# Design and Analysis of Algorithms

## Unit - I

**Dr. R. Bhuvaneswari**
Assistant Professor
Department of Computer Science
Periyar Govt. Arts College, Cuddalore.

Periyar Govt. Arts College
Cuddalore

**Syllabus**

**UNIT -I: ALGORITHM AND ANALYSIS**

What is an Algorithm? - Algorithm Specification - Performance Analysis - Randomized Algorithms.

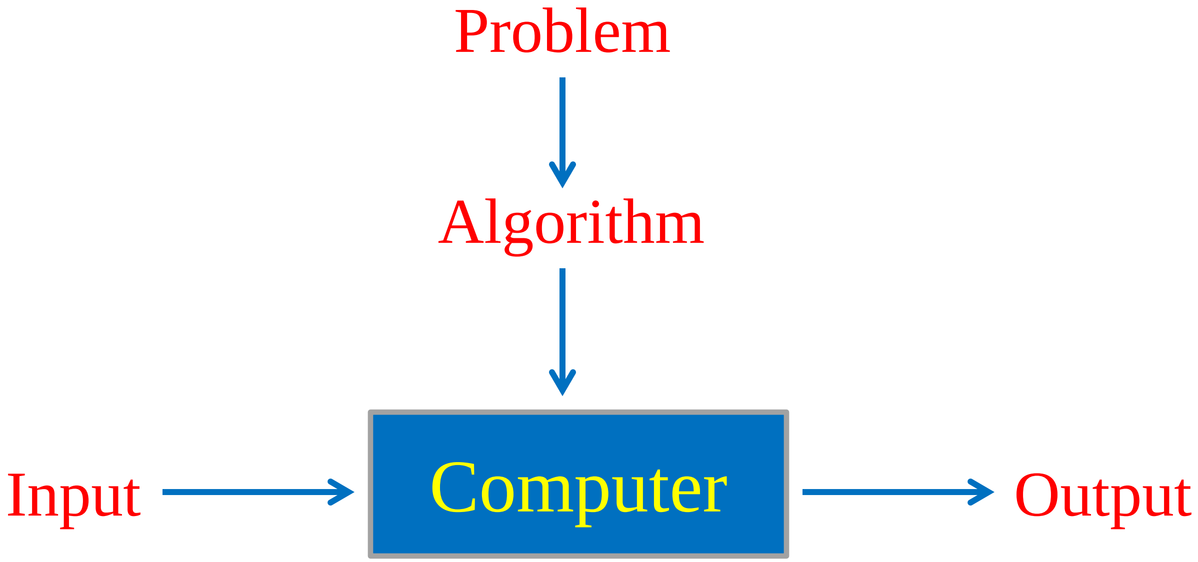
**TEXT BOOK**

Fundamentals of Computer Algorithms, Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, Galgotia Publications, 2015.

# Introduction to the Concept of Algorithms

- Algorithm
- Problem Solving
- Design of an Algorithm
- Analysis of an algorithm

# Notion of an Algorithm



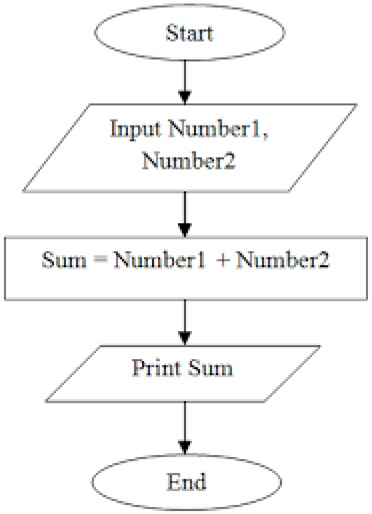
# Algorithm

- An **algorithm** is a finite set of instructions that, if followed, accomplishes a particular task i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- All algorithms must satisfy the following criteria:
  - **Definiteness.** Each instruction is clear and unambiguous.
  - **Effectiveness.** Every instruction must be very basic so that it can carried out, by a person using pencil and paper.
  - **Finiteness.** If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
  - **Input.** Zero or more quantities are externally supplied.
  - **Output.** At least one quantity is produced.

# Algorithm Specification

- An **algorithm** can be described in three ways:
  - Natural language in English
  - Graphic representation called flowchart
  - **Pseudo-code method**
    - In this method we typically represent algorithms as program, which resembles C language

1. Input two numbers
2. Add the two numbers
3. Print the result



# Pseudo-code Conventions

1. Comments begin with // and continue until the end of line.
2. Blocks are indicated with matching braces **{** and **}**.
3. An identifier begins with a letter. The data types of variables are not explicitly declared.
4. Assignment of values to variables is done using the assignment statement.

   ‹variable› := ‹expression›;
5. There are two Boolean values **true** and **false**.
   - Logical operators: AND, OR, NOT
   - Relational operators: $<, \leq, =, \neq, >, \geq$

# Pseudo-code Conventions

6. The following looping statements are used:
   **while**, **for** and **repeat-until**

**while loop:**

```
while ‹condition› do
{
        ‹statement 1›
              .
              .
        ‹statement n›
}
```

**for loop:**

```
for variable:= value1 to  value2
              step step-value do
{
              ‹statement 1›
                    .
                    .
              ‹statement n›
}
```

**repeat-until:**

```
repeat
              ‹statement 1›
                    .
                    .
              ‹statement n›
until ‹condition›
```

# Pseudo-code Conventions

7. A conditional statement has the following forms:

    **if** ‹condition› then ‹statement›

    **if** ‹condition› then ‹statement 1› **else** ‹statement 2›

**case statement:**

```
case
{
    :‹condition 1›: ‹statement 1›
                .
                .
    :‹condition n›: ‹statement n›
    :else: ‹statement n+1›
}
```

# Pseudo-code Conventions

8. Input and output are done using the instructions **read** and **write**.
9. There is only one type of procedure: Algorithm.

