



# Preliminaries

## **Instructors:**

Md Nazrul Islam Mondal

Department of Computer Science & Engineering

Rajshahi University of Engineering &

Technology Rajshahi-6204

# Outline

- 
- Mathematical Notations and Functions
  - Algorithm Notation
  - Control Structure
  - Complexity of Algorithm
  - Asymptotic Notations
  - Sub-algorithm
  - Variables, Data Type

# Mathematical Notations and Functions

# Mathematical Notations and Functions (Floor and Ceiling)

- Let  $x$  is a real number,

$\lfloor x \rfloor$  called floor of  $x$

$\lceil x \rceil$  called ceiling of  $x$

- If  $x$  is itself integer, then

$$\lfloor x \rfloor = \lceil x \rceil$$

- Otherwise,

$$\lfloor x \rfloor + 1 = \lceil x \rceil$$

*Example:*

$$\lfloor 3.4 \rfloor = 3 \text{ and } \lceil 3.4 \rceil =$$

$$\lfloor 3.4 \rfloor + 1 = 4 = \lceil 3.4 \rceil$$

# Mathematical Notations and Functions (Modular Arithmetic)

- Let  $k$  be an **integer** and  $M$  be a **positive integer**,

$$k(\bmod M)$$

(read  $k$  modulo  $M$ )

*Example:*

$$25 (\bmod 7) = 4$$

$$25(\bmod 5) = 0$$

$$-12(\bmod 7) = 2$$

# Mathematical Notations and Functions

## (Integer and Absolute value)

- Let  $x$  be any real number, the integer value of  $x$  (written as  $INT(x)$ ) converts  $x$  into an integer by deleting (truncation) the functional part of the number.

$$INT(3.4) = 3, INT(-8.5) = -8.5, INT(\sqrt{2}) = 1$$

- The absolute value of the real number  $x$  (written as  $ABS(x)$  or  $|x|$ ) is defined as the greater of  $x$  or  $-x$ .

$$|8.5| = 8.5, |-8.5| = 8.5$$

$$ABS(-4) = 4$$

# Mathematical Notations and Functions (Summation)

---

$$\sum_{i=0}^n a_i = a_0 + a_1 + a_2 + a_3 + \dots + a_n$$

$$\sum_{i=1}^5 i = 1 + 2 + 3 + 4 + 5 = 15$$

# Mathematical Notations and Functions (Factorial)

---

$$n! = 1.2.3.....(n - 2)(n - 1)n$$

$$4! = 1.2.3.4 = 24$$



# Mathematical Notations and Functions (Permutation)

- A **permutation** of a set of  $n$  elements is an arrangement of the elements in a given order.
- For **example**, the permutation of the set consisting of the elements  $a, b, c$  are as follows:

