21/11/18

Design And Analysis of Algorithm

Introduction To Algorithm.

Definition:-

* An algorithm is nothing but step by step procedure for solving computational problems.

(or)

* An algorithm is a finite set of instructions for solving (or) for accomplish a particular task.

Differences between algorithm and program:-

ALGORITHM	PROGRAM
1) It is a step by step process for solving computational problems.	1) A program is nothing but set of instructions (or) executable code.
2) An algorithm is designed by domain knowledge (designer)	2) The program can be implemented by specific programming
3) An algorithm is done at design time	3) program is done at Implementation time.
4) Any language eg: Mathematical notation, General english statements.	4) Any programming languages eg: C, C++, java, Ada, .NET
5) After completion of your algorithm you have to analyze based on some criteria's, such as time complexity & space complexity,	5) After completion of writting your program you have to do test your program

*3M characteristics of an algorithm:-

An algorithm characteristics are:

1. Input :- Externally we have to supply 0 (or) 1 quantity.
2. Output :- After supplying of input we have to produce atleast one quantity.
3. Finiteness :- whenever trace any algorithm, the

algorithm must be terminate (or) end at some point of time.

4) Definiteness:- we have to define clear and unambig. ous statements.

5) Effectiveness:- An algorithm allows only necessary statements need not to allow any un-necessary statements.

How to write an algorithm:- Heading & Body

1) Algorithm swap(a,b) ← heading
{ ← Body
temp = a;
a = b;
b = temp;
}

2) Algorithm swap (a,b)
BEGIN
temp = a;
a = b;
b = temp;
END

3) Algorithm swap (a,b)
{
temp := a;
a := b;
b := temp;
}

4) Algorithm swap (a,b)
BEGIN
temp ← a;
a ← b;
b ← temp;
END.

22/11/18

How to analyze an Algorithm:-

* Based on some criteria you have to analyze an algorithm.
* The criteria's are 1) Time complexity
2) Space complexity
3) Network consumption
4) power consumption
5) cpu registers

* Time complexity is done by using Frequency count (FC).
*2M

Frequency count (FC):-

The no. of times the statement will be executed by a compiler is known as Frequency count.

every simple stmt will take one unit of time

Write an algorithm for swapping of two numbers.

```
Algorithm swap(a,b)  ->0
{                    ->0
temp = a;            ->1
  a = b;             ->1
  b = temp;          ->1
}                    ->0
```

frequency count

$f(n) = 0+0+1+1+1+0$

$= 3$

Hence the time complexity is

Space:-

a -> 1

b -> 1

Temp -> 1

$S(n) = 1+1+1 = 3$ words.

$S(n) = O(1)$

```
Algorithm sum(A,n)   ->0
{                    ->0
 S= 0;               ->1
 for(i=0; i<=n; i++) ->n+1      i=0 i=1 i=2 i=3 i=4 i=5
  {                  ->0
  S=S+A[i];          ->n
  }                  ->0
 return S;           ->1
}                    ->0
```

$f(n) = 0+0+1+n+1+0+n+0+1+0$

$= 2n+3$

$= O(n)$

Space:-

A -> n

n -> 1

S -> 1

i -> 1

$n+3$

$O(n)$

```
Algorithm  Add(A,B,n)          ---> 0
                               ---> 0
{
for (i=0; i<n; i++)            ---> n+1
{                              ---> 0
 for (j=0; j<n; j++)           ---> n(n+1)
 {                             ---> 0
  c[i,j]= A[i,j] + B[i,j];     ---> n(n+1)   (n x n)
 }}                            ---> 0
}                              ---> 0
                               ---> 0
```

time

$f(n) = n+1+n^2+n+n^2$

$= 2n^2+2n+1$

$f(n) = O(n^2)$

Space

A → n×n

B → n×n

C → n×n

n → 1

i → 1

j → 1

$S(n) = 3n^2+3$

$= O(n^2)$

Write an algorithm to find multiplication of two matrix by using frequency count method.

```
Algorithm  mul(A,B,n)
{
for (i=0; i<n; i++)            ---> n+1
{
for (j=0; j<n; j++)            ---> n(n+1)   = n^2+n
{
 C(i,j)=0;                     ---> n x n    = n^2
for(k=0; k<n; k++)             ---> n x n x (n+1) = n^3+n^2
{
 C(i,j)= C(i,j) + A(i,k) * B(k,j)   ---> n x n x (n+1) = n^3+n^2
}}}
}
```

time $f(n) = 2n^3+3n^2+2n+1$

$f(n) = O(n^3)$

space:-

A → n×n = n^2

B → n^2

C → n^2

n → 1

i → 1

j → 1

k → 1

$S(n) = 3n^2+4$

$S(n) = O(n^2)$

How to Test an Algorithm:-

*After complition of analysis phase Now immediately go for Testing process.

