3/1/15

UNIT - 1

INTRODUCTION :

→ ALGORITHM : The word algorithm comes from the Persian author Abdullah Jaffar in 9th century who has given the definition of algorithm as follows.

- An algorithm is set of rules for carrying out calculations by the hand or on machine.
- An algorithm is a well defined competitional procedure that takes input and produces output
- An algorithm is a sequence of instructions or steps i,e., inputs to acheive some particular output.
- Any algorithm must follow the criteria /properties.

Input : It generally requires finite number of inputs.

Output : It must produce atleast one output.

Uniqueness : Each instruction should be clear and unambiguess (more clarity of statements)

Finiteness : It must terminate after a finite number of steps

→ ANALYSIS ISSUES OF ALGORITHM

1. what data structures to use ? (list, queue, stacks, trees etc)
2. Is it correct ? (or. all, only, most of the time)
3. How efficient is it ? (asymptotically, fixed or does it depend on the inputs)
4. If there any efficient algorithm? (P = Np or not)

There are four different areas in order to identify the algorithm

1. How to devise algorithm

2. How to validate algorithm

3. How to analyze an algorithm.

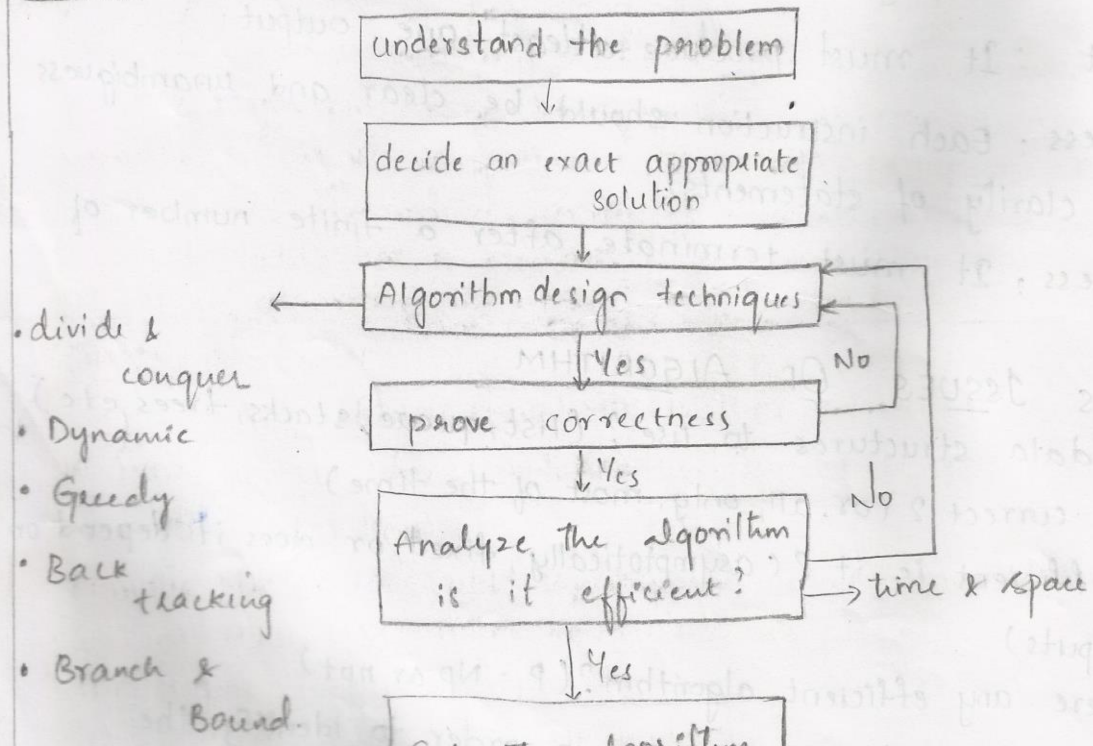
4. How to test the program

Test a program is nothing but performing the debugging and profiling an algorithm.

Validating an algorithm is nothing but cross checking every input it is producing exact output.

Analysis of algorithm is nothing but performance analysis which is done by time & space complexity.

PROCESS OF DESIGN ANALYSIS OF ALGORITHM



PERFORMANCE ANALYSIS

The evaluation can be done in two ways.

- Priori Estimates | Performance Analysis.
- Posteriori Testing | Performance Measurement

Priori	Posteriori
→ The time taken for executing the algorithm is analyzed prior to the execution of algorithm.	→ The execution time taken by an algorithm is evaluated while the algorithm is being executed.
→ It is also known as performance analysis that evaluates whether the code is readable or it performs the desired functions.	→ It is also known as Performance measurement that measures the accuracy of algorithm.
→ It focusses on determining the oder of execution of statement.	→ It focusses on determing the time & space complexity of particular algorithm.
→ It provides approximate values	→ It provides accurate values.
→ It is very expensive (depends upon the system which has been used for execution)	→ It is very less expensive (manually calculated)

The performance of any analysis of an algorithm is calculated using two types of complexities.