$abc, acb, bac, bca, cab, cba$

# Mathematical Notations and Functions

## (Exponents and Logarithm)

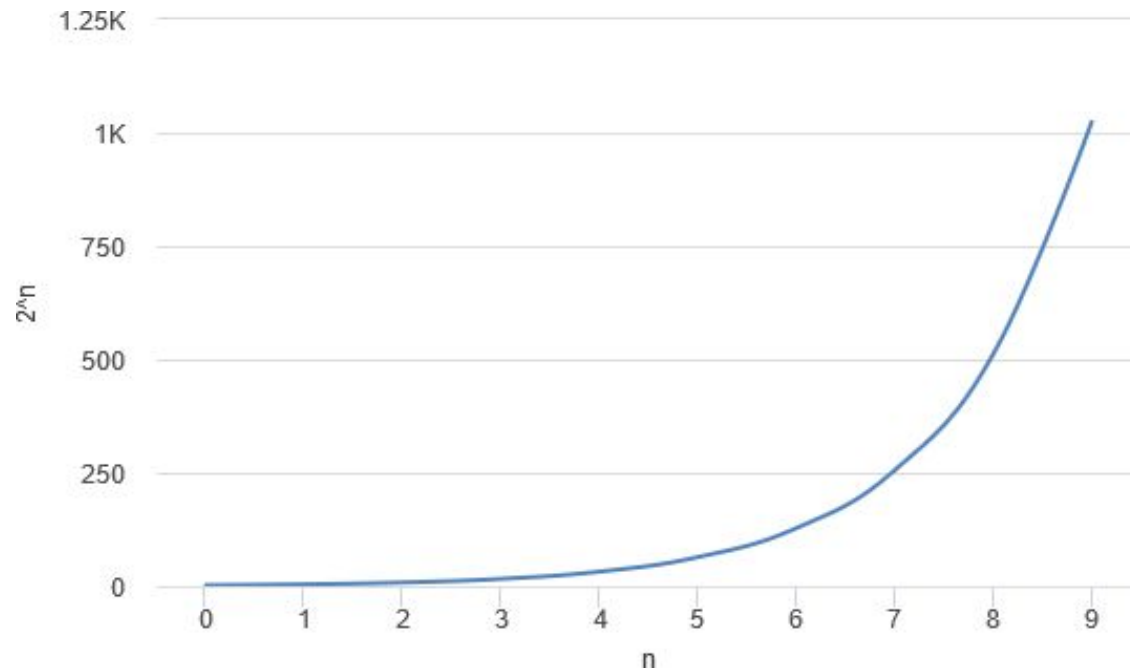
---

m be positive integer,  $a^m = a.a.a....a$ (m times)

m be zero,  $a^0 = 1$

m be negative integer,  $m = -n$ ,  $a^{-n} = \frac{1}{a^n}$

# Mathematical Notations and Functions (Exponents and Logarithm)



# Mathematical Notations and Functions (Exponents and Logarithm)

---

$$\log_b x$$

$$\textit{input} \rightarrow x$$

$$y = \log_b x$$

$$x = b^y$$

$$\textit{objective} \rightarrow \textit{find } y$$

# Mathematical Notations and Functions (Exponents and Logarithm)

---

$$\log_2 64$$

$$\textit{input} \rightarrow 64$$

$$y = \log_2 64$$

$$64 = 2^y$$

$$\textit{objective} \rightarrow y = 6$$

# Mathematical Notations and Functions

## (Exponents and Logarithm)

---

- programA.c and programB.c solve same problem.
- Let, 64 integers are used as input of programA.c and programB.c i.e.  $n = 64$
- Let, the complexity of programA.c is  $2^n = 2^{64}$
- Let, the complexity of programB.c is  $\log_2 n = \log_2 64 = 6$

# Algorithm Notation

# Algorithm Notations

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set** **K:=1, LOC:=1 and MAX:=DATA[1].**

**Step 2.** [Increment counter.] **Set** **K:=K+1**

**Step 3.** [Test counter.] **If** **K>N**  
    **then:** **Write:** **LOC, MAX and**  
    **Exit [End of If structure]**

**Step 4.** [Compare and update.] **If** **MAX<DATA[K]**  
    **then:** **Set** **LOC:=K and MAX:=DATA[K].**  
    **[End of If structure]**

**Step 5.** [Repeat loop.] **Go to Step 2.**



# Algorithm Notations (Identifying Number)

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

### Identifying Number:

**Step 1.** [initialize.] **Set** **K := 1**, **LOC := 1** and **MAX := DATA[1]**.

**Step 2.** [Increment counter.] **Set** **K := K + 1**

**Step 3.** [Test counter]. **If** **K > N** **then** :

**Write:** **LOC, MAX** **and** **Exit**

**Step 4:** [Compare and update.] **If** **MAX < DATA[K]** **then**:

**2.** **Set** **LOC := K** and **MAX := DATA[K]**.

Refers to the first  
algorithm in Chapter 2

# Algorithm Notations (Steps, Control and Exit)

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set K:=1, LOC:=1 and MAX:=DATA[1].**

**Step 2.** [Increment counter.] **Set K:=K+1**

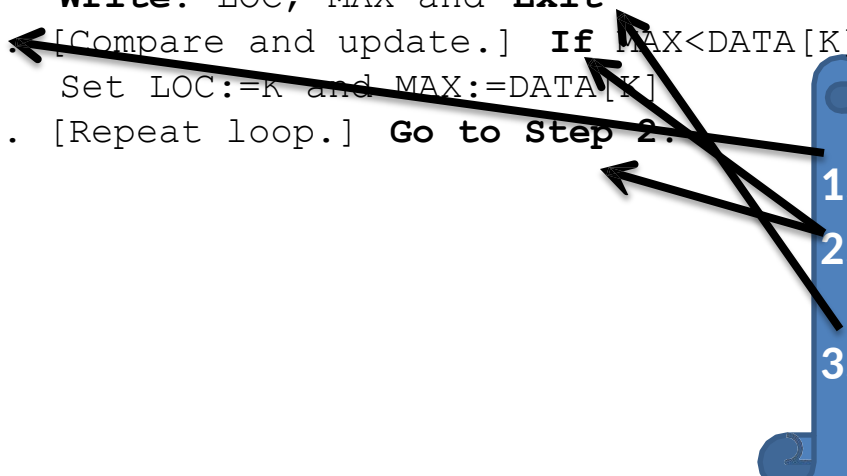
**Step 3.** [Test counter.] **If K>N then:**