*By using debugging and profiling you can test our algorithm (or) program

Debugging:-

*Debugging is nothing but the process of checking an algorithm and finally get genune results.

*Once the debugging process is completed, the algorithm is correct.

*2M profiling:-

The process of calculating Time & space complexities is known as profiling.

** Pseudo code for expressing an algorithms:- (-> set of instructions.) (; -> delimiter)

Algorithm is basically sequence of instructions writting general english statements

In general algorithm can be represented in two ways

1) Flowchart representation

2) pseudo code

Flow-chart Representation:-

*The graphical representation of an algorithm is known as flowchart representation.

Pseudo code:-

*Pseudo code is sequence of instructions based on some programming Constraints.

*@ algorithm is a procedure consisting of heading & body. heading consist name of the procedure and body consisting of list of parameters.

eg:- Algorithm Algorithm-name (list of parameters)

*The Begining and end of blocks should be indicated by { or then }, BEGIN and END.

* The delimiters (;) is used at end of the statement.
* The identifier should begin by letters and not by digit.
* Using Assignment operator (:=) an assignment statement can be given

ex:- variable := Exp;

* The algorithm allows operators like Relational, Arithmetic operators and boolean operator.
* An algorithm allows arrays may be one dimensional and Two dimensional
* Algorithms allows looping statements like while, do-while, for

while (condition)
{
}

* Algorithm allows conditional statements like if, else, if else-if
* The algorithm allows repeat and until statement
* Algorithm allows break statement, break statement is used to exit from Inner loop.
* The return statement is used to return control from one point to another.

Performance Analysis:-

* The efficiency of an Algorithm is declared by measuring the performance of an algorithm.
* We can measure the performance of an algorithm computing by time & space requirement.
* We can measure the performance of an algorithm we can follow 1) space complexity
2) Time complexity

Space complexity:-

The amount of space (or) memory require an algorithm to run is known as space complexity.

* The space complexity follows two factors 1) constant variable
2) Instance variable.

* The space complexity $S(p) = C + Sp$

where, $S(p)$ is a space complexity

C is constant variable

Sp is instance variable.

* C is always fixed in size

* Sp is instance variable is always size may be go up and down (increment (or) decrement)

* Space complexity is always $\boxed{S(p) \geq C + Sp}$

Ex:- Algorithm add(a,b,c)

```
{
return a+b+c;
}
```

Here C=3, Sp=0

a=1

b=1

c=1

3

→Hence S(n) = C+Sp

= 3+0

S(n) = 3

$\boxed{S(p) \geq 3}$

* Multiplication of two matrices using frequency count; method.

Algorithm mul(a,b,c,n)

```
{
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
c(i,j)=0;
for(k=0; k<n; k++)
{
c(i,j)= c[i,j) + A(i,k)*B(k,j)
}}}
```

Space complexity:-

S(p) = C+Sp

= 4+9

Time complexity:-

* The amount of time required an algorithm to run to complition is known as Time complexity.

* Types of computing Times

1) compile Time

2) Run Time

* Time complexity allows only runtime (or) execution Time.

* Time complexity can be calculated interms of FC.

* Time complexity is mainly classified into 5 types based on frequency count
1) constant
2) Linear
3) Quadratic
4) Cubic
5) Logarithmic

Constant:- The statement will be executed by the compile only once.

ex:- statement;

Linear:- The statement will be executed by the compiler in n no.of times

ex:- for i=1 to n do i=i+1

Statement;

Quadratic:- The statement will be executed by the compiler 'nxn' times that is n^2 times.

ex:- for i=1 to n do i=i+1;

for j=1 to n do j=j+1

Statement;

cubic:- the statement will be executed by the compiler nxnxn times that is n^3 times, such kind of time complexity is known as cubic. time complexity

ex:- for i=1 to n do i=i+1

for j=1 to n do j=j+1

for k=1 to n do k=k+1

statement;

Logarithmic:- for each and every time the work area is divided into half of its work area.

```
while(low ≤ high)
{
 mid := (low + high)/2;
 if (key < a[mid])
 high = mid-1;
  if (key > a[mid])
    low=mid+1;
  else
    return mid;
 }
```

Time complexity can be represented in 3 ways 1) Best case 2) Worst case 3) Average case.

'Ω'

Best case:- If an algorithm takes (or) require minimum time to complete an algorithm for a set of specific inputs is known as Best case

eg:- In the linear search we have to find out first element in the given list.

'O'

Worst Case:- If an algorithm ~~takes~~ takes maximum time to complete an algorithm for a set of specific inputs is known as Worst case.

eg:- In the given linear search we have to find out the last element in the given set of elements.

'θ'

Average Case:- If an algorithm takes average time to complete an algorithm for a set of specific inputs is known as Average case.

eg:- In the given linear search we have to find out the middle of the element in the given set of elements.

