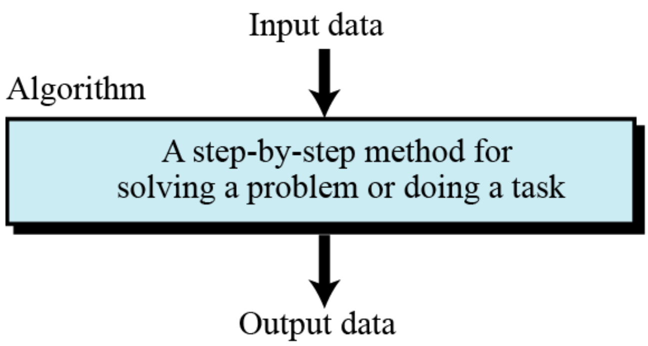
DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

Module1,2

Radhika Chapaneri

An informal definition of an algorithm is:

Algorithm: a step-by-step method for solving a problem or doing a task.



Definition

- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.
- An algorithm is any well defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
- An algorithm is thus a sequence of computational steps that transform the input into the output.

What is Analysis of algorithms

- ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.

Properties of Algorithms

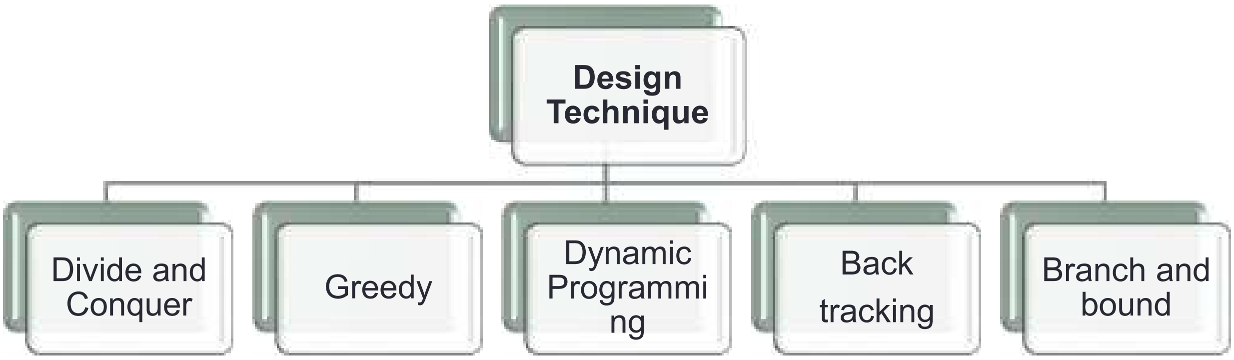
- **Input :** an algorithm accepts zero or more inputs
- **Output :** it produces at least one output.
- **Finiteness:** If we trace out the instructions of an algorithm, then for all cases, the terminates after a finite number of steps.
- **Definiteness:** Each step in algorithm is unambiguous. This means that the action specified by the step cannot be in multiple ways & can be performed without any confusion.

Effectiveness: It consists of basic instructions that are realizable. The operations are doable

Algorithm design techniques

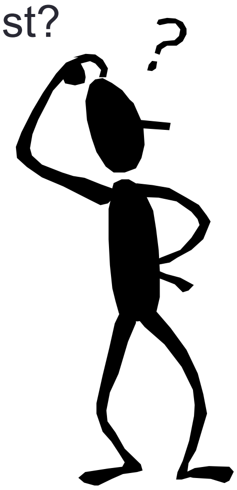
- **Brute force**: For many nontrivial problems, there is a natural brute-force search algorithm that checks every possible solution.
- Typically takes 2^n time or worse for inputs of size n.
- Unacceptable in practice.