**Write: LOC, MAX and Exit**

**Step 4.** [Compare and update.] **If MAX<DATA[K] then:**

**Set LOC:=k and MAX:=DATA[K]**

**Step 5.** [Repeat loop.] **Go to Step 2.**

- 
1. Step: statements
  2. Control: Loop and branch
  3. Exit: Complete the algorithm

# Algorithm Notations (Comments)

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set K:=1, LOC:=1 and MAX:=DATA[1].**

**Step 2.** [Increment counter.] **Set K:=K+1**

**Step 3.** [Test counter.] **If K>N then:**

**Write: LOC, MAX and Exit**

**Step 4.** [Compare and update.] **If MAX<DATA[K] then:**

**Set LOC:=K and MAX:=DATA[K].**

**Step 5.** [Repeat loop.] **Go to Step 2.**

Comments

# Algorithm Notations (Variable Names)

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set** **K:=1, LOC:=1 and MAX:=DATA[1].**

**Step 2.** [Increment counter.] **Set** **K:=K+1**

**Step 3.** [Test counter.] **If** **K>N** **then:**

**Write:** **LOC, MAX** and **Exit**

**Step 4.** [Compare and update.] **If** **MAX<DATA[K]** **then:**

**Set** **LOC:=K** and **MAX:=DATA[K].**

**Step 5.** [Repeat loop.] **Go to Step 2.**

**Variable Name:  
Use Capital Letter**

# Algorithm Notations (Assignment Statement)

## Algorithm 2.1:

(Largest Element in Array) **LARGE(DATA, N, LOC, MAX)**

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set K:=1, LOC:=1 and MAX:=DATA[1].**

**Step 2.** [Increment counter.] **Set K:=K+1**

**Step 3.** [Test counter.] **If K>N then:**

**Write: LOC, MAX and Exit**

**Step 4.** [Compare and update.] **If MAX<DATA[K] then:**

**Set LOC:=K and MAX:=DATA[K].**

**Step 5.** [Repeat loop.] **Go to Step 2.**

Assignment  
Statement

# Algorithm Notations (Input and Output)

## Algorithm 2.1:

(Largest Element in Array) **LARGE**(**DATA**, **N**, **LOC**, **MAX**)

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set** **K:=1**, **LOC:=1** and **MAX:=DATA[1]**.

**Step 2.** [Increment counter.] **Set** **K:=K+1**

**Step 3.** [Test counter.] **If** **K>N** **then**:

**Write**: **LOC**, **MAX** and **Exit**

**Step 4.** [Compare and update.] **If** **MAX<DATA[K]** **then**:

**Set** **LOC:=K** and **MAX:=DATA[K]**.

**Step 5.** [Repeat loop.] **Go to Step 2**

Read: x

Output  
Input

# Algorithm Notations (Procedures)

## Algorithm 2.1:

(Largest Element in Array) **LARGE**(**DATA**, **N**, **LOC**, **MAX**)

A nonempty array **DATA** with **N** numerical values is given. This algorithm finds the location **LOC** and the value **MAX** of the largest element of **DATA**.

