ch-1 :

PAGE NO.: 01

* Introduction to Algorithms.

* Algorithm :- It is the finite sequence of operations / instructions which transform the given input to correct output.

* properties of algorithm :-

→ Input from a specified set,
→ output from a specified set (solution),
→ definiteness of every step in the computation.
→ correctness of output for every possible input.
→ finiteness of the number of calculation steps.
→ Effectiveness of each calculation step
→ Generality for a class of problems.

* Efficiency of algorithms :-

→ which algorithm needs to be chosen when more than one algorithm available ?

* Three approaches :-

(1). Empirical :- programming all the techniques and trying (posteriori) :- them at different instance.

(2). Theoretical :- determining mathematically the quantity of (priori). resources as a function of the size instance. Resource like computing time / storage space

(3). Hybrid :- Algo's efficiency is determined theoriti- approach cally and required numerical parameters are determined Empirically.

Scanned with CamScanner

* Ch-2 Performance analysis.

PAGE NO.: 19 01 '23

* Two criteria are used to judge algorithms :-

(1). Space complexity :- Space Complexity of the program is the amount of the memory it needs to run to completion.

(2). Time complexity :- Time complexity of the program is the amount of CPU time it needs to run to completion.

*(1). Space Complexity : S(P)

⇒ Components of space Complexity :-

(i) Instruction space :-

~ space needed to store the compiled version of program instructions.

~ It depends on...

- The compiler used to compile the program.
  → Ex :- a+b+(b*c)+(a+b-c)/(a+b)/4.
- The compiler options in Effect.
  → Ex :- overlay option.
- The target computer :
  → Ex :- floating point hardware.

Scanned with CamScanner

(ii) Data space :-

→ space needed by constants and simple variables.
→ space needed by dynamically allocated objects (arrays, class instance, etc...)

(iii) Environmental stack space :-

→ The return address
→ All local variables.
→ All formal parameters.
→ Recursion stack space.
  ↳ • The space needed by local variables and parameters.
  • The maximum depth of recursion.

* Memory space S(p) Needed by a program p, consists of two components.

→ Fixed part :- Needed for instruction space (bytecode), simple variable space, constants space etc..
  Ex :- C.

→ A variable part :- Dependent on a particular instance of Input and output data.
  → S_p (instance).

$$* \; S(p) := C + S_p(\text{instance})$$

Scanned with CamScanner

Ex:-1). Algorithm abc (a,b,c)
{
return a+b+b*c+(a+b-c)/(a+b)+4.0;
}

→ For every instance 3 computer words required to store variables: a, b and c.

→ Therefore $S_p() = 3$.
$S(p) = 3$.

Ex:-2). Algorithm sum (a[], n)
{
s: 0.0;
for i=1 to n do
s: s+a[i];
return s;
}

→ Every instance needs to store array a[] & n.
- space needed to store n = 1 word.
- space needed to store a[] = n floating points (words can at least n words)
- space needed to store i and s = 2 words

→ $S_p(n) = (n+3)$.
Hence $S(p) = (n+3)$

Scanned with CamScanner

Ex:3). Algorithm Rsum (a[], n)
{
if (n>0)
return Rsum (a, n-1) + a[n-1];
return 0;
}

→ It is recursive function:

→ Every Recursive function call needs to store address of a[], the value of n & return address.

- space needed to store n = 1 word.
- space needed to store the address of a[] = 1 word.
- space needed to store return address = 1 words
- Depth of Recursion = n+1
- space needed to store a[] = n words.

→ Sp(n) = n.
→ Hence S(P) = 3(n+1)+n.

#NOTE:- In simple function we can't call address many times, so we can't store this address and not counting it's word in solution.

In Recursive function we need to call address many times, so we store this address and counting it's space in solution.

Scanned with CamScanner

Ex:- 4). Algorithm Fact (int n)
{
if (n<=1)
return 1;
else return (n * Fact (n-1));
}

→ It is also Recursive Function.

→ Every Recursive function call needs to store the value of n & return address.

- Space needed to store n=1 word.
- Space needed to store return address = 1 words.
- Depth of Recursion = n.

