

In mathematics and computer science, an algorithm usually means a *step by step procedure designed to perform an operation, and which will lead to the result that solves a recurrent problem.*

Outline of this lecture

- Definition of Algorithm
- Criteria to be satisfied by the Algorithm
- Writing Algorithm
- Recursive Algorithm
- How to measure code execution time?
- How to calculate running time?

Definition of Algorithm Design and Analysis

Design: is the description of the algorithm in (English, pseudo code, or flowchart) and proof of correctness (i.e. Algorithm solves the given problem in all cases).

Analysis: deals with performance (time and storage complexity).

Criteria to be satisfied by the Algorithm

An algorithm must satisfy the following criteria:

1. **Input:** These are the values that are supplied externally to the algorithm (zero or more inputs).

2. **Output**: These are the results that are produced by the algorithm (one or more outputs).
3. **Definiteness**: Each step must be clear and unambiguous.
4. **Finiteness**: The algorithm must terminate after a finite number of steps.
5. **Effectiveness**: Each step must be feasible, i.e. it should be practically possible to perform the step (done in finite time).

Writing Algorithm

- An algorithm consists of **heading** and **body**.
- The heading takes the form
Algorithm Name (parameter-list)
where "**Name**" is the name of procedure and "**parameter-list**" is the list of parameters
- Body has one or more statements within braces { ... }
- An algorithm may or may not return any values

Example1:

//Algorithm to find sum of array values. 'A' is an array of values and 'n' is the size of array

```
Algorithm Sum (A, n)
{
    s = 0;
    for i = 1 to n
        s = s + A[i];
    return s;
}
```

Example2

//Algorithm to find maximum of array values. 'A' is an array of values and 'n' is the size of array

```
Algorithm Max (A, n)
{
    result = A[1];
    for i = 2 to n
        if (A[i] > result) then result = A[i];
    return result;
}
```

Recursive Algorithm

An algorithm said to be recursive if the same algorithm is invoked in the body, there are two types: **Direct** recursive, an **Indirect** recursive.

- **Direct Recursive**

- Algorithm A is said to be direct recursive if it calls itself

- **Indirect Recursive**

- Algorithm A is said to be indirect recursive, if it calls another algorithm which in turn calls A.

- **Every recursive algorithms have two elements**

- **Base Case**

- This statement solves the problem
- Every recursive function must have a base case.

- **General Case**

- Each call reduces the size of the problem.
- The rest of function is known as the general case

Example1: *factorial*

- *Base Case*

- $\text{Factorial}(0) = 1$

- *General Case*

- $\text{Factorial}(n) = n * \text{factorial}(n-1)$

//Algorithm to find factorial of number. 'n' is the number

Algorithm Factorial(n)

```
{  
    if ( n == 0)  
        return 1;  
    else  
        return n * Factorial(n-1);  
}
```

Example2: *Fibonacci series*

- The Fibonacci series for few numbers is as follows
0, 1, 1, 2, 3, 5, 8, 13, 21, 34

- *Base Case*

- $\text{fib}(0) = 0$

- $\text{fib}(1) = 1$

- *General Case*

- $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

```
//Algorithm to generate 'n'th value in Fibonacci series
Algorithm Fib (n)
{
    if (n == 0 or n == 1)
        return n;
    else
        return Fib(n-1) + Fib(n-2);
}
```

How to measure code execution time?

System.Diagnostics.Stopwatch

Provides a set of methods and properties that you can use to accurately measure elapsed time.

```
private void button1_Click(object sender, EventArgs e)
{
    Stopwatch The_Time = new Stopwatch();
    The_Time.Start();
    //    your code here
    The_Time.Stop();
    MessageBox.Show(" the time:" + The_Time.Elapsed);
}
```

How to calculate running time?

Running time dependence on:

1. Single vs. multi processor?
2. Read/Write speed to memory?
3. 32 bits vs. 64 bits?
4. Input size?

Algorithm design and analysis depends on **Input size**

Example: summation of n numbers.



$$sum = \sum_{i=0}^{1024} i = 0 + 1 + 2 + \dots$$

'A' answered after [1024] seconds



$$sum = \frac{n(n+1)}{2} = \frac{1024(1025)}{2}$$

'B' answered after [3] seconds

*We asked again both of them to add numbers
from [0] to [100,000]*



$$\text{sum} = \sum_{i=0}^{100000} i = 0 + 1 + 2 + \dots$$

'A' answered after [100,000] seconds
[27.77] hours



$$\text{sum} = \frac{n(n+1)}{2} = \frac{10^5(10^5+1)}{2}$$

'B' answered after [5] seconds

So, 'B' is better than 'A' .. Right?

If answer is YES, then WHY?

Because person 'B' is faster **than** 'A'.

Easy method.

Correct results.

Less Time.

In Lab:

1. Execute method A and B of summation in page 6.
2. Recursive examples include:
 - Tower of Hanoi puzzle.
 - Inside Squares drawing.

Next Lecture

Performance Analysis

(Time & Space Complexities)