**Step 1.** [initialize.] **Set** **K:=1**, **LOC:=1** and **MAX:=DATA[1]**.

**Step 2.** [Increment counter.] **Set** **K:=K+1**.

**Step 3.** [Test counter.] **If** **K>N** **then**:

**Write**: **LOC**, **MAX** and **Exit**

**Step 4.** [Compare and update.] **If** **MAX<DATA[K]** **then**:

**Set** **LOC:=K** and **MAX:=DATA[K]**.

**Step 5.** [Repeat loop.] **Go to Step 2**.

Procedure  
Example  
Procedure 4.3

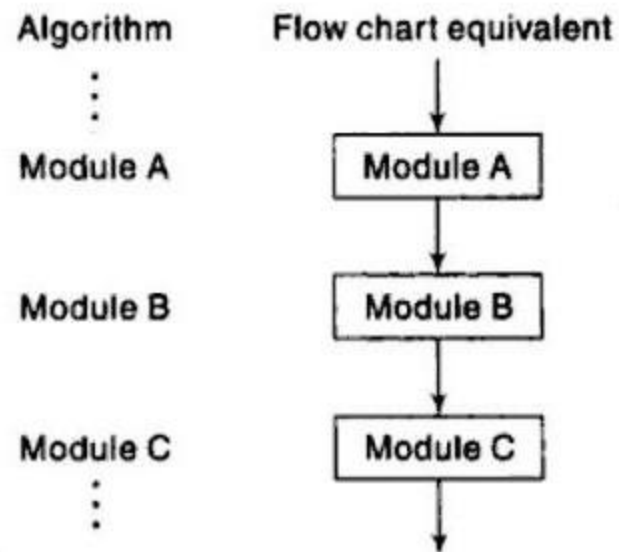
# Control Structures



# Control Structures

- 
- Sequence Logic or Sequential Flow
  - Selection Logic or Conditional Flow
  - Iteration Logic or Repetitive Flow

# Control Structures (Sequence Logic)

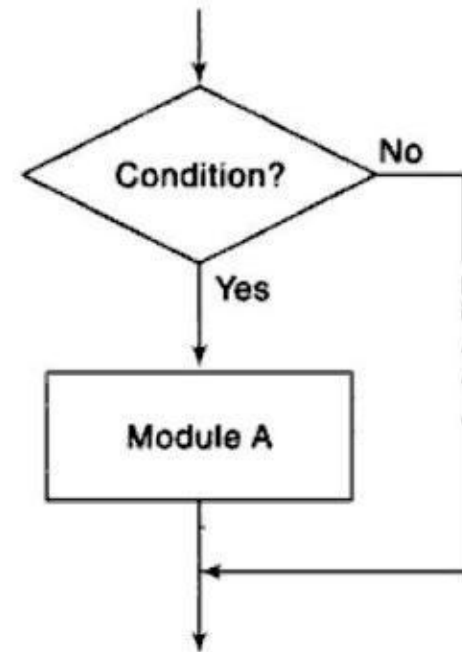


**Fig. 2.3** Sequence Logic

# Control Structures (Selection Logic )

## 1. Single Alternative:

```
If condition, then:  
    [Module A]  
[End of If structure]
```

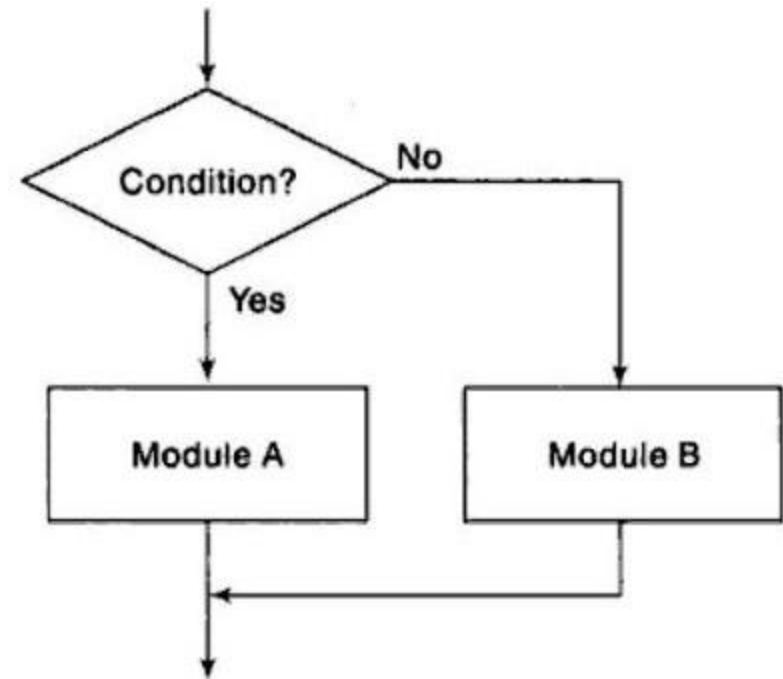


(a) Single alternative.