→ Hence S(p) = 2(n).

*(2). Time Complexity :- T(P)

*→ Time Required T(P) to run a program p also Consists of two Components :

→ fixed part :- Compile time which is independent of the problem Enstance.
Ex:- C

→ Variable part :- Run time which depends on the problem instance.
→ t(p) instance.

{* T(P) :- C+tp (instance)}

Scanned with CamScanner

* How To Measure T(P)?

1. Measure Experimentally, using a "stop watch".
→ T(p) obtained in sec, msecs.

2. Count Number of major operations / instructions:-
→ Identify one or more major operations and determine how many times each is executed.

→ Ignore the other minor operations / instructions.

Ex:- In problem we have for loop so we consider it as major operation and count T(p). we consider condition, return value etc... and Ignore minor operations.

3. Count program steps:~> T(p) obtained as step count.

→ We count all the steps of the program from first step to last step.

NOTE:- Fixed part is usually ignored; only the variable part tp() is measured.

Ex:-1) Algorithm IndexofMax (int a[], int n)

```
{
    int index = 0;
    for (int i=1; i<n; i++)
    if (a[index] < a[i])
      index = i;
    return index;
}
```

Scanned with CamScanner

Soln:- → Comparison can be considered as major operation in above algorithm.

→ Total number of comparison = max{n-1, 0}

→ Hence, T(n) = n-1

Ex:- 2 Algorithm polyEval (int coeff[], int n, int x)
{
int y=1, value = coeff[0];
for (int i=1; i<n; i++)
{
y = y * x;
value = value + y * coeff[i];
}
}

Soln:- → Here, additions and multiplications can be identified as major operations.

- Total Number of additions = n-1
- Total number of multiplications = 2(n-1)

- Hence, total number of operations = (n-1)+2(n-1)
- T(n) = 3n-3.

Scanned with CamScanner

* Step Count.

- Consider all Executable statements of an algorithm.
- what is program step?

- a+b+b*c+(a+b)/(a-b) → one step;
- Comments → zero steps;
- while (<Expr>) do → step count equal to the number of times <Expr> is Executed.
- for i = <Expr> to <Expr1> do → step count equal to number of times <Expr1> is checked.

Ex:-1).

Statements	S/E	freq.	Total.
1. Algorithm sum(a[],n)	0	-	0
2. {	0	-	0
3. s = 0.0 ;	1	1	1
4. for i=1 to n do	1	n+1	n+1
5. s = s+a[i];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
			2n+3

→ space complexity = 3+n.

Scanned with CamScanner

Ex:- 2).

	Statements	S/E	freq.	Total
1	Algorithm sum(a[], n, m)	0	-	0
2.	{	0	-	0
3.	for i = 1 to n do :	1	n+1	n+1
4.	for j = 1 to m do	1	n(m+1)	n(m+1)
5.	s = s + a[i][j] ;	1	nm	nm
6.	return s ;	1	1	1
7.	}	0	-	0
				2nm+2n+2

→ Space complexity = mn+5.

Ex:- 3).

	Statements	S/E	freq.	Total.
1.	Algo Transpose (a[], row)	0		0
2.	{	0		0
3.	for (i=0 ; i<row ; i++)	1	row+1	row+1
4.	for (j = i+1 ; j<row ; j++)	1	row(row+1)/2 - "	
5.	swap (a[i][j] , a[j][i] ;	1	row(row-1)/2 - "	
6.	}	0		0
				row^2+row+1

→ 5th line $\sum_{n=1}^{n} \frac{n(n+1)}{2}$

$$\sum_{n=1}^{n-1} \frac{(n-1)(n-1+1)}{2}$$

$$\leq \frac{n(n-1)}{2}$$

Scanned with CamScanner

Ex:- 4)

statements	S/E	Freq.	Total.
1. Algo ABC (a[], b[], n)	0	-	0
2. {	0	-	0
3. for (j=0; j<n; j++)	1	n+1	n+1
4. b[j] = sum(a, j+1) ;	2j+6	n	n(n+5)
5. }	0	-	0
			n^2+6n+1

→ sum (a, n) → 2n+3

↓ → 2(j+1)+3

j+1 → 2j+6

(2j+6) → n

$$\sum_{j=0}^{n-1} (2j+6)$$

$$2\sum_{j=0}^{n-1} j + 6\sum_{j=0}^{n-1}$$

$$2\left(\frac{n(n-1)}{2}\right) + 6n$$

$n^2 - n + 6n$

$n^2 + 5n$

$n(5+n)$.