Asymptotic Notations:-

Asymptotic notations are used to represent time complexity.

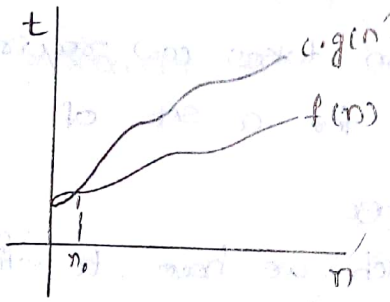
Time complexity may be Best, Worst, Average case

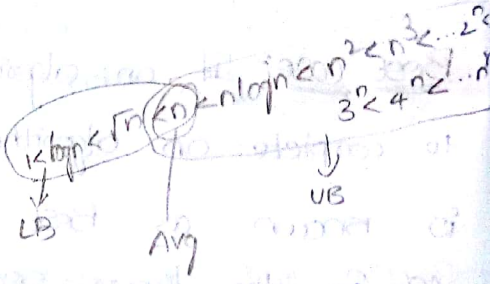
To represent best, worst and average time complexities we can use the notations like Ω, O and theta (Θ)

Big-oh "O" Notation:-

* Big-oh notation is indicated by "O".
* Big-oh notation is used to represent worst case.

Definition:- The function $f(n) = O(g(n))$ iff and only if there exist ($\exists$) constant c and positive integer n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$





Here f(n), g(n) are two non negative functions

c is constant

n, n_0 are two positive integers.

Eg: $f(n) = 3n+2$ & $g(n) = n$ prove that $f(n) = O(g(n))$

Given $f(n) = 3n+2$

$g(n) = n$

$f(n) \leq c \cdot g(n)$

for $c=1$, $n=1$

$3n+2 \leq c \cdot n$

$3(1)+2 \leq 1 \cdot 1$

$5 \leq 1$ False

$2n+2 \leq n^2$

$2(3)+2 \leq 3^2$

$8 \leq 9$

for $c=1$, $n=2$

$3(2)+2 \leq 1 \cdot 2$

$8 \leq 2$ False

for $c=1$, $n=3$

$3(3)+2 \leq 1 \cdot 3$

$11 \leq 3$ False

if $c=2$, $n=1$

$5 \leq 2$ false

if $c=2$, $n=2$

$8 \leq 4$ false

if $c=3$, $n=1$

$5 \leq 3$ false

if $c=3$, $n=2$

$8 \leq 6$ False

if $c=4$, $n=1$

$5 \leq 4$ false

if $c=4$, $n=2$

$8 \leq 8$ True

if $c=5$, $n=1$

$5 \leq 5$ True

if $c=5$, $n=2$

$8 \leq 10$ True

② $2n+2 \leq n^2$

if $n=1$, $c=1$

$2+2 \leq 1$ False

if $c=1$, $n=2$

$4+2 \leq 4$

$6 \leq 4$ false

if $c=1$, $n=3$

$6+2 \leq 3^2$

$8 \leq 9$ True

if $c=1$, $n=4$

$8+2 \leq 4^2$

$10 \leq 16$ True

hence it is proved $n=1$, $c=5$, $n_0=1$

Omega (Ω) Notation:

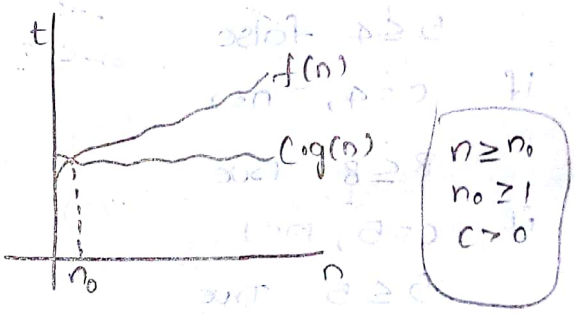
* Omega is indicated by Ω.
* It is used to represent best case time complexity.
* It takes minimum amount of time to run an algorithm hence it is called lower boundary.

Definition:

* The function $f(n) = \Omega(g(n))$ if and only if $\exists$ constant C, possitive integers n, n_0 such that $f(n)$ is always

$$f(n) \geq C * g(n) \quad \text{for all} \quad n \geq n_0, \; n_0 \geq 1 \text{ and } C > 0.$$

Eg: $f(n) = 2n+2$, $g(n) = n$ poove that $f(n) = \Omega(g(n))$



Given $f(n) = 2n+2$

$g(n) = n$

$f(n) \geq C \cdot g(n)$

for $C = 1, \; n = 1$

$2n+2 \geq C \cdot n$

$2(1)+2 \geq 1 \cdot 1$

$4 \geq 1$ True

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

where $C = 1, \; n = 1, \; n_0 = 1$

Eg:2 $f(n) = 3n+2$, $g(n) = n$ PT $f(n) = \Omega(g(n))$

Given $f(n) = 3n+2$

$g(n) = n$

$f(n) \geq C \cdot g(n)$

for $C = 1, \; n = 1$

$3n+2 \geq C \cdot n$

$3(1)+2 \geq 1 \cdot 1$

$5 \geq 1$ (True)