# Control Structures (Selection Logic )

## 1. Double Alternative:

```
If condition, then:  
    [Module A]  
Else:  
    [Module B]  
[End of If structure]
```



(b) Double alternative.

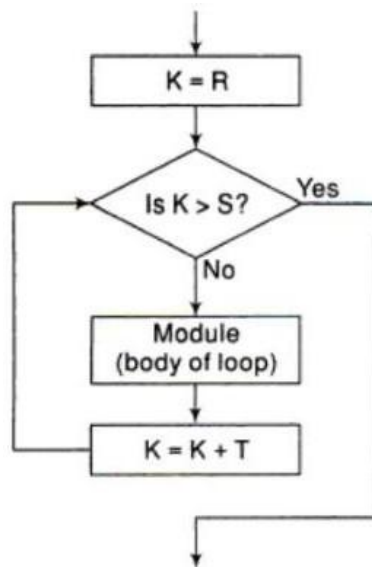
# Control Structures (Selection Logic )

## 1. Multiple Alternative:

```
If condition(1) , then:  
    [Module A1]  
Else If condition(2) , then:  
    [Module A2]  
    .  
    .  
    .  
Else If condition(M) , then:  
    [Module AM]  
Else:  
    [Module B]  
[End of If structure]
```

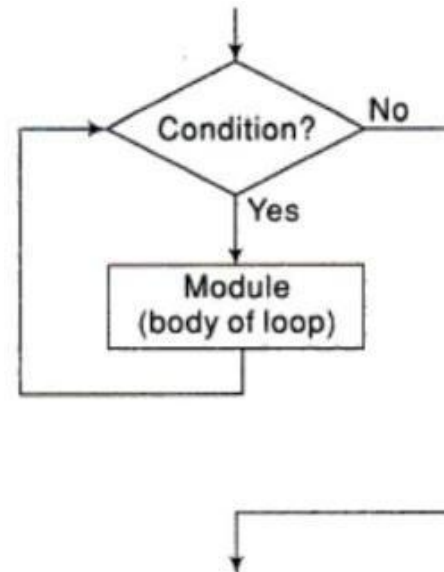
# Control Structures (Iteration Logic)

Repeat for  $K = R$  to  $S$  by  $T$ :  
    [Module]  
[End of loop]



(a) Repeat-For structure.

Repeat while condition:  
    [Module]  
[End of loop]



(a) Repeat-While structure.

# Complexity of Algorithm

# Complexity of Algorithm

---

- Let, M is an algorithm.
- The **time** and **space** used by M are the two measures for the efficiency of M.
- Time
  - Measured by counting the **number of key operations** – in sorting and searching algorithm, for example, the **number of comparisons**.
- Space
  - Measured by counting the maximum of memory needed by the algorithm.



# Complexity of Algorithm

---

- The complexity of M is **the function  $f(n)$**  which gives the running time and/or storage space requirement of the algorithm in terms of the size  $n$  of the input data.
- Frequently, space complexity =  **$Cn$**
- **The term “complexity ” shall refer to the running time of the algorithm.**

# Complexity of Algorithm

---

- Find the complexity function  $f(n)$  for certain cases-
  - **Best Case**
    - The minimum possible value of  $f(n)$  for any possible input
  - **Worst Case**
    - The maximum possible value of  $f(n)$  for any possible input
  - **Average Case**
    - The expected value of  $f(n)$

# Complexity of Algorithm

- **Average Case:**

- The analysis of average case assume a **probabilistic distribution** for the input data.
- Suppose the numbers  $n_1, n_2, \dots, n_k$  occur with respective **probabilities**  $p_1, p_2, \dots, p_k$ . The **expectation or average value  $E$**  is given by

$$E = n_1 p_1 + n_2 p_2 + \dots + n_k p_k$$

# Complexity of Algorithm (Linear Search)

**Algorithm 2.4:** (Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

1. [Initialize] Set  $K := 1$  and  $LOC := 0$ .
2. Repeat Steps 3 and 4 while  $LOC = 0$  and  $K \leq N$ .
3.     If  $ITEM = DATA[K]$ , then: Set  $LOC := K$ .
4.     Set  $K := K + 1$ . [Increments counter.]
- [End of Step 2 loop.]
5. [Successful?]  
   If  $LOC = 0$ , then:  
      Write: ITEM is not in the array DATA.  
   Else:  
      Write: LOC is the location of ITEM.  
   [End of If structure.]
6. Exit.