Algorithm design techniques



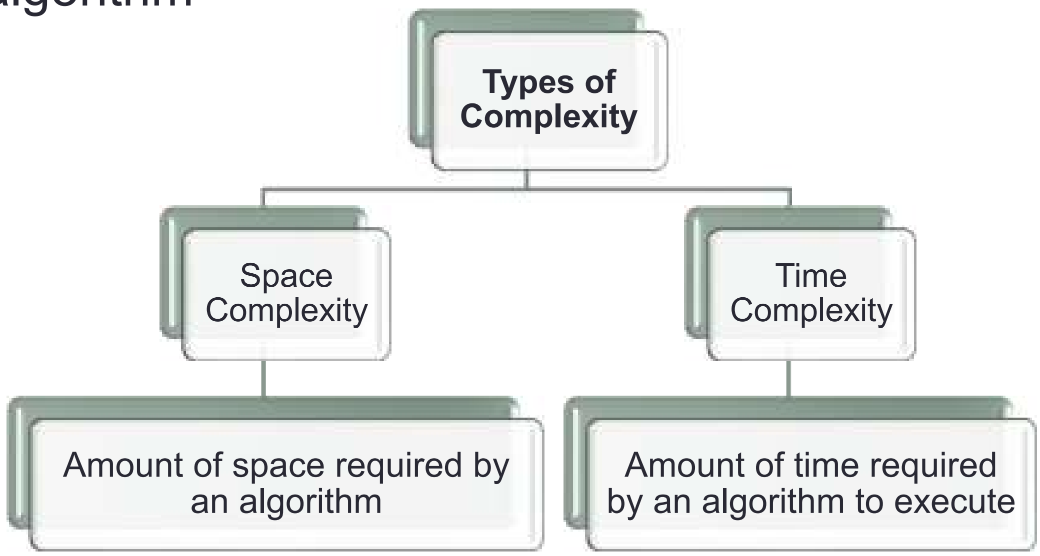
Performance analysis

- When is a program / algorithm said to be better than another?
- The algorithms are correct, but which is the best?
- What are the measures for comparison?



Performance analysis

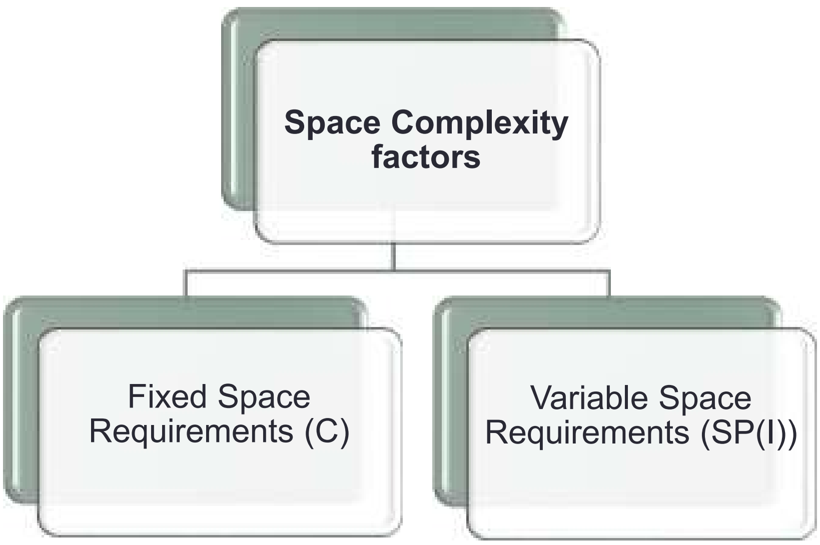
- The computational complexity of an algorithm addresses the resources needed to run the algorithm



Notes Compiled by Radhika Chapaneri

Space Complexity

- **Space complexity** = The amount of memory required by an algorithm to run to completion



Space Complexity

- **Fixed Space Requirements (C)** : **Independent of the characteristics of the inputs and outputs**
  - instruction space
  - space for simple variables
  - fixed-size structured variable
  - Constants
- **Variable Space Requirements (SP(I)) : depend on the instance characteristic I**
  - number, size, values of inputs and outputs associated with I
  - recursive stack space, local variables, return address

Space Complexity

Total space complexity:

$$S(P) = C + S_P(I)$$

Where

C = Fixed Space Requirements

$S_P(I)$ = Variable Space Requirements

S(P) = Total space complexity

.

Space Complexity : Simple Example

// This algorithm computes addition of three elements

// Input: a, b, c are of floating type

// Output: The addition is returned

Algorithm:

```
Add (a, b, c)
{
    return a + b + c
}
```

Space Complexity:

$S(P) = C + S_P(I)$

$S_P(I) = 0$, $S(P) = C$

If we assume that a, b and c occupy one word size then $S(P) = 3$

Space Complexity : Using array

// This algorithm computes addition of all the elements in an array.

