CSE231: Data Structures

Lecture 04

by

Mehedi Hasan Bijoy

ALGORITHM

- An algorithm is a set of well-defined instructions to solve a particular problem.
- It takes a set of input(s) and produces the desired output.

Qualities of a good algorithm:

- Input and output should be defined precisely.
- Each step in the algorithm should be clear and unambiguous.
- Algorithms should be most effective among many different ways to solve a problem.

SUB-ALGORITHM

- A sub-algorithm is a complete and independently defined algorithmic module which is used by some main algorithm or by some other sub-algorithms.
- A sub-algorithm receives values, called arguments, from an originating (calling) algorithm then perform computations and sends back the results to the calling algorithm.
- The sub-algorithm is defined independently.

Variable

Variable is basically nothing but the name of a memory location that we use for storing data.

For using a variable in C, we have to first define it to tell the compiler about its existence so that compiler can allocate the required memory to it.

Syntax:

```
data_type variable_name = value;    // defining single variable
```

or

```
data_type variable_name1, variable_name2;    // defining multiple
```

Variable

- On the basis of scope
  - Local Variable
  - Global Variable
- On the basis of Storage class
  - Static Variable
  - Automatic Variable
  - Extern Variable
  - Register Variable
- Constant Variable
- Pointer Variable

Local Variable

Local variables in C are those variables that are declared inside a function or a block of code. Their scope is limited to the block or function in which they are declared.

```
void function()
{
	int x = 10; // local variable
}

int main() {
	function();
}
```

Global Variable

Global variables in C are those variables that are declared outside the function or a block of code. Their scope is the whole program i.e. we can access the global variable anywhere in the C program after it is declared.

```
int x = 100;

void function1(){
	//…
}
void function2(){
	//…
}


int main() {
	function1();
	function2();
}
```

Data Type

- six basic data types: void, bool, char, int, float, and double

Data Type	Size (in Bytes)
int	4
char	1
float	4
double	8
long int	8
short int	2
signed int	4
unsigned int	4

HOW TO WRITE AN ALGORITHM

Most frequently used 3 ways of writing an algorithm are:

- Text(English)-Like Algorithm
- Flowchart
- Pseudocode

Text-Like Algorithm

Find the largest number between two numbers

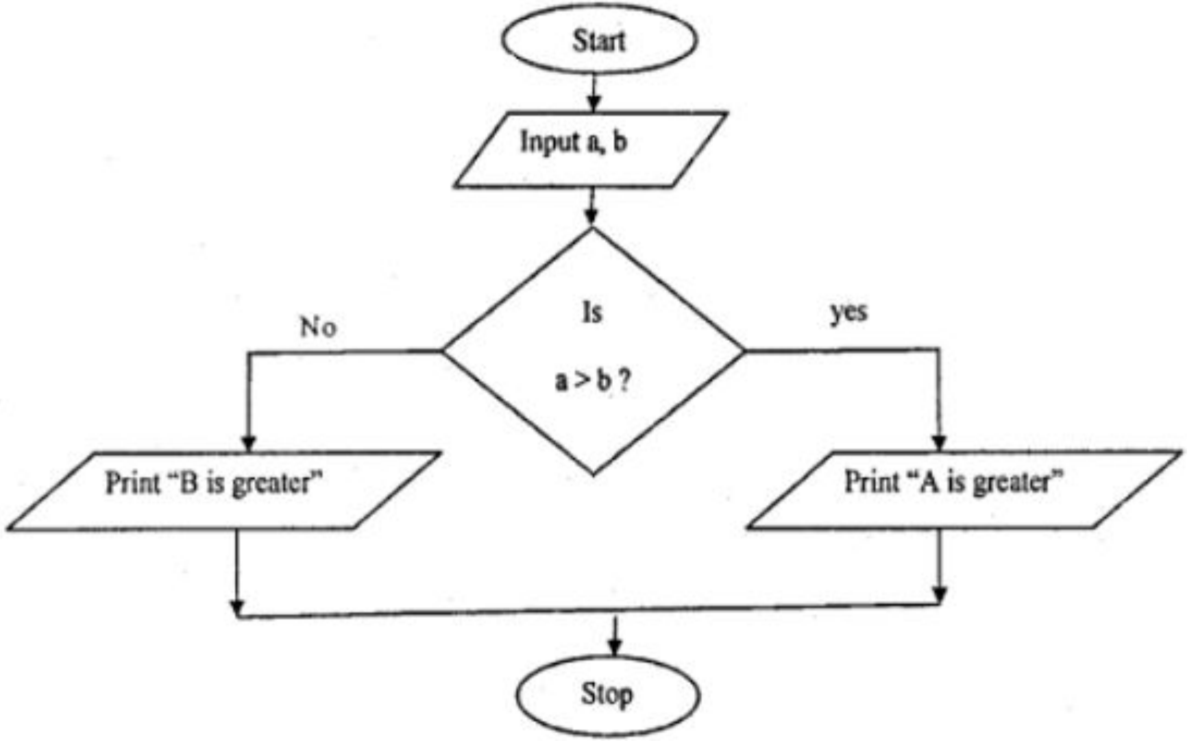
```
Step 1: Start
Step 2: Declare variables a and b.
Step 3: Read variables a and b.
Step 4: If a > b
            Display a is the largest number.
        Else
            Display b is the largest number.
Step 5: Stop
```