# Complexity of Algorithm (Linear Search)

- The complexity of the search algorithm is given by the number of  $C$  of comparisons between ITEM and DATA[K].
- **Worst Case:**
  - Clearly, the worst case occurs when ITEM is the last element in the array DATA or is not there at all.

$$C(n) = n$$

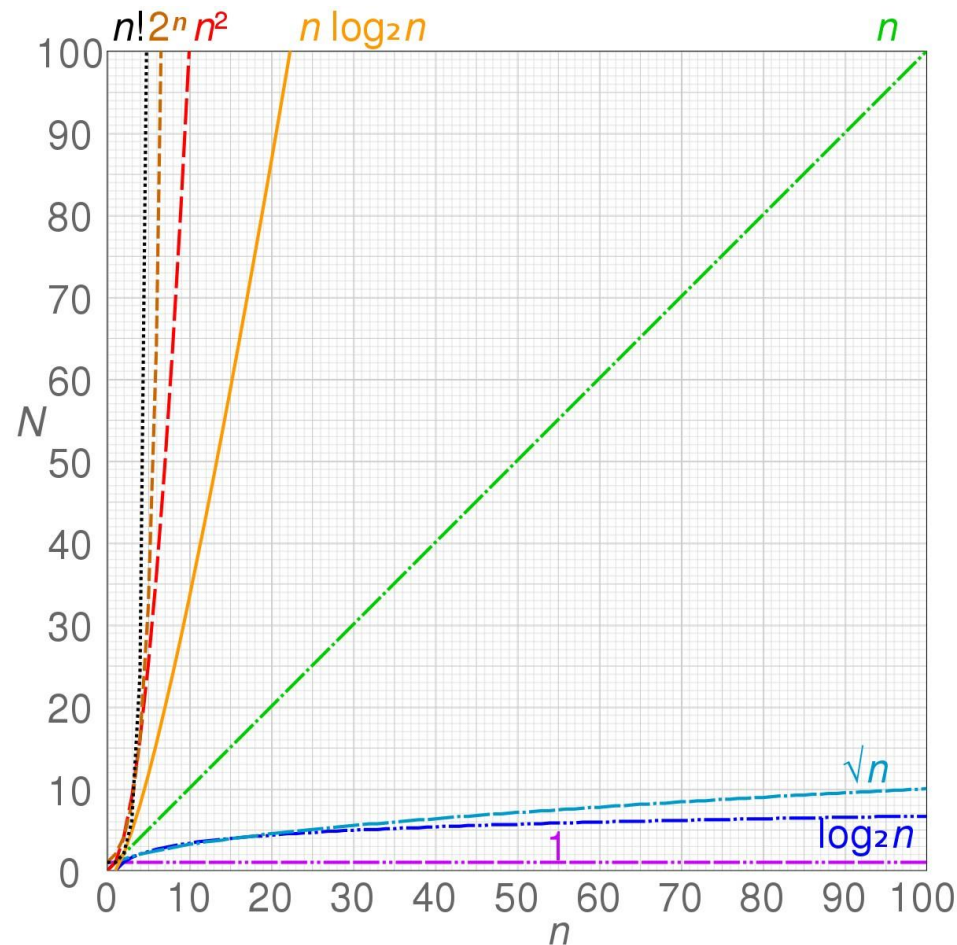
# Complexity of Algorithm (Linear Search)

- **Average Case:**

- Assume that ITEM does appear in DATA
- Accordingly the number of comparisons can be any of the numbers 1, 2, 3, . . . , n
- Each number occurs with probability  $p = 1/n$

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + 3 \cdot \frac{1}{n} + \dots + n \cdot \frac{1}{n} = \frac{1}{n} (1 + 2 + 3 + \dots + n) = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

