SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF MCA

19CAT602 – DATA STRUCTURES & ALGORITHMS

I YEAR I SEM

UNIT IV – Greedy and Backtracking

TOPIC:19 – Fundamentals of the analysis of algorithm efficiency

Fundamentals of the analysis of algorithm efficiency

Algorithm

Definition:

1. An **algorithm** is a set of instructions designed to perform a specific task.
2. It is unambiguous set of instructions
3. An algorithm is a set of well-defined instructions in sequence to solve a problem.

Algorithm to add two numbers entered by the user

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

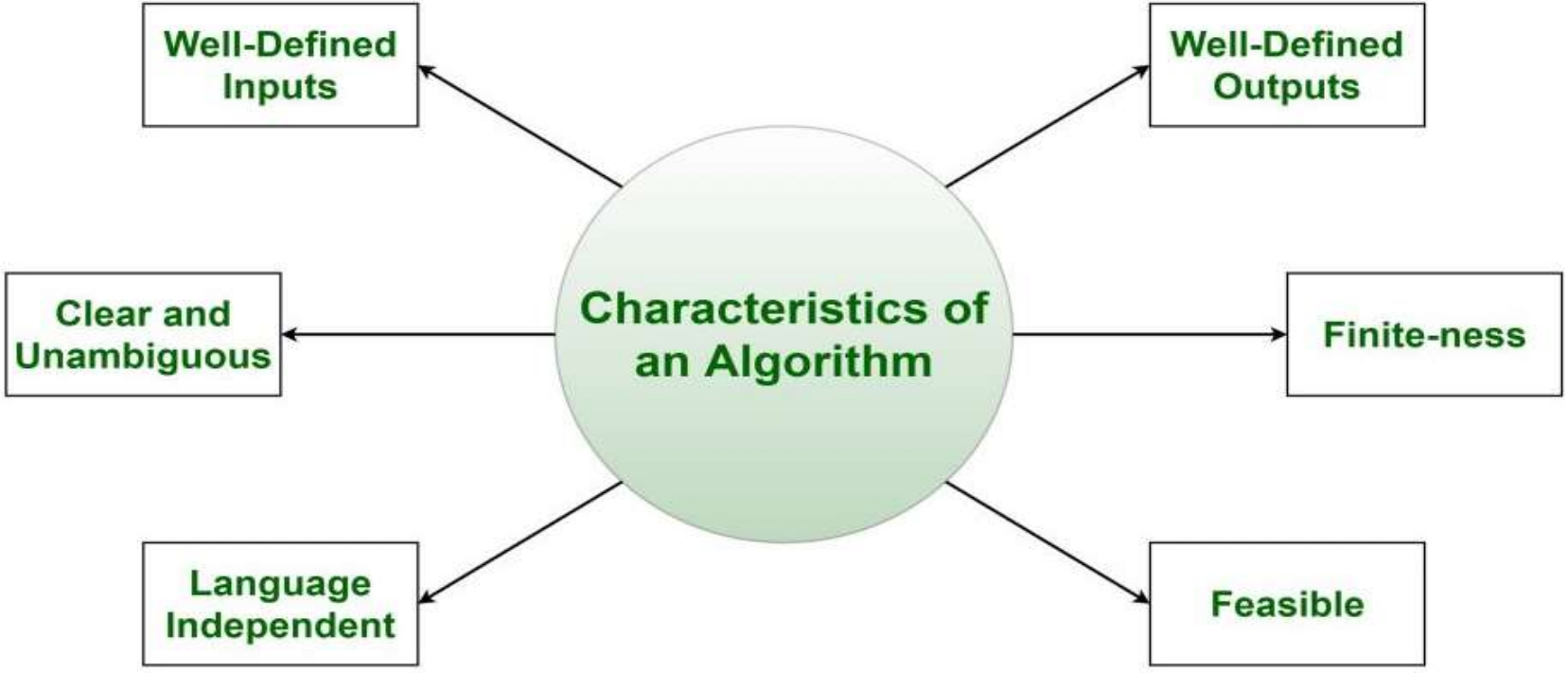
Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

sum←num1+num2

Step 5: Display sum Step 6: Stop

Qualities of a good algorithm



ANALYSIS FRAMEWORK

- Efficiency of an algorithm can be in terms of time or space. Thus, checking whether the algorithm is efficient or not means analyzing the algorithm. This systematic approach is modelled by a framework called as **ANALYSIS FRAMEWORK.**

