5/8

Design and Analysis of Algorithm :

UNIT : I : Algorithm Analysis Techniques

Notion of Algorithm :

Set of Rules to obtain the expected output from given output.

Input – Set of rules to obtain the expected output from given output – output.

Algorithm can be used in computer science, mathematics, Research, Artificial intelligence etc..

Need for Algorithm

=> Problem solving
=> Automation
=> Innovation
=> Generalization
=> scalability
=> Performance.

characteristics

=> unambiguous
=> well defined input
=> well defined output
=> feasible
=> Robust
=> optimality
=> Easy to understa
=> Cross Lingual
=> Resource usage
=> Scalable
=> Modular.

properties of an algorithm:

Input
Output
Finitness
Effectieveness
Determines

problem solving Techniques :

Brute Force Approach (Trial and error method)
⇒ It is a traditional method)

Greedy approach (In maximum cases it takes less time)

Back-tracking approach (DFS)

Dynamic approach (It have an idea of memorizing something that is previous step or remember the intermidiate and to reuse after sometime).

Types of Algorithm

Brute Force, hashing, Graph
Recursive, divide and conquer, string
Backtracking, Greedy, Numerical
Searching, Dynamic programming, Machine Learning
sorting, Randomized, deep learning
cryptography.

Analysis of algorithm:

Time efficiency (Time complexity)

Space efficiency (space complexity)

6/8

Analysis of algorithmic efficiency :

Units for measuring Running time

The running time of an algorithm is to be measured with a unit that is independent of the extraneous factors like

Processor speed
quality of implementation
compiler and etc..

$$\underbrace{T(n)}_{\text{running time}} \approx \overbrace{C_{op}}^{\text{Execution time for basic operation}} \; C(n)$$

n → inputsize

$C(n)$ → Number of times basic operation is executed

units for measuring Running time.

Main process (main operation) → Basic operation

Ex:

Searching for a key in a list of n items

Basic operation : key comparison (main operation)

[Every single statement takes constant amount of time]

Assume every statement take 1 unit time

```
{
  int a;               -> 1
  scanf("%d", &a);     -> 1
  printf("%d", a);     -> 1
}
```

total : 3 (time taken)

```
{
  int a;                      -> 1
  scanf("%d", &a);            -> 1
  for (int i=0; i<a; i++)     ~ a+1
  {
    printf("%d", a);
  }
}
```

[Execute time : a]

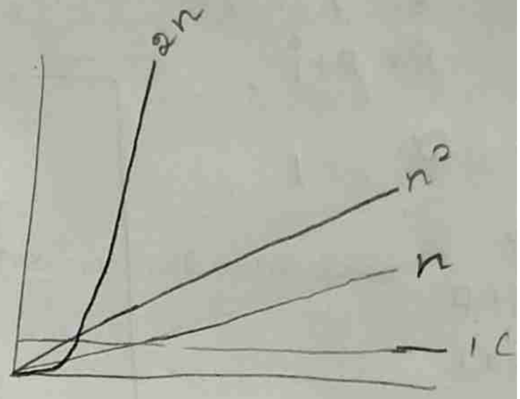
Total time : 1+1+a+1 = 2a

Order of growth:

I/P Time

→ Most best case (linear time complexity)

I/P	1	n	n^2	2^n → worst time complexity (time c
1	1	1	1	2
2	1	2	4	4
3	1	3	9	8
4	1	4	16	16
5	1	5	25	32
6	1	6	36	64
7	1	7	49	128



Order of growth

n^2 ⇒ can be occur in nested loop.

```
1 → int i,j;
1 → int n;
n+1 → for (i=0; i<=n; i++){
n(n+1) → for (j=0; j<=n; j++){
n(n) → stmt;
}
}
```

n^2 → is the greatest time taken to complete skip a components

Hence time taken: n^2

Amount of time calculating using substitution method.

