    if $m \leq n \Rightarrow f(m) \leq f(n)$.
  - **monotonically decreasing**
    if $m \geq n \Rightarrow f(m) \geq f(n)$.
  - **strictly increasing**
    if $m < n \Rightarrow f(m) < f(n)$.
  - **strictly decreasing**
    if $m > n \Rightarrow f(m) > f(n)$.

Exercise

Express functions in A in asymptotic notation using functions in B.

A	B	
$5n^2 + 100n$	$3n^2 + 2$	$A \in \Theta(B)$

$A \in \Theta(n^2),\ n^2 \in \Theta(B) \Rightarrow A \in \Theta(B)$

$\log_3(n^2)$	$\log_2(n^3)$	$A \in \Theta(B)$

$\log_b a = \log_c a / \log_c b$; $A = 2\lg n / \lg 3$, $B = 3\lg n$, $A/B = 2/(3\lg 3)$

$n^{\lg 4}$	$3^{\lg n}$	$A \in \omega(B)$

$a^{\log b} = b^{\log a}$; $B = 3^{\lg n} = n^{\lg 3}$; $A/B = n^{\lg(4/3)} \to \infty$ as $n \to \infty$

$\lg^2 n$	$n^{1/2}$	$A \in o(B)$

$\lim_{n\to\infty} (\lg^a n / n^b) = 0$ (here $a = 2$ and $b = 1/2$) $\Rightarrow A \in o(B)$

What Next ?