Flowchart

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

Flowchart

Find the largest number between two numbers



Pseudocode

Swap two numbers

```
Swap(a, b)
{
    temp = a;
    a = b;
    b = temp;
}
```

```
Swap(a, b)
{
    temp := a;
    a := b;
    b := temp;
}
```

```
Swap(a, b)
{
    temp ← a;
    a ←  b;
    b ← temp;
}
```

Complexity of Algorithms

- The complexity of an algorithm 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.
- In other words, the complexity of an algorithm computes the amount of time and spaces required by an algorithm for an input of size (n).
- The complexity of an algorithm can be divided into two types: The time complexity and the space complexity.

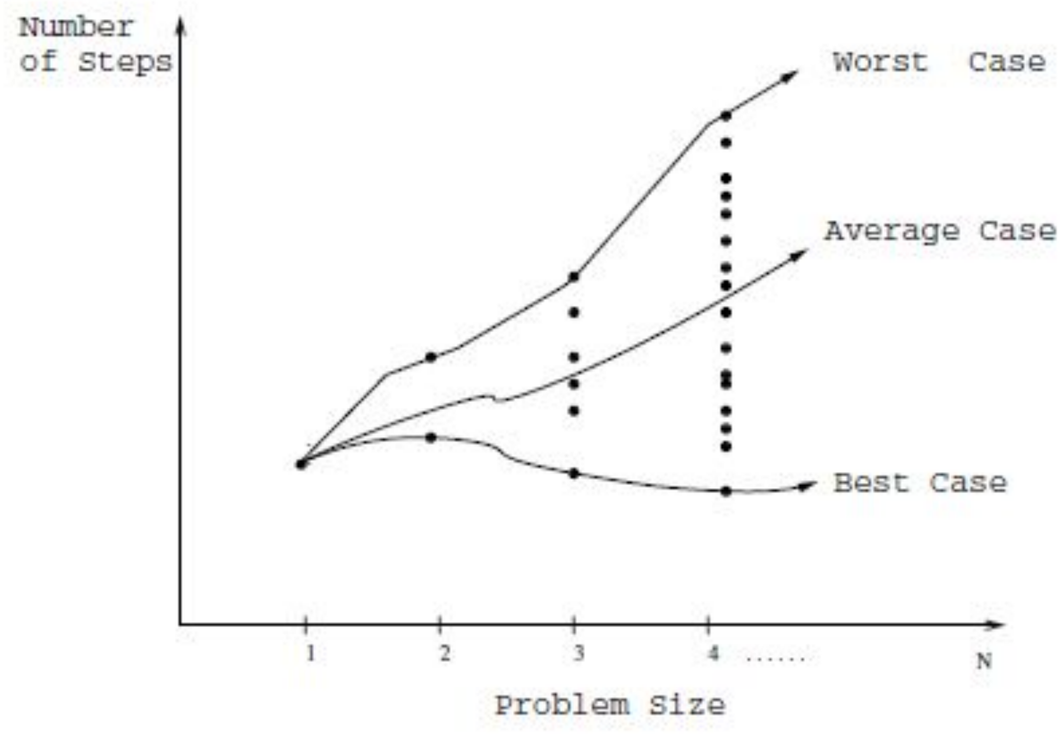
Complexity of Algorithms

`Performing Linear Search`

10	5	2	1	6	3	4	7	9	8

- Worst Case
- Average Case
- Best Case

Complexity of Algorithms



Complexity of Algorithms

- **Worst Case** (Big-Oh Notation)
  - It is the maximum time taken by an algorithm to solve a problem.
  - Worst case occurs when item is the last element in the array data or is not there at all. In either situation $C(n) = n$
  - Most of the time, we do worst-case analyses to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is good information.

