Unit 2: Algorithm

1. Concept and definition
2. Design of algorithm
3. Characteristic of algorithm
4. Big O notation

Algorithm

An algorithm is a precise specification of a sequence of instructions to be carried out in order to solve a given problem. Each instruction tells what task is to be done. There should be a finite number of instructions in an algorithm and each instruction should be executed in a finite amount of time.

Properties of Algorithms:

1. **Input:** A number of quantities are provided to an algorithm initially before the algorithm begins. These quantities are inputs which are processed by the algorithm.
2. **Definiteness:** Each step must be clear and unambiguous.
3. **Effectiveness:** Each step must be carried out in finite time.
4. **Finiteness:** Algorithms must terminate after finite time or step
5. **Output:** An algorithm must have output.
6. **Correctness:** Correct set of output values must be produced from the each set of inputs.

Write an algorithm to find the greatest number among three numbers:

Step 1: Read three numbers and store them in X, Y and Z
Step 2: Compare X and Y. if X is greater than Y then go to step 5 else step 3
Step 3: Compare Y and Z. if Y is greater than Z then print "Y is greatest" and go to step 7 otherwise go to step 4
Step 4: Print "Z is greatest" and go to step 7
Step 5: Compare X and Z. if X is greater than Z then print "X is greatest" and go to step 7 otherwise go to step 6
Step 6: Print "Z is greatest" and go to step 7
Step 7: Stop

Big Oh (O) notation

When we have only asymptotic upper bound then we use O notation. A function f(x)=O(g(x)) (read as f(x) is big oh of g(x)) iff there exists two positive constants c and x_0 such that for all $x >= x_0$, $f(x) <= c*g(x)$.

The above relation says that g(x) is an upper bound of f(x)
O(1) is used to denote constants.

Example:
$f(x)=5x^3+3x^2+4$ find big oh(O) of f(x)

solution: $f(x)= 5x^3+3x^2+4 \quad <= 5x^3+3x^3+4x^3$ if $x>0$

$<=12x^3$

Therefore, $f(x)<=c.g(x)$, where c=12 and $g(x)=x^3$

Thus by definition of big oh O, $f(x)=O(x^3)$

Big Omega (W) notation:

Big omega notation gives asymptotic lower bound. A function f(x) =W (g(x)) (read as g(x) is big omega of g(x)) iff there exists two positive constants c and x_0 such that for all $x >=x_0$, $0 <= c*g(x) <= f(x)$.

The above relation says that g(x) is a lower bound of f(x).

Big Theta (Q) notation:

When we need asymptotically tight bound then we use notation. A function f(x) = (g(x)) (read as f(x) is big theta of g(x)) iff there exists three positive constants c_1, c_2 and x_0 such that for all $x >= x_o$, $c_1*g(x) <= f(x) <= c_2*g(x)$

The above relation says that f(x) is order of g(x)

Example:

$f(n) = 3n^2 + 4^n + 7$

$g(n) = n^2$, then prove that f(n) = Q(g(n)).

Proof: let us choose c_1, c_2 and n_0 values as 14, 1 and 1 respectively then we can have,

$f(n) <= c_1*g(n)$, $n>=n_0$ as $3n^2 + 4n + 7 <= 14*n^2$, and

$f(n) >= c_2*g(n)$, $n>=n_0$ as $3n^2 + 4n + 7 >= 1*n^2$

for all n >= 1(in both cases).

So $c_2*g(n) <= f(n) <= c_1*g(n)$ is trivial.

Hence f(n) = Q (g(n)).

Example : Fibonacci Numbers

Input: *n*

Output: n_{th} Fibonacci number.

Algorithm: assume *a* as first(previous) and *b* as second(current) numbers

```
fib(n)
{
	a = 0, b= 1, f=1 ;
	for(i = 2 ; i <=n ; i++)
	{
		f = a+b ;
		a=b ;
		b=f ;
	}
```

```
        return f ;
}
```

Efficiency:

Time Complexity: The algorithm above iterates up to n-2 times, so time complexity is O(n).

Space Complexity: The space complexity is constant i.e. O(1).

Example : Bubble sort

Algorithm

```
BubbleSort(A, n)
{
        for(i = 0; i <n-1; i++)
        {
                for(j = 0; j < n-i-1; j++)
                {
                        if(A[j] > A[j+1])
                        {

                                temp = A[j];
                                A[j] = A[j+1];
                                A[j+1] = temp;
                        }
                }
        }
}
```

Time Complexity:

Inner loop executes for (n-1) times when i=0, (n-2) times when i=1 and so on:

Time complexity = (n-1) + (n-2) + (n-3) + ... +2 +1

= $O(n^2)$