Hence it is proved $f(n) = \Omega(g(n))$

where $C = 1, \; n = 1, \; n_0 = 1$

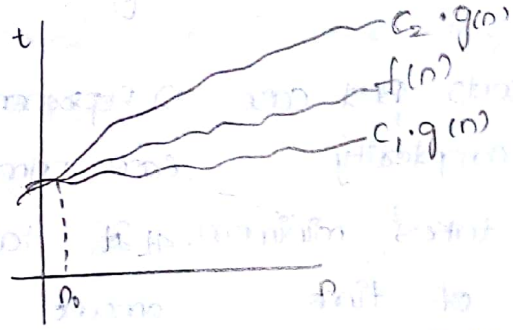
Theta (θ) Notation:-

* Theta is indicated by 'θ'
* It is used to represent Average case time complexity
* It takes Average time to run an algorithm hence it is called average bound.

Definition:-

The function $f(n) = \theta(g(n))$ if and only if there exist ∃ constants c_1 and c_2, positive integers n, n_0 such that

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all $n \geq n_0$, $n_0 \geq 1, c > 0$



ex:- $f(n) = 2n+2$, $g(n) = n$ P.T $f(n) = \theta(g(n))$

Given $f(n) = 2n+2$

$g(n) = n$

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

for $c=1$, $n=1$, $c_2 = 4$

$1 * n \leq 2n+2 \leq c_2 * n$

$1 * 1 \leq 2(1) + 2 \leq c_2 * 1$

$1 \leq 4 \leq 4$

Hence it is proved $f(n) = \theta(g(n))$

where $c_1 = 1$, $n = 1$, $c_2 = 4$, $n_0 = 1$.

Differences and comparison b/w "Big-oh", "Big-omega", "Big-theta"

Big-oh	Omega	Theta
1) The function $f(n) = O(g(n))$ if and only if their exist ($\exists$) constant c and +ve integer n_0 such that $f(n) \le c \cdot g(n)$ $\forall$ $n \ge n_0$	1) The function $f(n) = \Omega(g(n))$ if and only if $\exists$ constant c, +ve integers n, n_0 such that	1) The function $f(n) = \theta(g(n))$ if and only if $\exists$ constants c_1 & c_2, positive integers n, n_0 such that $c_1 * g(n) \le f(n) \le c_2 * g(n)$ $\forall$ $n \ge n_0$, $n_0 \ge 1$, $c > 0$
2) It is indicated by "O"	2) It is indicated by "Ω"	2) It is indicated by "θ"
3) Represents Worst case Time Complexity	3) Represents Best case Time complexity	3) Represents Average case Time complexity
4) It takes maximum amount of time	4) It takes minimum amount of time	4) It takes Average amount of time
5) It is an upper Bound	5) It is an lower Bound	5) It is an Average Bound.
6) $f(n) = O(g(n))$	6) $f(n) = \Omega(g(n))$	6) $f(n) = \theta(g(n))$
7) $f(n) \le c \cdot g(n)$	7) $f(n) \ge c \cdot g(n)$	7) $c_1 * g(n) \le f(n) \le c_2 * g(n)$
8) The constant c i.e $c > 0$	8) The constant c i.e $c > 0$	8) The constant c i.e $c > 0$
9) The +ve integers (n, n_0) $n \ge n_0$ $n_0 \ge 1$	9) The +ve integers n and n_0 i.e $n \ge n_0$, $n_0 \ge 1$	9) The +ve integers n & n_0 i.e $n \ge 0$; $n_0 \ge 1$.

* Amortized Analysis:-

Definition:- Finding Average running time per operation over the Worst case sequence of operations.

* Amortized analysis does not allows random numbers.
* Amortized analysis and average case analysis are diff
* In case of average analysis we can find averaging

all the inputs, but whereas amortized analysis we are averaging over a sequence of operations.

* To compute Amortized analysis we can follow the following techniques.

1) AGGREGATE Method
2) Accounting Method
3) potential Method

Aggregate Method:-

* Aggregate method is used to find the total cost per 'n' no.of worst case sequence of operations.

* Aggregate Method is equal to $\frac{T(n)}{n}$

where, $T(n)$ = Total cost

n = no.of operations

Accounting Method:-

* By using accounting method we can find individual cost per an operation.

* finally we fand total cost i.e $\frac{T(n)}{n}$

potential Method:-

* potential Method is like a accounting method but operations depend upon its Lower cases and Upper cases

Note:-

1. Upper bound, Lower bound & Average bound can be represented by the following statement.