Complexity of Algorithms

- **Best Case** (Omega Notation)
  - It is the minimum time taken by an algorithm to solve a problem.
- **Average Case** (Theta Notation)
  - It is in between best case and worst case. Means the average time taken by an algorithm to solve a problem.

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

$$= (1 + 2 + \cdots + n) \cdot \frac{1}{n}$$

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

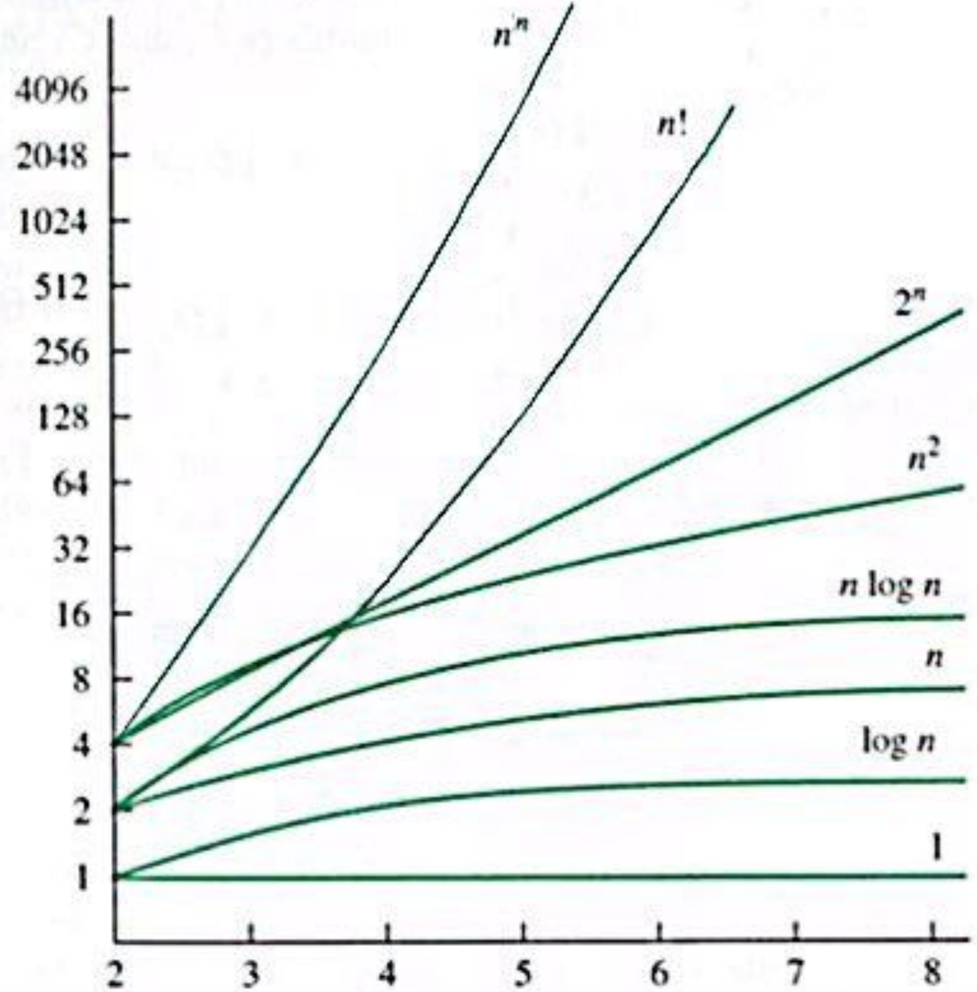
Asymptotic Notations

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < 3^n < n^n$$

$1 < \log n < \sqrt{n} < n$

$< n\log n < n^2 < n^3$

$< 2^n < 3^n < n^n$



Asymptotic Notations

Big-Oh Notation (upper bound)