Eg:1 $P=0;$

for ($i=0$; $P<=n$; $i++$)   (Based upon dependency time is reduced)

{

$P = P+i;$

}

i	P
0	$0+0$
1	$0+1$
2	$0+1+2$
3	$0+1+2+3$
4	$0+1+2+3+4$
.	
K	$0+1+2+3+4 \cdots +K \Rightarrow \frac{n(n+1)}{2}$

$P > n \Rightarrow$ this excution will stop

$\frac{K(K+1)}{2} > n$

$K^2 + K > 2n$

// We alway take greater time taken

$K^2 > 2n$

$K^2 > n$

$\boxed{K > \sqrt{n}}$

Eg:2 [. In updation part if you have $*, /$ → Time taken is in log)

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

{

stmt; // statement

}

$i = i+2$

i	stmt
1	1
2	1
2^2	1
2^3	1
2^4	1
2^5	1
⋮	⋮
2^k	1

1
2
4
6
8

} here some number are skipped hence time taken is little bit small compare to running n times without skipping

$i > n$ (excution should stop)

$2^k > n$

taking log on both sides

$\log 2^k > \log n$

$k > \log_2 n$

Eg : 3

```
for (i=n; i>=1; i = i/2)
{
stmt;
}
```

for (i=0; i<=n; i++) (n times)
for (i=n; i>=1; i++) (n times)

i	stmt
n	1
$n/2$	1
$n/2^2$	1
$n/2^3$	1
⋮	⋮
$n/2^k$	1

$i < 1$ (execution should stop)

$\frac{n}{2^k} < 1$

$n < 2^k$ ⇒ taking log on b.s

$\log n = \log 2^k$

$k = \log_2 n$

$\boxed{\log_2 n}$

Eg : 4

```
for (i=1; i*i <=n; i++)
{
stmt;
}
```

i
1
2
⋮
k

$i^2 > n$ (execution stop)

$k^2 > n$

$\boxed{k > \sqrt{n}}$

Eg : 5

```
P = 0
for ( i = ϕ ; i<n ; i = i*2 )
  { p++; }
for ( j = ϕ; j < p; j = j*2 )
  {
   stmt;
  }
```

→ log n

log p

p → log n

→ log (log n)

↓

as it depends on previous factor

Total time taken (log log n)

Eg : 6

```
for ( i = 0 ; i <= n; i++ )
{
for ( j = 1 ; j <= n ; j = j*2 )
   {
   stmt;
   }
}
```

→ n (times)

→ log n

$\boxed{}$ i → 1

$\boxed{}$ j – 1

2

↙

Space complexity

as we only using the space and we didn't need any space.

Time taken n log n

for (i = 0 ; i<n; i++) → n (amount of time)

for (i = 0 ; i<n; i = i+2) → n (amount of time)

for (i = 1 ; i<=n; i = i*2) → $\log_2 n$ (amount of time)

i*3 → $\log_3 n$

for (i = n; i>=n; i = i/2) → $\log_2 n$ (amount of time)

i = i/3 → $\log_3 n$

; // If semicolon is given after the forloop then it comple that but loop will terminate

```
int a = 0 , int i ;
for (     i = 1 ; i<=10; i++)
{
if ( i % 2 ==0)
{
Printf ("even = %d" , i);
}

else
{
Printf ("odd = %d", i);
}
}

Print (" \n Bye %d", a + i);
```

O/P:

```
a = 5

switch(a) {

case 5: Print ("A");
case 6 : Printf ("B");
case 7 : Printf ("C");
default : Print f ("D")
```

O/P ABCD

continue should used only in loops cannot be used in switch

1/8

Recurrence Relation

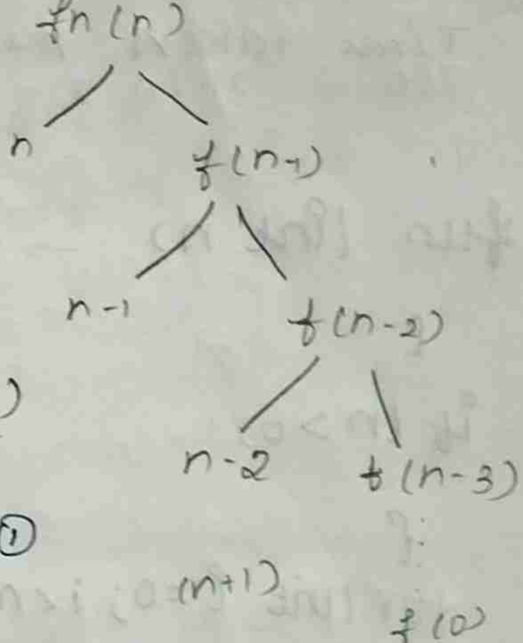
Methods :

- ↳ Substitution
- ↳ Iteration
- ↳ Master theorem.