Scanned with CamScanner

Ex:- (1). for (i=0; i<n; i=i+2);

→ Ans:- Time complexity = $\frac{n}{2}+1$.

(2). for (i=1; i<n; i=i*2);

→ soln:- Take different values for i

i=2 → 1, 2 → 2 times (for loop)
i=3 → 1, 2, 4 → 3 times
i=4 → 1, 2, 4 → 3 times
i=5 → 4 times
i=6 → 4 times
i=7 → 4 times

↔ Time complexity = $(\log n)_2$

(3). for (i=0; i<n; i++)
{
for (j=0; j<n; j++)
{
Statements;
}
}

→ solution on next page.

Scanned with CamScanner

→ soln:-

	Statements	S/E	Freq.	Total
(1).	for (i=0; i<n; i++)	1	n+1	n+1
(2).	{	0	0	0
(3).	for (j=0; j<n; j++)	1	(n)(n+1)	n(n+1)
(4).	{	0	0	0
(5).	statement;	1	n·n	n^2
(6).	}	0	0	0
(7).	}	0	0	0
				$2n^2+2n+1$

* Asymptotic Notations :-

Ex:- $f(n) = 2n^3 + n^2 + 2n$

→ $O(n^3)$.

Ex:- $f(n) = 4n^3 + 2n + 3$

→ $O(n^3)$.

Ex:- $f(n) = 2^n + 6n^2 + 3n$

→ $O(2^n)$.

Scanned with CamScanner

* Linear Homogenous Reccurance Relation :-

*↔ General form of homogenous reccurance relation

→ $a_0 t_n + a_1 \cdot t_{n-1} + \ldots + a_k t_{n-k} = 0$.

*↔ characteristic Equation.

→ $a_0 x^k + a_1 x^{k-1} + \ldots + a_k = 0$,

*↔ all roots are distinct (different) :-

→ $t_n = \sum_{i=1}^{k} c_i r_i^n$

→ $t_n = c_1 r_1 n + c_2 r_2 n + c_3 r_3 n$

*↔ all roots are not distinct :-

→ $t_n = \sum_{i=1}^{l} \sum_{j=0}^{m_i - 1} c_{ij} n^j r_i^n$

→ $t_n = c_1 1^n + c_2 2^n + c_3 n 2^n + c_4 3^n + c_5 n 3^n + c_6 n^2 3^n + c_7 n^3 3^n$

Ex:- $x^2 - x - 1 = 0$

→ $x = \frac{-b^2 \pm \sqrt{b^2 - 4ac}}{2a} = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1+\sqrt{5}}{2} \rightarrow r_1$

$= \frac{1-\sqrt{5}}{2} \rightarrow r_2$

Scanned with CamScanner

Problem statement: Find n^{th} Fibonacci number.

Iterative Method:
Algorithm:
f1(n):
first=0;
second=1;
for i = 3 to n
{
next = first + second;
first = second;
second = next;
}
return next;
 	
Time complexity: O(1) + O(n) + O(1) = O(n)
	Recursive Method:
	Algorithm:
	f2(n):
	if n <= 1
	return n
	return f2(n - 1) + f2(n - 2)
	
	Time complexity: ?
	If n=0 => F(0) = 0
	n=1 => F(1) = 1
	n>1 => F(n) = F(n-1) + F(n-2)

Example-1:

What is the solution of the recurrence relation fn = fn-1 + fn-2 with f0=0 and f1=1?