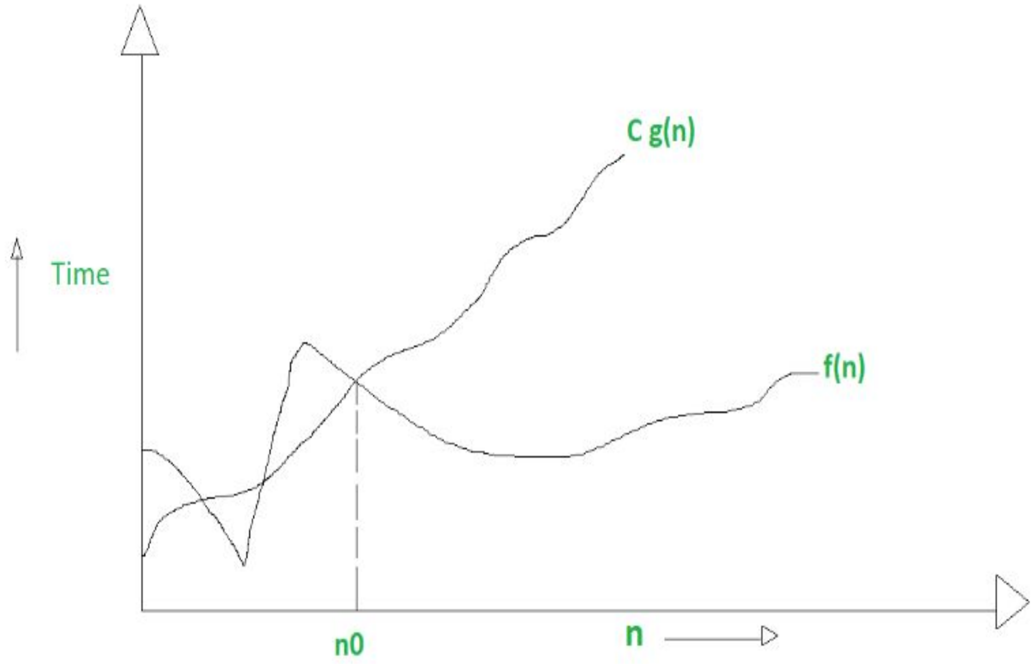
The function f(n) = O(g(n)) iff there exist some positive constants c and n0 such that f(n) <= c.g(n) for all n >= n0

Example:

f(n) = 2n + 3

2n + 3 <= 5n (for all n >= 1)

2n + 3 <= 1n (for all n >= 1) wrong



Asymptotic Notations

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < 3^n < n^n$$

Lower Bound

Average Bound

Upper Bound

Asymptotic Notations

Omega Notation (lower bound)

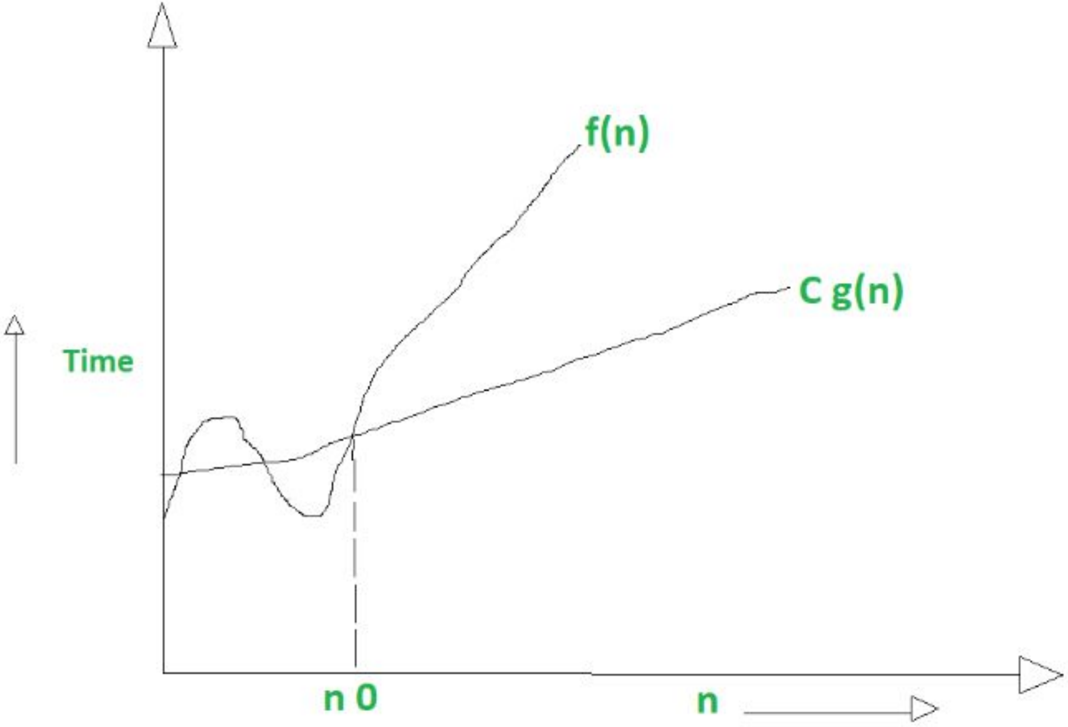
The function f(n) = Ω(g(n)) iff there exist some positive constants c and n0 such that f(n) >= c.g(n) for all n >= n0