Eg:

fn (int n) → T(n) → To find time taken for this fun

```
{
if (n>0)
  {
  printf("%d", n);  } — 1
  fn(n-1);          → T(n-1)
  }
}
```



$T(n) = T(n-1)+1 \rightarrow ①$

$\boxed{n = n-1}$

$T(n-1) = T(n-1-1)+1$

$T(n-1) = T(n-2)+1 \rightarrow ②$

Sub ② in ①

$T(n) = T(n-2)+1+1$

$T(n) = T(n-2)+2$

$T(n-3)+2+1$

$= T(n-3)+3$

$\vdots$

$T(n-k)+k$

$n-k=0$ (this exution will stop)

$n-k=0$

$n=k$

$T(n-n)+n$

$T(n) = T(n-1)+1$

sub $n = n-1$

$T(n-1) = T(n-1-1)+1$

$T(n-1) = T(n-2)+1$

$T(n-2) = T(n-1-2)+1,$

$T(n-2) = T(n-3)+1$

$T(n-2) = T(n-2-1)$

$n(n-2) = T(n-3)+1$

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1)+1, & n>0 \end{cases}$$

```
fact (int n)
 {
 if (n>0)
   {
   n*fact (n-1);
   }
 }
```

$$T(n) = \begin{cases} 1, & n = 0 \\ T(n-1)+1, & n > 0 \end{cases}$$

Time taken: n+1

```
fun (int n)      -> T(n)
 {
 if (n>0)
  {
  for (int i=0; i<n; i++)
       printf ("%d", n);  — n
       fun (n-1);  — T(n-1)
     }
  }
```

$T(n) = T(n-1) + n$

$$T(n) = \begin{cases} 1, & n = 0 \\ T(n-1)+n, & n > 0 \end{cases}$$

$T(n) = T(n-1) + n \rightarrow (1)$

$= \underline{T(n-2) + (n-1) + n}$

// sub $T(n-2) = T(n-3) + n-2$

$= T(n-3) + (n-2) + (n-1) + n$

$\vdots$

$T(n) = T(n-1)$

sub $n = n-1$ in (1)

$n = n-1$

$T(n-1) = T(n-1-1) + n-1$

$= T(n-2) + n-1$

sub in (1)

$T(n) = T(n-2)$

again $n = n-2$ in (1)

$T(n-2) = T(n-3) + (n-2)$

$= T(n-k) + (n-(k-1)) + (n-(k-2)) + \cdots n$

$= T(n-k) + (n-k+1) + (n-k+2) + \cdots n \rightarrow (3)$

$n - k = 0$ (This exuition will stop)

$\boxed{n = k}$

sub in (3)

$T(\ = T(0) + \underbrace{1 + 2 + \cdots \quad n}$

$1 + \frac{n(n+1)}{2} \rightarrow$ natural numbers formula $\boxed{\frac{n(n+1)}{2}}$

$= 1 + \frac{\boxed{n^2} + n}{2}$

Time Complexity : n^2

8/8/24

$T(n) = T(n-1) + 1 \Rightarrow n$ (Time complexity)

$T(n) = T(n-1) + n \Rightarrow n^2$

$T(n) = 2T(n-1) + 1 \Rightarrow 2^n$

$a = 1$

$r = 2$

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

$\Rightarrow 2^{n+1} - 1$

$\boxed{\Rightarrow 2^n}$

$\Rightarrow 2^{n+1} - 1$

$$\boxed{\Rightarrow 2^n}$$

Time Complexity $\Rightarrow 2^n$

$T(n) = 2T(n-1) + 1 \longrightarrow (1)$

$T(n) = \begin{cases} 1, & n = 0 \\ 2T(n-1)+1, & n > 0 \end{cases}$

$T(n) = T(n-1)$

sub $n = n-1$ in (1)

$T(n-1) = 2T(n-2)+1 \longrightarrow (2)$

sub (2) in (1)

$T(n) = 2(2T(n-2)+1)+1)$

$= 2^2 T(n-2) + 2 + 1 \longrightarrow (3)$

$T(n) = T(n-2)$

$n = n-2$ in (1)

$T(n-2) = 2T(n-3)+1 \longrightarrow (4)$

sub (4) in (3)

$T(n) = 2^2 (2T(n-3)+1) + 2 + 1$

$= 2^3 T(n-3) + 4 + 2 + 1 \longrightarrow (5)$

$T(n) = T(n-3)$

$n = n-3$ in (1)

$T(n-3) = 2T(n-3-1) + 1$

$T(n-3) = 2T(n-4) + 1 \longrightarrow (6)$

sub (6) in (5)