Solution:

Rewrite recurrence, $f_n - f_{n-1} - f_{n-2} = 0$

Compare recurrence with $a_0 t_n + a_1 t_{n-1} + \ldots. + a_k t_{n-k} = 0$

K=2, a_0=1, a_1=-1, a_2=-1

Characteristic eq. is $x^2 - x - 1 = 0$

Quadratic equation

$ax^2+bx+c=0$

a=1, b=-1, c=-1

delta=root(b^2-4ac) = root(1 + 4) =root(5)

root1 = (-b + delta)/2a = (1 +root(5))/2

root2= (-b-delta)/2a = (1 -root(5))/2

Possible roots are r1 = (1+√5)/2, r2 = (1-√5)/2

General Solution is $fn = c1r1^n + c2r2^n$

$= c1 ((1+\sqrt{5})/2)^n + c2((1-\sqrt{5})/2)^n$

Use initial condition to find c1 and c2

n=0 => $f0 = c1 + c2$ => $0 = c1 + c2$ => $c1=-c2$

n=1 => $f1 = c1((1+\sqrt{5})/2) + c2((1-\sqrt{5})/2)$ => $1 = c1((1+\sqrt{5})/2) + c2((1-\sqrt{5})/2)$

Solving above equations, $c1=1/\sqrt{5}$, $c2=-1/\sqrt{5}$

Thus, $fn = (1/\sqrt{5}) ((1+\sqrt{5})/2)^n - (1/\sqrt{5}) ((1-\sqrt{5})/2)^n$

$fn = O(1.6^n) = O(2^n)$

Example-2:

Solve the recurrence:

$$t_n = \begin{cases} 0 & , \text{if } n = 0 \\ 5 & , \text{if } n = 1 \\ 3t_{n-1} + 4t_{n-2} & , \text{otherwise} \end{cases}$$

Rewrite recurrence, $t_n - 3t_{n-1} - 4t_{n-2} = 0$

Compare recurrence with $a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0$

K=2, $a_0=1$, $a_1=-3$, $a_2=-4$

Characteristic eq. is $x^2 - 3x - 4 = 0$

Possible roots are r1 = -1, r2 = 4

General Solution is $tn = c1r1^n + c2r2^n$

$= c1 (-1)^n + c2(4)^n$

Use initial condition to find c1 and c2

n=0 => $t0 = c1 + c2$ => $0 = c1 + c2$ => $c1=-c2$

n=1 => $t1 = c1(-1) + c2(4)$ => $5 = c2 + 4c2$ => $c2=1$

Solving above equations, c1=-1, c2=1

Thus, $tn = (-1) (-1)^n + (1) (4)^n$

$tn = O(4^n)$

Example-3:

Solve the recurrence:

$$t_n = \begin{cases} n & , \text{if } n = 0, 1, 2 \\ 5t_{n-1} - 8t_{n-2} + 4t_{n-3} & , \text{otherwise} \end{cases}$$

Rewrite recurrence, $t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$

Compare recurrence with $a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0$

K=3, a_0=1, a_1=-5, a_2=8, a_3=-4

Characteristic eq. is $x^3 - 5x^2 + 8x - 4 = 0$

Possible roots are r1 = 1, r2 = 2, r3 = 2 => r1=1, m1=1 and r2=2, m2=2

General Solution is $tn = c1r1^n + c2r2^n + c3*n*r2^n$

$$tn = c1\,(1)^n + c2(2)^n + c3n(2)^n \quad \text{---------- (a)}$$

Use initial condition to find c1,c2 and c3

n=0 => t0 = c1 + c2 => 0 = c1 + c2 => c1=-c2 ---------- (1)

n=1 => $t1 = c1(1)^1+c2(2)^1+c3(2)^1$ => 1 = c1+2c2 + 2c3 ---------- (2)

n=2 => $t2 = c1(1)^2+c2(2)^2+c32(2)^2$ => 2 = c1+4c2+8c3 ---------- (3)

Solving above equations (1), (2) and (3), c1=-2, c2=2, c3=-(1/2)

Put c1, c2 and c3 into equation (a)

$tn = -2(1)^n + 2*2^n - (1/2)*n*2^n$

Thus, $tn = 2*2^n - n*2^{n-1} - 2$

$tn = O(2^n)$