// Input: Array x of floating type, n is total number of elements

// Output: returns sum which is of data type float

Algorithm:

```
{
        sum = 0.0;
        for i = 1 to n do
                sum = sum + x[i]
        return sum
}
```

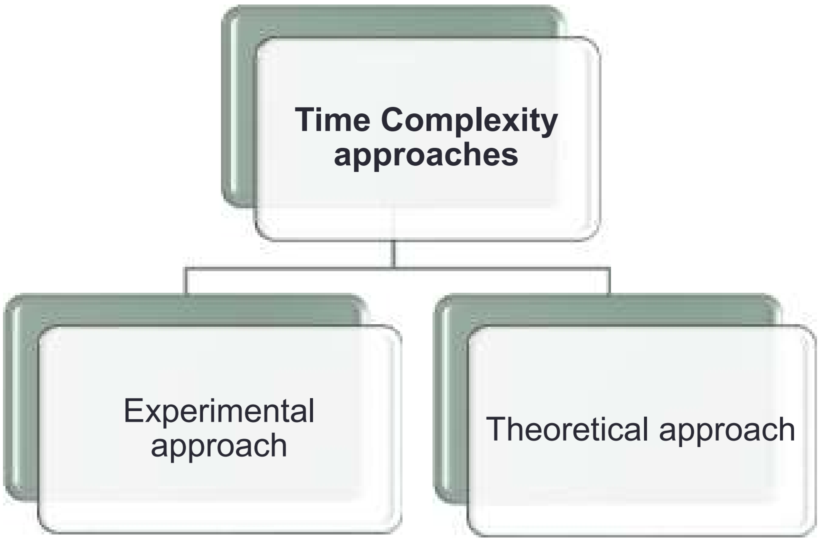
Space Complexity:

$S(P) = C + S_P(I)$

The 'n' space required for x[ ],
one unit space for n,
one unit for I and
one unit for sum.

$S(P) \geq (n+3)$

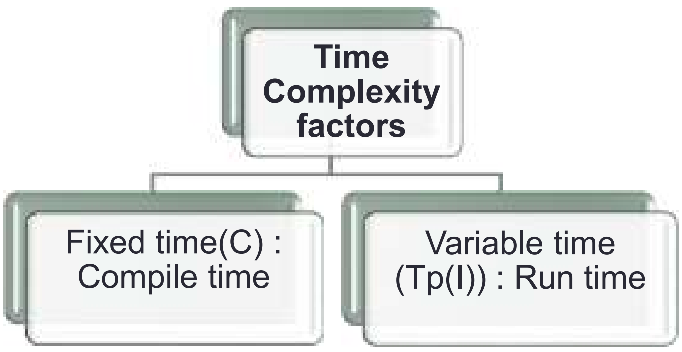
Time Complexity



Notes Compiled by Radhika Chapaneri

How to find time complexity?

$$T(P) = C + T_P(I)$$



- The compile time does not depend on the instance characteristics.
- We assume that a compiled program run several times without recompilation
- Hence we are concern just about the running time of the program.

Time Complexity : Using array

// Algorithm computes addition of all the elements in an array.
// Input: Array x of floating type, n is total number of elements
// Output: returns sum which is of data type float

```
Algorithm:
{
  sum = 0.0;
  for i = 1 to n do
          sum = sum + x[i];
  return sum;
}
```

Algorithm	Steps/ execution	Frequency	Total steps
Algorithm Sum(a[ ], n)	0	-	0
{	0	-	0
sum = 0.0;	1	1	1
for i = 1 to n do	1	n + 1	n + 1
sum = sum + x[i];	1	n	n
return sum;	1	1	1
}	0	-	0
Time Complexity step per execution × frequency			**2n + 3**

Time Complexity : Matrix addition

// This algorithm computes addition of all the elements in matrix
// Input: two matrices a, b
// Output: returns sum in matrix

```
Algorithm:
Add(a,b,c,m,n)
{
 for i = 1 to m do
   for j = 1 to n do
      c[i, j] = a[i, j] + b[i, j]
}
```

Algorithm	s/e	f	Total steps
Add(a,b,c,m,n)	0	-	0
{	0	-	0
for i = 1 to m do	1	m + 1	m + 1
for j = 1 to n do	1	m (n+1)	mn +m
c[i, j] = a[i, j] + b[i, j]	1	mn	mn
}	0	-	0
Time Complexity			**2mn + 2m + 1**

Order of growth

Time complexity of algorithm A and B

Input	Algo1 = 100n	Algo B = $10n^2$	Algo C = 2^n
n	100n	$10n^2$	2^n
5	500	250	32
10	1000	1000	1024
20	2000	4000	1,048,57
50	5000	25,000	$1.259X10^{15}$
1000	100,000	10,000,000	$>10^{\wedge 300}$

Order of growth

- If an algorithm need 2^n steps for execution, then when n = 40, the number of steps needed is approximately
  $1.1* 10^{12}$.
- On a computer performing one billion steps per second, this would require about 18.3 minutes for n = 40.
- If n = 50 the same algorithm would run for about 13 days on this computer.
- When n = 100 about $4 *10^{13}$ years are needed.
- So utility of algorithm with exponential complexity is limited to small n (n<40)