- The efficiency of an algorithm can be decided by measure the performance of an algorithms

Analysis of algorithm is the process of investigation of an algorithm's efficiency respect to two resources:

I. Running time

II. Memory space

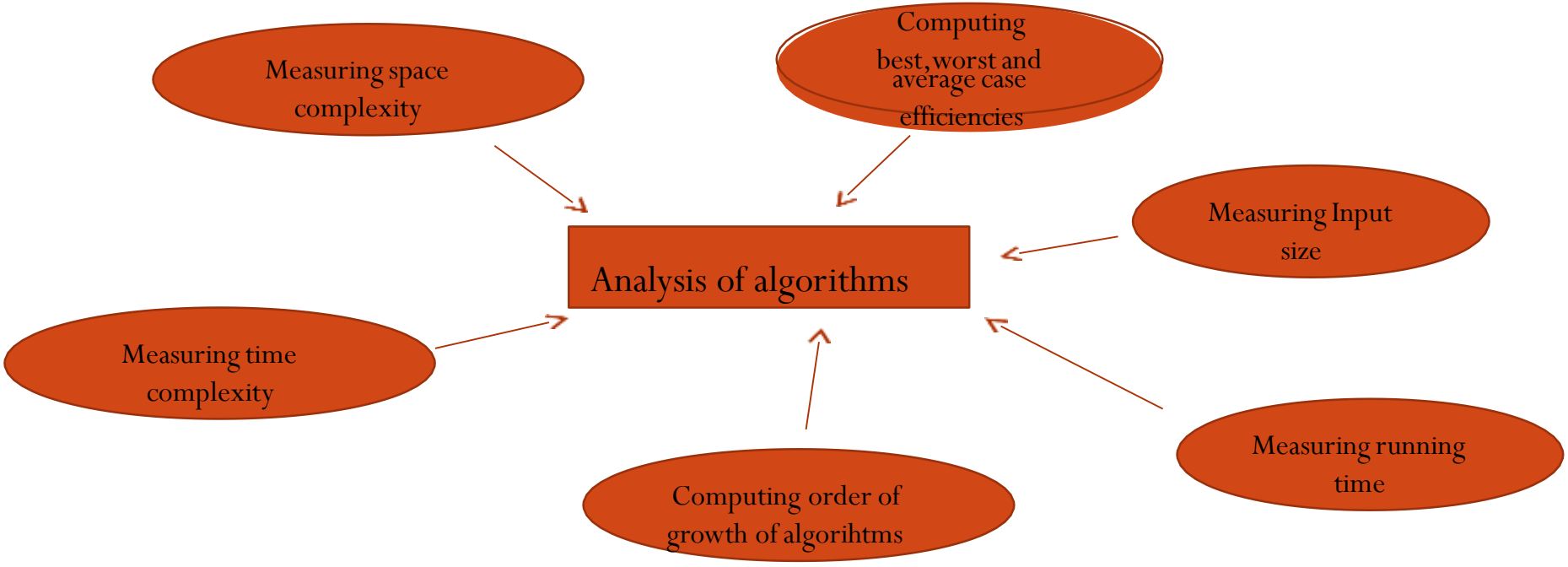
The reason for selecting these two criteria are

- Simplicity
- Generality
- Speed
- Memory

Time efficiency or time complexity indicates how fast an algorithm runs.