# Complexity of Algorithm



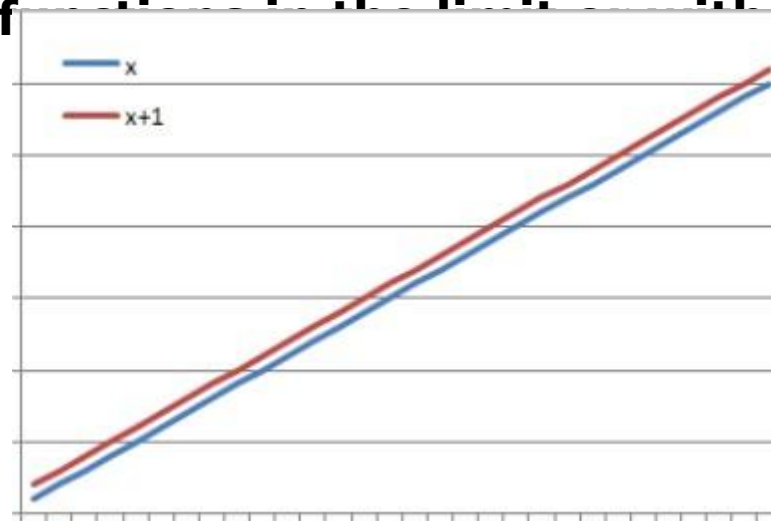
# Asymptotic Notations



# Asymptotic Notations

- **Asymptotic:**

- It means a line that continually approaches a given curve but does not meet it at any finite distance.
- **Example:**  $x$  is asymptotic with  $x + 1$  as shown in graph.
- Asymptotic may also be defined as a way to **describe the behavior of f** **out bounds**



# Asymptotic Notations

---

- Big-Oh Notation ( $O$ )
- Big-Omega Notation ( $\Omega$ )
- Big-Theta Notation ( $\Theta$ )
- Little-Oh Notation ( $o$ )

# Asymptotic Notations

- 
- Suppose  $f(n)$  and  $g(n)$  are **positive functions** with the property that
    - $f(n)$  is **bounded** by some multiple of  $g(n)$  for almost all  $n$ .

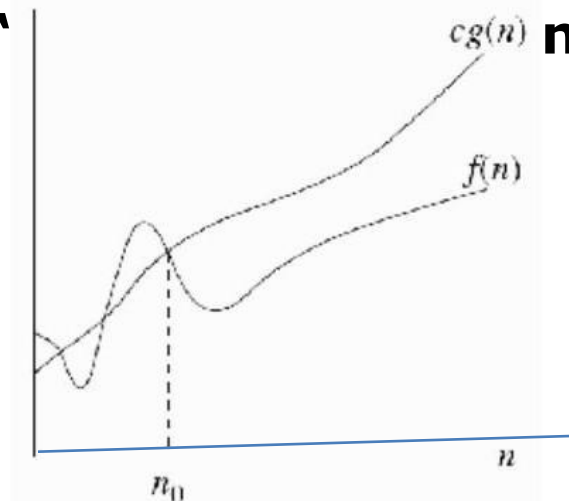
# Asymptotic Notations (Big-Oh Notation)

- Suppose there exist a positive integer  $n_0$  and a positive number  $M$  such that, for all  $n \geq n_0$ , we have
 
$$|f(n)| \leq M |g(n)|$$

- Then we may **write**

$$f(n) = O(g(n))$$

- Which is **read** “ $f(n)$  is  $O(g(n))$ ”



# Asymptotic Notations (Big-Oh Notation)

- **Example:**
  - Given,  $f(n) = n^2 + 50n$
  - **We can write,  $n^2 + 50n \leq n^2 + 50n^2$  when  $n \geq 0$**
  - **$n_0 = 0$**
  - We have,  $n^2 + 50n \leq 51n^2$  where  $n \geq n_0$
  - **$M = 51, g(n) = n^2$**
  - We can write,  $f(n) = O(g(n))$  i.e.  **$f(n) = O(n^2)$**
- **Upper Bound**

# Asymptotic Notations (Big-Omega Notation)

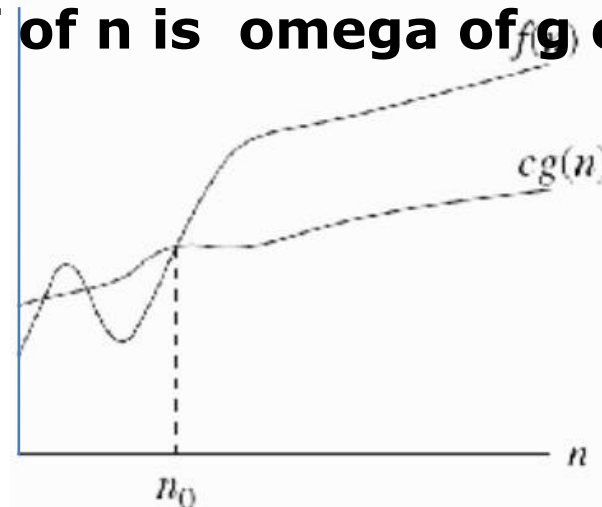
- Suppose there exist a positive integer  $n_0$  and a positive number  $M$  such that, for all  $n \geq n_0$ , we have