Example:

f(n) = 2n + 3

2n + 3 >= 5n (for all n >= 1) wrong

2n + 3 >= 1n (for all n >= 1)



Asymptotic Notations

Theta Notation (average bound)

f(n) is said to be Θ(g(n)) if f(n) is

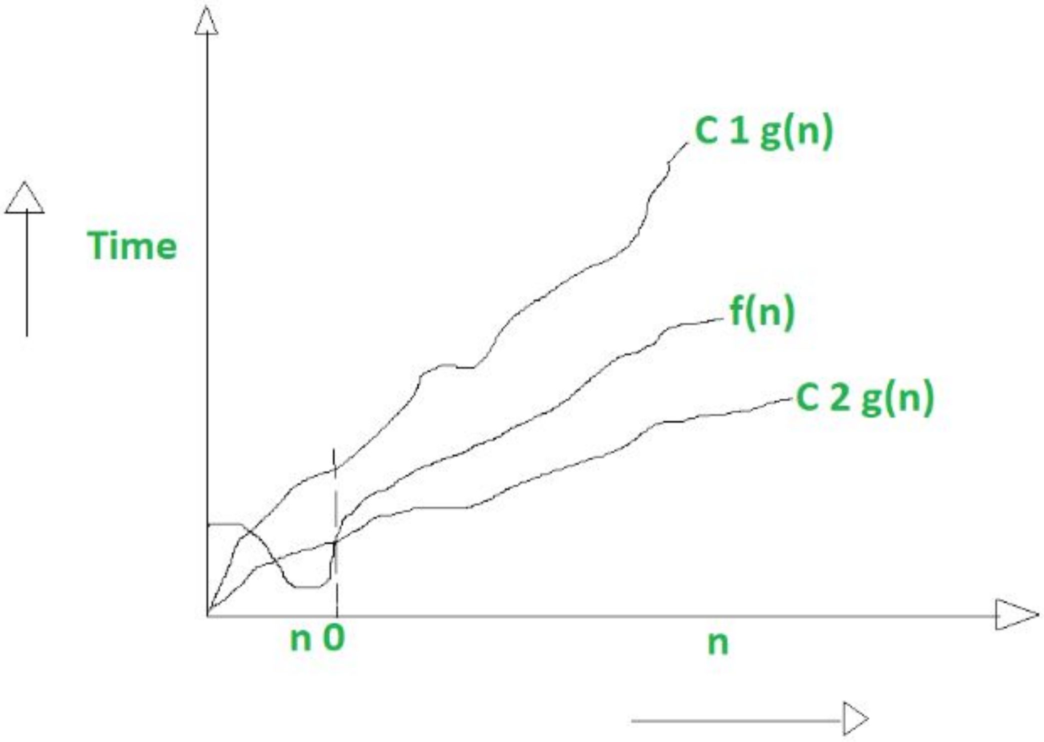
O(g(n)) and f(n) is Ω(g(n)).

Mathematically can be defined as follows:

C2g(n) <= f(n) <= C1g(n) for n >= n0

f(n) = 2n + 3

1n <= 2n + 3 <= 5n



Frequency Count Method

5	3	1	7	2

```
SumOfList(Arr, n)
{
    sum = 0
    for( i = 0;  i < n;  i++ )
    {
        sum = sum + Arr[i]
    }
    return sum;
}
```

Frequency Count Method

```
Add(Arr1, Arr2, row, col)
{
     for( i = 0;  i < row;  i++ )
     {
          for( j = 0;  i < col;  j++ )
          {
               Arr3[i][j] = Arr1[i][j] + Arr2[i][j]
          }
     }
     return sum;
}
```

Thank You!