Space Efficiency or space complexity is the amount of memory units required by the algorithm including the memory needed for the i/p & o/p

ANALYSIS OF ALGORITHMS



Space complexity

- The amount of memory required by an algorithm to run.
- To Compute the space complexity we use two factors: constant and instance characteristic

$$S(p) = C + Sp$$

C – Constant -> space taken by instruction, variable and identifiers Sp – space dependent upon instance characteristic

For eg: add(a,b) return a+b $S(p) = C + Sp$ $S(p)=C+2$

a,b occupy one word size then total size come to be 2

Time complexity

- The amount of time required by an algorithm to run for completion.
- For instance in multiuser system , executing time depends on many factors such as:
  1. System load
  2. Number of other programs running
  3. Instruction set used
  4. Speed of underlying hardware

  Frequency count is a count denoting number of times of execution of statement

For eg: Calculating sum of n numbers

```
for(i=0;i<n;i++)
{
sum=sum+a[i];
}
```

Statement	Frequency count
i=0	1
i<n	N+1
i++	n
Sum=sum+a[i]	n
Total	3n+2

Time complexity normally denotes in terms of Oh notation(O).
Hence if we neglect the constants then we get the time complexity to be O(n)

Eg: Matrics addition

```
For(i=0;i<n;i++)
{
   for(j=0;j<n;j++)
   {
          c[i][j]=a[i][j]+b[i][j]
   }
}
```

The frequency count is:

Statement	Frequency count
i=0	1
i<n	N+1
i++	n
j=0	N * 1 = n For Outer loop; Initialization of j
j<n	N * (n+1) =n^2 + n times For Outer loop
j++	n * n = n^2
C[i][j]=a[i][j]+b[i][j]	n * n = n^2
Total	$3n^2$ +4n+2 $O(n^2)$

Measuring an Input size:

- Efficiency measure of an algorithm is directly proportional to the input size or range
- So an alg efficiency could be measured as a function of n, where n is the parameter indicating the algorithm i/p size.
- For ex: when multiplying two matrices, the efficiency of an alg depends on the no. of multiplication performed not on the order of matrices.
- The i/p given may be a square or a non-square matrix.
- Some algortihm require more than one parameter to indicate the size of their i/p
- In such situation, the size is measured by the number of bits in the n's binary representation:

$$B=\text{floor}(\log_2 n+1)$$

Eg:

- Sorting
- Naive Algorithm – n^2
- Best Algorithm – nlogn

Units for measuring Running time

- The running time of an alg depends on:
  - Speed of a particular computer
  - Quality of a program
  - Compiler used

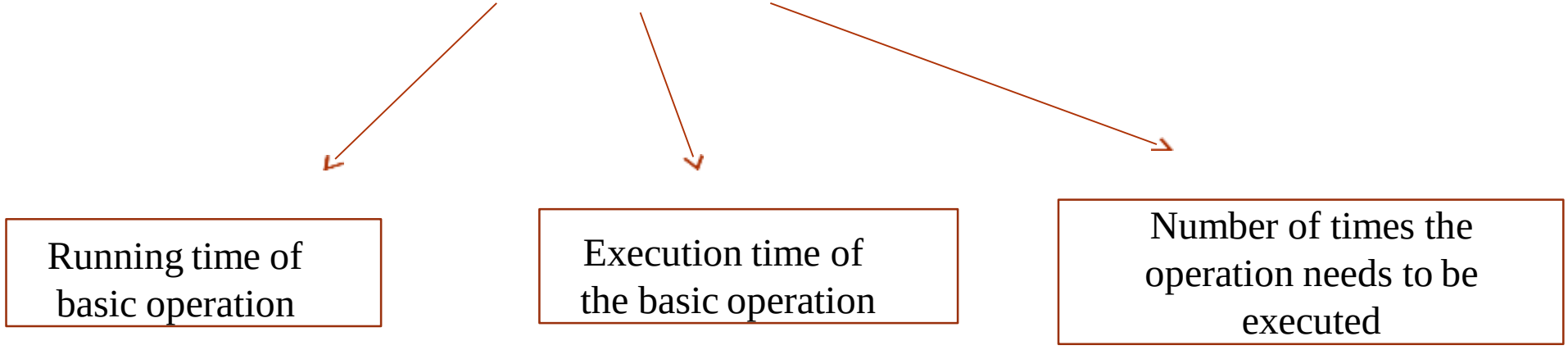
To measure the alg efficiency:

- Identify the important operation(core logic) of an algorithm.
  This operation is called **basic operation**
- So compute the no. of times the basic operation is executed will give running time
- Basic operation mostly will be in inner loop, it is time consuming

Problem statement	Input size	Basic operation
Searching a key element from the list of n elements	List of n elements	Comparison of key with every element of list
Perform matrix multiplication	The two matrices with order n x n	Actual multiplication of the elements in the matrices
Computing GCD of two numbers	Two numbers	Division

- Using the formula the computing time can be obtained

$$T(n) = Cop\ C(n)$$

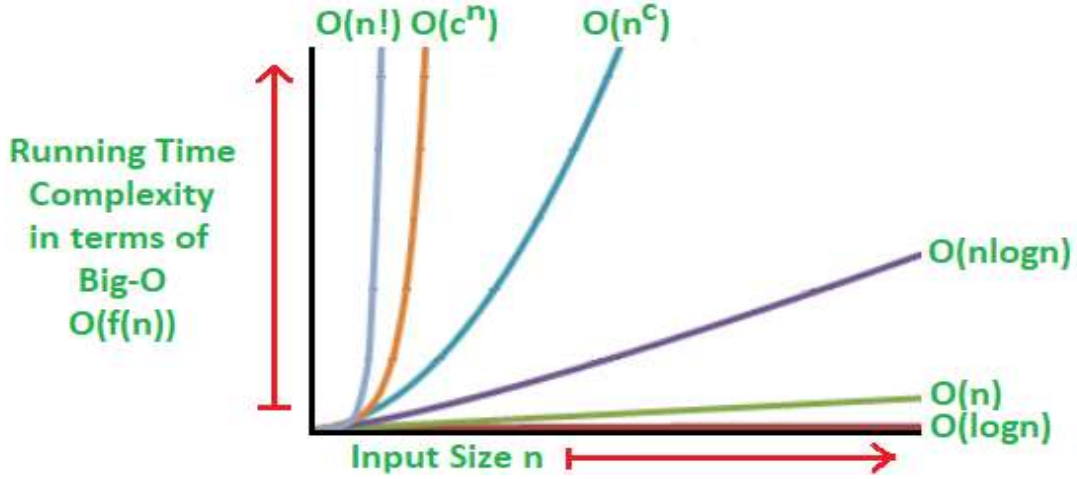


Order of Growth

- Measuring the performance of an algorithm in relation with the input size 'n' is called order of growth.

Order of growth for varying input size of 'n'

n	Log n	n log n	n^2	2^n
1	0	0	1	1
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296



O(n!),O(c^n),O(n^c) - Worst

O(nlogn) - Bad

O(n) - Fair

O(logn) - Good

O(1) - Best

Rate of growth of common computing time function

Three case efficient

Worst Case Analysis (Usually Done)

we calculate upper bound on running time of an algorithm

The element to be searched (x in the above code) is not present in the array

```
int arr[] = { 1, 10, 30, 15 };
Searching Element=11
Time complexity=O(n)
```

Best Case Analysis (Bogus)

```
int arr[] = { 1, 10, 30, 15 };
Searching Element=1
Time complexity=O(1)
```

we calculate lower bound on running time of an algorithm.

minimum number of operations to be executed

x is present at the first location

Three case efficient

Average Case Analysis (Sometimes done)

we take all possible inputs and calculate computing time for all of the inputs.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)}$$

$$= \frac{\theta((n+1)*(n+2)/2)}{(n+1)}$$

$$= \Theta(n)$$