- Space Complexity
- Time complexity.

The space complexity of an algorithm is a amount of memory it needs to run for the completions. It consists of two components → fixed part | static (independent of size of I/O, space for code, constant variables, simple variables.

→ variable part | dynamic (dependent on the size of variables declared for a problem to be solved i.e., space for referrenced variables, recursion stack space. It is denoted as

$$\underbrace{S(P)}_{\text{problem}} = \underbrace{C}_{\text{static fixed}} + \underbrace{S_P}_{\text{dynamic variable}} \rightarrow \text{instance characteristics}$$

Note: We concentrate only on measuring the / estimating the Space required for dynamic path.

→ Space Complexity refers to the worst case and denoted as an asymptotic expression in size of input.

O(n): Space algorithm requires a constant amount of memory for input

O(1): Space algorithm requires a constant amount of ~~sio~~ space independent of size of input.

Ex: Algorithm sum (a, n)

```
{
    s : 0.0;
  for i := 1 ton do
    s := s+a[i];
   return s;
}
```

First we must identify the number of variables declared in the algorithm. After identifying, allocate one space for each variable. In above algorithm s, i, n will occupy one space each. Since a is declared as array it requires n words of space that 'a' must hold for n elements to be sumed. The total space occupied by above algorithm is n+3.

→ TIME COMPLEXITY

The time complexity is the amount of the compute time it needs to run for completion i,e., sum of compile time and run time (execution)

Compile time does not depends on instant (variable) characteristics. So we concentrate more on runtime to perform.

The types of time complexity are i) Count method. (ii) frequency method (iii) Asymptotic Notations.

$$t_p(n) = C_a ADD(n) + C_s SUB(n) + C_m MUL(n) + \ldots$$

problem instant Characteristic time complexity for addition

→ COUNT METHOD :

We introduce a new variable count into the program, it is a global variable with initial value 0.

Each time a statement in the original program is executed count is incremented by the step count of that statement.

Ex: Algorithm sum(a, n)

```
{
    s: = 0.0        // count : = count + 1; ①
    for i: = 1 to n do
 2n [ { // count : = count + 1
        s : = s + a[i]; // count : = count + 1
      } // count: = count + 1
    return s; // count : = count + 1
}
```

$\Rightarrow 2n+3$

2) Algorithm Add(a, b, c, m, n)

```
{
    for i: = 1 to m do
 2m [ { count : = count + 1
        for j: = 1 to n do
   2mn [ { count : = count + 1
           c[i, j] = a[i, j] + b[i, j]; count : = count + 1
         }
         count : = count + 1
      }
      count: = count + 1
}
```

$\Rightarrow 2mn + 2m + 1$

3) Algorithm MUL(a, b, c, m, p, n)

```
{
    for(i: = 1 to m do
 2m [ { Count: = count + 1 ;
        for j: = 1 to p do
  3mp [ { count: = count + 1.
          c[i, j] : = 0 ; count: = count + 1
          for k: = 1 to n do
    2mnp [ { count : = count + 1
             c[i, j] : = c[i, j] + a[i, k] * b[k, j]; count: = count + 1
           }
           count : = count + 1
        }
      } count : = count + 1
} count : = count + 1
```

$\Rightarrow 2mnp + 3mp + 2m + 1$

→ FREQUENCY METHOD :

The 2^{nd} method is said to be frequency method which is to be determine the step count of an algorithm is to build a table in which we list the total no. of steps contributed by each statement.

1^{st} column → Statement in which create the algorithm of a given problem.

2^{nd} column → s/e, which indicate steps for execution of the statement

3^{rd} column → is frequency which indicates the total no. of (frequency) times each statement is executed.

4^{th} column → total steps that is "s/e × frequency".