    Algorithm contains

    - **Heading**
    - **Body**

    The heading takes the form

    Algorithm Name (‹parameter list›) ⟶ heading

    {

    ……

    ……  } body

    }

# Pseudo-code Conventions

```
Algorithm Max(A, n)
// A is an array of size n.
{
Result := A[1];
for i :=2 to n do
    if A[i] > result then
        Result := A[i];
return Result;
}
```

n = 5, Result = 10

A[1] = 10

A[2] = 87 Result = 87

A[3] = 45

A[4] = 66

A[5] = 99 Result = 99

# Performance Analysis

1. Space Complexity
2. Time Complexity

**Space complexity** of an algorithm is the amount of memory it needs to run to complete.

Space needed by an algorithm is given by
S(P) = C(fixed part) + Sp(variable part)

**fixed part:** independent of instance characteristics. Eg. Space for simple variables, constants etc.

**variable part:** space for variables whose size is dependent on particular problem instance

```
Algorithm Max(A, n)
// A is an array of size n.
{
Result := A[1];
for i :=2 to n do
if A[i] > result then
        Result := A[i];
return Result;
}
```

variables i, n, result = 1 unit each
variable A = n units

# Performance Analysis

**Algorithm-1**

```
Algorithm abc(a,b,c)
{
return a+b+b*c+(a+b-c)/(a+b)+4.0;
}
```

a → 1
b → 1
c → 1
3 units

**Algorithm-2**

```
Algorithm Sum(a, n)
{
s:=0.0;
for i:=1 to n do
      s := s + a[i];
return s;
}
```

i → 1
s → 1
n→ 1
a→ n units
n+3 units

# Performance Analysis

**Algorithm-3**

```
Algorithm RSum(a, n)
{
if (n≤ 0) then
return 0.0;
else
Return RSum(a, n-1)+a[n];
}
```

RSum(a,n) = 1(a[n]) + 1(n) + 1(return) = 3 units

RSum(a,n-1) = 1(a[n-1]) + 1(n) + 1(return)

…………

…………

RSum(a,n-n) = 1(a[n-n]) + 1(n) + 1(return)

Total → ≥ 3(n+1) units

# Performance Analysis

## 2. Time Complexity

The **time complexity** of an algorithm is the amount of computer time it needs to run to complete.

T(P) = compile time + execution time

T(P) = Tp(execution time)

**Step count:**

- For algorithm heading → 0
- For braces → 0
- For expressions → 1
- For any looping statements → number of times the loop is repeating

# Performance Analysis

| Algorithm-1 | |
|---|---|
| Algorithm abc(a,b,c) | → 0 |
| { | → 0 |
| return a+b+b*c+(a+b-c)/(a+b)+4.0; | → 1 |
| } | → 0 |
| | 1 unit |

| Algorithm-2 | |
|---|---|
| Algorithm Sum(a, n) | → 0 |
| { | → 0 |
| s:=0.0; | → 1 |
| for i:=1 to n do | → n+1 |
| s := s + a[i]; | → n |
| return s; | → 1 |
| } | → 0 |
| | 2n+3 units |

# Performance Analysis

```
Algorithm-3
Algorithm RSum(a, n)
{
if (n≤ 0) then
   return 0.0;
else
   return RSum(a, n-1)+a[n];
}
```

$$T(n) = \begin{cases} 2 & \text{if } n = 0 \\ 2 + T(n-1) & \text{if } n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= 2 + T(n-1) \\ &= 2 + (2 + T(n-2)) \\ &= 2 + 2 + T(n-2) = 2*2 + T(n-2) \\ &= 2*2 + (2 + T(n-3)) \\ &= 2*2 + 2 + T(n-3) = 2*3 + T(n-3) \\ &\ldots\ldots\ldots\ldots \\ &\ldots\ldots\ldots\ldots \\ &= 2*n + T(n-n) = 2n + T(0) \end{aligned}$$

$T(n) = 2n+2$ units

# Randomized algorithms

- Makes use of randomizer (random number generator).
- Decisions made in the algorithm depends on the output of the randomizer.
- Output and execution time may very from run to run for the same input.

# Randomized algorithms

**Algorithm** RepeatedElement(a,n)
{
while(true) do
{
i = Random() mod n+1;
j = Random() mod n+1;
if ((i # j) and (a[i] = a[j])) then
        return i;
}
}

**Eg.**

**i = 1, j = 6**
1 # 6 and a[1] # a[6]
**i = 1, j = 5**
1 # 5 and a[1] # a[5]
**i = 2, j = 2**
2 = 2
**i = 4, j = 9**
4 # 9 and a[4] # a[9]
**i = 9, j = 3**
9 # 3 and a[9] # a[3]
**i = 6, j = 7**
6 # 7 and a[6] = a[7]

| a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] | a[10] |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 60 | 60 | 60 | 60 |