$$|f(n)| \geq M|g(n)|$$

- Then we may **write**

$$f(n) = \Omega(g(n))$$

- Which is **read "f of n is omega of g of n"**



# Asymptotic Notations (Big-Omega Notation)

- **Example:**
  - Given,  $f(n) = n^2 + 50n$
  - **We can write,  $n^2 + 50n \geq n^2$  when  $n \geq 0$**
  - **$n_0 = 0$**
  - We have,  $n^2 + 50n \geq n^2$  where  $n \geq n_0$
  - **$M = 1, g(n) = n^2$**
  - We can write,  $f(n) = \Omega(g(n))$  i.e.  **$f(n) = \Omega(n^2)$**
- **Lower Bound**

# Asymptotic Notations (Big-Theta Notation)

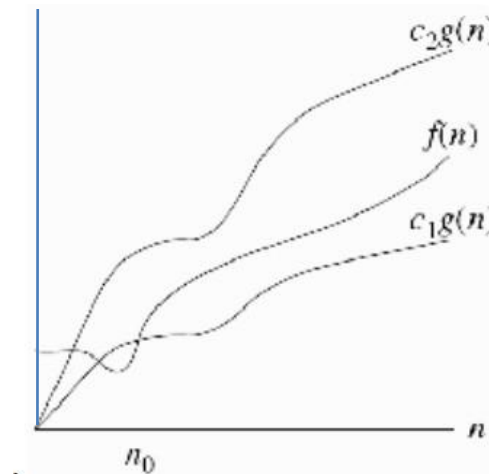
- Suppose there exist a positive integer  $n_0$  and two positive number  $c_1$  and  $c_2$  such that, for all  $n \geq n_0$ , we have

$$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|$$

- Then we may **write**

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

- Which is **read "f of n is theta of g of n"**





# Asymptotic Notations (Big-Omega Notation)

- **Example:**
  - Given,  $f(n) = n^2 + 50n$
  - **We can write,  $n^2 + 50n \geq n^2$  when  $n \geq 0$**
  - **$c_1 = 1, n_0 = 0$**
  - **We can write,  $n^2 + 50n \leq n^2 + 50n^2$  when  $n \geq 0$**
  - **$c_2 = 51, n_0 = 0$**
  - **We can write,  $n^2 \leq n^2 + 50n \leq 51n^2$  when  $n \geq 0$**
  - **$g(n) = n^2$**
  - We can write,  $f(n) = \Theta(g(n))$  i.e.  **$f(n) = \Theta(n^2)$**
- **Both upper and lower Bound**

# Asymptotic Notations (Little-Oh Notation)

- Iff  $f(n) = O(g(n))$  and  $f(n) \neq \Omega(g(n))$
- Then we may **write**  
$$f(n) = o(g(n))$$
- Which is **read “f of n is little oh of g of n”**

# Asymptotic Notations (Little-Oh Notation)

- **Example:**
  - Given,  $f(n) = 18n+9$
  - **We can write,  $18n+9 \leq 18n+9n$  when  $n \geq 0$**
  - **So  $f(n) = O(n)$**
  - **We can write,  $18n+9 \geq 18n$  when  $n \geq 0$**
  - **So  $f(n) = \Omega(n)$**
  - **So  $f(n) \neq o(n)$**

**But**

- **We can write,  $18n+9 \leq 27n^2$  when  $n \geq 1$**
- **So  $f(n) = O(n^2)$**
- **But  $f(n) \neq \Omega(n^2)$**
- **So  $f(n) = o(n)$**

# Sub-Algorithm

# Sub-Algorithm

- A subalgorithm is a complete and **independently** defined algorithm module which is used (or invoked or called) by some **main algorithm** or by some **other subalgorithm**.

# Variables, Data Type

# Variables, Data Type

---

- Four Data Type:
  - » Character
  - » Real (or floating point)
  - » Integer (or fixed point)
  - » Logical
- Local and global variables

---

# END