Ex :

1)

statement	s/e	frequency	total steps
Algorithm sum(a,n)	0	0	0
{	0	0	0
s := 0.0 ;	1	1	1
for i := 1 to n do	1	n+1	n+1
s := s+a[i];	1	n	n
return s;	1	1	1
}	0	0	0

total : 2n+3

2)

statement	s/e	frequency	total steps
Algorithm add(a,b,c,m,n)	0	0	0
{	0	0	0
for i := 1 to m do	1	m+1	m+1
{	0	0	0
for j := 1 to n do	1	m(n+1)	m(n+1)
{	0	0	0
c[i,j] := a[i,j] + b[i,j]	1	mn	mn
}	0	0	0
}	0	0	0
}	0	0	0

total : 2mn + 2m + 1

Statement	s/e	frequency	total steps
Algorithm mul(a,b,c,m,p,n)	0		0
{	0		0
for i:=1 to m do	1	m+1	m+1
for j:=1 to p do	1	m(p+1)	m(p+1)
c[i,j]:=0;	1	mp	mp
for k:=1 to n do	1	mp(n+1)	mp(n+1)
c[i,j]:=c[i,j]+a[i,k]*b[k,j];	1	mnp	mnp
}	0		0

total steps: $2mnp+3mp+2m+1$

→ ASYMPTOTIC NOTATIONS

There are used for classification of functions according to the rate of growth of functions. i.e., we try to find the order of growth running time of an algorithm but not the exact running time.

Order of growth:

If algorithms are faster for values of n and slower when n is large then we cannot say these algorithms are good.

This is what we call order of growth.

S.no	Function	Name
1.	1	constant
2.	log n	logarithmic
3.	n	linear
4.	n log n	n log n
5.	n^2	quadratic
6.	n^3	cubic
7.	2^n	exponential
8.	n!	factorial

1) BIG 'O' NOTATION:

The function $f(n) = O\,g(n)$ if and only if there exists positive constant c, n_0 such that $f(n) \leq c * g(n)$ for all values of 'n', $n \geq n_0$.

2) 'Ω' NOTATION:

The function $f(n) = \Omega\,g(n)$ if and only if there exists positive constant c, n_0 such that $f(n) \geq g(n) * c$ for all values of 'n', $n \geq n_0$.

3) 'θ' NOTATION:

The function $f(n) = \theta\,g(n)$ if and only if there exists positive constants n_0, c_1, c_2 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all 'n', $n \geq n_0$.

4) LITTLE 'Oh' NOTATION:

The function $f(n) = o(g(n))$ if and only if

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ $g(n) \rightarrow$ upperbond.

5) LITTLE 'ω' NOTATION

The function $f(n) = \omega\,g(n)$ if and only if

$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$ $f(n) \rightarrow$ upper bond.

→ Problems :-

1) $3n + 3 = O(n)$

$f(n) = O(g(n))$

$f(n) < c * g(n)$

$3n + 3 \leq c * n$

$3n + 3 < 4n$

$n = 1$ $6 \leq 4$ ✗

$n = 2$ $9 \leq 8$ ✗

$n = 3$ $12 \leq 12$ ✓

$n \geq 3$

2) $10n^2+4n+2 = O(n^2)$

$f(n) = O(g(n))$

$f(n) \le c * g(n)$

$10n^2+4n+2 \le c*n^2$

$10n^2+4n+2 \le 11n^2 \quad (n \geq 5)$

$n=1$

$16 \le 11(1)$ ×

$n=2$

$50 \le 22(2)$ ×

$n=3$

$104 \le 33(3)$ ×

$n=4$

$160+16+2 \le 44$

$178 \le 44.4$ ×

$n=5$

$250+20+2 \le 275$

$275 \le 275$ ✓

$\boxed{n \geq 5}$

3) $6*2^n+n^2 = O(2^n)$

$f(n) = O(g(n))$

$f(n) \le c*g(n)$

$6\times 2^n + n^2 \le c \times 2^n$

$6\times 2^n + n^2 \le 7*2^n$ $\boxed{n \geq 1}$

$n=1$

$12+1 \le 7\times 2$

$13 \le 14$ ✓

7) $3n+2 = \Omega(n)$

$f(n) = \Omega(g(n))$

$f(n) \geq c*g(n)$

$3n+2 \geq 3n$

$n=1$

$5 \geq 3$

$n=2$ $\boxed{n \geq 1}$

4) $100n+6 = O(n)$

$f(n) \le g(n) \times c$

$100n+6 \le 101n$

$n=1$

$100+6 \le 101$

$106 \le 101$ ×

$n=2$

$206 \le 202$ ×

$n=3$

$306 \le 303$ ×

$n=4$

$406 \le 404$ × $\boxed{n=6}$

$n=5$

$506 \le 505$ ×

$n=6$

$606 \le 606$ ✓

8) $3n+3 = \Omega(n)$

$3n+3 \geq 3n$

$n=1$ $6 \geq 3$.