$T(n) = 2^3 (2T(n-4)+1) + 4 + 2 + 1$

$= 2^4 (T(n-4) + 8 + 4 + 2 + 1$

$= 2^4 T(n-4) + 2^3 + 2^2 + 2^1 + 2^0$

$\vdots \qquad \vdots$

$= 2^k T(n-k) + 2^{k-1} + 2^{k-2} + 2^{k-3} + \cdots 2^1 + 2^0 \rightarrow$ (7)

$n-k=0$

$n=k$

sub n in place of k in (7)

$T(n) = 2^n T(n-n) + 2^{n-1} + 2^{n-2} + \cdots 2^1 + 2^0$

$= 2^n + 2^{n-1} + 2^{n-2} + \cdots 2 + 1$

(Formula : $a + ar + ar^2 + ar^3 + ar^4 + \cdots a^n) = \frac{a(r^{n+1}-1)}{r-1}$

$a = 1$, $r = 2$ $\Rightarrow$ $\frac{1(2^{n+1}-1)}{1}$

$= 2^{n+1} - 1$

$T(n) = 2^n$

Time complexity $= 2^n$.

$2T(n-1)+n \rightarrow (1)$

$T(n) = \begin{cases} 1, & n=0 \\ 2T(n-1)+n, & n>0 \end{cases}$

$T(n) = 2T(n-1)+n \rightarrow (1)$

sub $n = n-1$ from (1) in (1)

$T(n-1) = 2T(n-2)+(n-1) \rightarrow (2)$

sub (2) in (1)

$T(n) = 2[2T(n-2)+(n-1)]+n$

$= 2^2 T(n-2)+2(n-1)+n \rightarrow (3)$

from (3) $n = n-2$
sub in (1)

$T(n-2) = 2T(n-3)+(n-2) \rightarrow (4)$

sub (4) in (3)

$= 2^2[2T(n-3)+(n-2)]+2(n-1)+n$

$= 2^3 \cancel{\,} T(n-3)+2^2(n-2)+2(n-1)+n$

$= 2^3 T(n-3)+2^2(n-2)+2(n-1)+n \rightarrow (5)$

$\vdots \quad \vdots \quad \vdots$

$= 2^k T(n-k)+2^{k-1}(n-(k-1))+2^{k-2}(n-(k-2)) \cdots +n$

$= 2^k T(n-k)+2^{k-1}(n-k+1)+2^{k-2}(n-k+2+\cdots+n$

$T(0) \rightarrow$ Excution stop

$n-k=0$

$n=k$ [sub n in place of k

$= 2^n T(n-n)+2^{n-1}(n-n+1)+2^{n-2}(n-n+2)+\cdots+n$

as it is greatest $\leftarrow$ $= 2^n + 2^{n-1}\cdot 1 + 2^{n-2}\cdot 2 + 2^{n-3}\cdot 3 + \cdots 1\cdot n$

$2T(n-1)+n \rightarrow ①$

$T(n) = \begin{cases} 1, & n=0 \\ 2T(n-1)+n, & n>0 \end{cases}$

$T(n) = 2T(n-1)+n \rightarrow ①$

sub $n = n-1$ from ① in ①

$T(n-1) = 2T(n-2)+(n-1) \rightarrow ②$

sub ② in ①

$T(n) = 2[2T(n-2)+(n-1)]+n$

$= 2^2 T(n-2)+2(n-1)+n \rightarrow ③$

from ③ $n = n-2$ sub in ①

$T(n-2) = 2T(n-3)+(n-2) \rightarrow ④$

sub ④ in ③

$= 2^2[2T(n-3)+(n-2)]+2(n-1)+n$

$= 2^3 T(n-3)+2^2(n-2)+2(n-1)+n$

$= 2^3 T(n-3)+2^2(n-2)+2(n-1)+n \rightarrow ⑤$

$\vdots$

$= 2^k T(n-k)+2^{k-1}(n-(k-1))+2^{k-2}(n-(k-2)) \cdots + n$

$= 2^k T(n-k)+2^{k-1}(n-k+1)+2^{k-2}(n-k+2+\cdots+n$

T(0) -> Excution stop

$n-k=0$

$n=k$ [sub n in place of k

$= 2^n T(n-n)+2^{n-1}(n-n+1)+2^{n-2}(n-n+2)+\cdots+n$

as it is greatest

$= 2^n + 2^{n-1}\cdot 1 + 2^{n-2} \cdot 2 + 2^{n-3}$