$\boxed{n \geq 1}$

9) $100n+6 = \Omega(n)$

$f(n) \geq c*g(n)$

$100n+6 \geq 100n$

$n=1$

$106 \geq 100$ ✓

$\boxed{n \geq 1}$

According to definition of Ω, the no value should be greater than zero always

10) $3n+3 = \Theta(n)$

$C_1*g(n) \leq f(n) \leq C_2*g(n)$

$\cancel{3n+3}$ $n * C_1 \leq 3n+3 \leq C_2*n$

$3n \leq 3n+3 \leq 4n$

$n=1$

$3 \leq 6 \leq 4$

$n=2$

$6 \leq 9 \leq 8$

$n=3$

$9 \leq 12 \leq 12$ ✓

$\boxed{n \geq 3}$

11) $10n^2+4n+2 = \Theta(n^2)$

$c_1 * g(n) \leq f(n) \leq c_2*g(n)$

$n^2 * c_1 \leq 10n^2+4n+2 \leq n^2 * c_2$

$10n^2 \leq 10n^2+4n+2 \leq 11n^2$

$n=1$

$10 \leq 16 \leq 11$ ×

$n=2$

$40 \leq 50 \leq 44$ ×

$n=3$

$90 \leq 104 \leq 99$ ×

$n=4$

$160 \leq 178 \leq 176$ ×

$n=5$

$250 \leq 272 \leq 275$ ✓

$\boxed{n \geq 5}$

→ FINDING TIME COMPLEXITY USING NOTATIONS.

Statement	S/e	Frequency	Total steps
Algorithm sum(a,n)	0	0	0
{	0	0	0
s := 0.0 ;	1	1	1
for i := 1 to n do	~~n+~~1	n+1	n+1
s = s+a[i];	~~n~~1	n	n
return s;	1	1	1
}	0	0	0

$\theta(1) + \theta(n+1) + \theta(n) + \theta(1)$

$\Rightarrow \theta(n+1) + \theta(n)$ ($\because$ $\theta(1)$ = constant according to asymptotic notations constants are neglected)

$\Rightarrow \theta(n) + \theta(1) + \theta(n)$

$\Rightarrow 2\theta(n)$

$\Rightarrow \theta(n)$

2)

Statement	s/e	Frequency	total steps
Algorithm Add(a,b,c,m,n)	0	0	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
}			

$\theta(m+1) + \theta(mn+m) + \theta(mn)$

$\Rightarrow \theta(m) + \underline{\theta(1)} + \theta(mn) + \theta(m) + \theta(mn)$

$\Rightarrow 2\theta(m) + 2\theta(mn)$

$\Rightarrow \theta(m) + \theta(mn)$

$\Rightarrow \theta(mn)$ $\because n < mn$

3)

Statement	s/e	Frequency	total steps
Algorithm mul(a,b,c,m,n)	0	0	0
{	0	0	0
for i := 1 to m do	1	m+1	m+1
for j := 1 to p do	1	m(p+1)	m(p+1)
c[i,j] := 0.0;	1	1	1
for k := 1 to n do	1	mp(n+1)	mp(n+1)
c[i,j] = c[i,j]+a[i,k]*b[k,j];	1	mnp	mnp
}	0	0	

$\theta(m+1) + \theta(mp+m) + \theta(mnp+mp) + \theta(mnp) + \underline{\theta(1)}$

$\Rightarrow \theta(m) + \theta(1) + \theta(mp) + \theta(m) + \theta(mnp) + \theta(mp) + \theta(mnp)$ ($\because \theta(1) = $ constant)

$\Rightarrow 2\theta(m) + 2\theta(mp) + 2\theta(mnp)$

$\Rightarrow \theta(m) + \theta(mp) + \theta(mnp)$

$\Rightarrow \underline{\theta(mnp)}$

4)

Statement	s/e	frequency	total steps
Algorithm mul (a,b,c,m,n)	0	0	0
{	0	0	0
for i := 1 to ~~m~~ n do	1	n+1	n+1
for j := 1 to n do	1	n(n+1)	n^2+n
C[i,j] := 0.0;	1	1	1
for k := 1 to n do	1	$n^2(n+1)$	n^3+n^2
C[i,j] := C[i,j] + a[i,k] * b[k,j]	1	1	1
}			

$\theta(n+1) + \theta(n^2+n) + \theta(n^3+n^2) + \theta(1) + \theta(1)$

$\Rightarrow \theta(n+1) + \theta(n^2+n) + \theta(n^3+n^2)$ ($\because \theta(1) \rightarrow$ constant)

$\Rightarrow \theta(n) + \theta(n^2) + \theta(n) + \theta(n^3) + \theta(n^2) + \theta(1)$

$\Rightarrow 2\theta(n) + 2\theta(n^2) + \theta(n^3)$

$\Rightarrow \theta(n) + \theta(n^2) + \theta(n^3)$

$\Rightarrow \underline{\theta(n^3)}$

DIVIDE AND CONQUER :-