DATE : / /
PAGE NO. :

DAA

10/8/23

D - Design &
A - Analysis of
A - Algorithm

Algorithm - The set of instructions in a particular order to perform a particular task

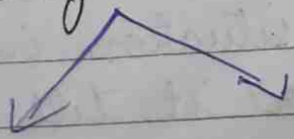
→ Algorithm should not be ambiguous
→ Algorithm should not contain infinite statement
→ Statements in algorithm should be in particular order.
→ Algorithm should be effective ← Time, Space
→ Termination should be there in Algorithm

Approaches to solve a Problem -

1 D & C (Divide & Conquer)

Merge Sort | Binary Search | Quick Sort | Matrix

2 Greedy Method -



K map | Huffman

3 Dynamic Programming method -

↓

4 Randomized

Time Complexity of Quick sort in worst case scenario is n^2 when we apply D & C rule

Eg Quick sort

5 7 9 10 12 13

Here 5 is the pivot element and is present at the right position so we cannot divide the situation in 2 parts so we have to compare 5 till n times to get the correct position

Now we take 7 as a pivot element & still we cannot divide the situation in 2 parts so we have to compare it till $n-1$ times

Time Complexity = $n + n-1 + n-2 - - - - 1$

DATE : / /
PAGE NO. :

So time complexity of Quick sort in worst case scenario is n^2

In Quick sort, worst case scenario is found when the problem is already sorted in either increasing or decreasing order

In Quick sort, in best case & average case scenario the time complexity will be $n \log n$ when we apply divide & conquer rule.

& When we apply randomized pivot element approach the time complexity for Quick sort in best, average & worst case scenario will be constant i.e $n \log n$.

5 ~~Approximation~~

When we can't able to find out the soln of a problem then we try to find the approximate soln. Eg - finding factorial of higher no.

6 Branch & Bound

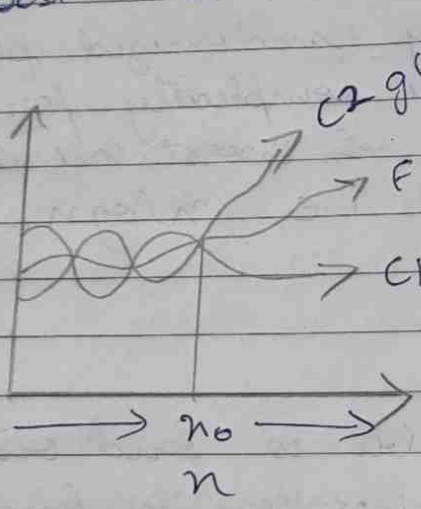
In this we solve the situation in a tree manner which has many branches to reach the soln & has lower bound (min) & upper bound (max)

7 Backtracking -

In this we move on a path to find the solu" & if we get to know that we don't find the solu" on this path then we take a back step & explores a new path to get the solu".

Asymptotic Notation -

Worst case = Big O
Avg case = Θ
Best = ω



After some time the F(n) will be lesser then C2 g(n) & greater then C1 g(n)

Algorithm -

Characteristics -

1) Finiteness

DATE : / /
PAGE NO. :

2) Ambiguity
3) Sequence
4) I/O
5) Feasibility

Analysis –

1. Time
2. Space
3. Energy

Bubble Sort

The time complexity of bubble sort is $O(n^2)$.

In every case (best, worst & average), time complexity of bubble sort will be $O(n^2)$.

In the modified bubble sort the time complexity when the array is already sorted will be $O(n)$.

In modified bubble sort we take a flag value which is initially set to 0 (False) and if the statement goes in if loop then swapping will occur & the flag value goes 1 (True) & we check the flag value at the end of the if loop if it is false then this means that the array is already sorted.

Selection Sort

Time complexity for selection sort (best, worst, avg) ∞ / for every case will be $O(n^2)$

In this sorting we find the smallest no. from the array & then swipe it with the first no. of the array then we find the second smallest no. from the array & swipe it with the second no. of the array & so on.

Eg

5	3	8	2	10

2	3	8	5	10

2	3	5	8	10

Insertion Sort

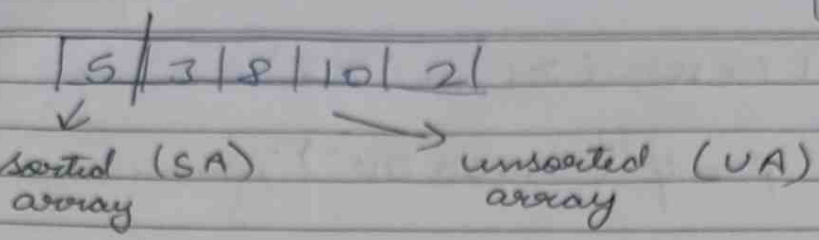
In this sorting we take the first no. of the array as the sorted array & rest n-1 no. as the unsorted array (divide the array in 2 parts - sorted array with first no. of the array & unsorted array with the rest n-1 no.) & we try to insert the no.'s of unsorted array in the sorted array at their correct position one by one.

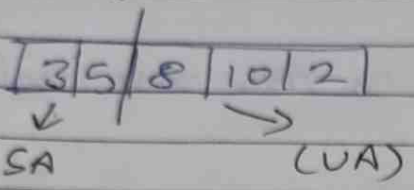
In this sorting the time complexity of worst & average case will be $O(n^2)$.

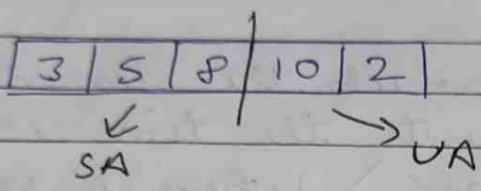
In this sorting the time complexity of best case will be $O(n)$.

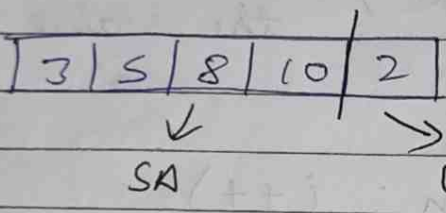
DATE : / /
PAGE NO. :

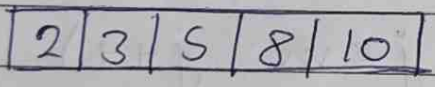
Eg











→ Sorted array

Time Complexity –

Eg
```
for (i=1; i<n; i+k)
{
    printf ("Hello JV");
}
```

The time complexity will be $O\left(\frac{n}{k}\right)$

→

Eg -
```
for(i=n; i>1; i-2)
{
    printf("Hello JV");
}
```

The time complexity in this case will be $O(n/2)$

The time by which the element increments or decrements the time complexity will be $O(n/k)$ where k is the no by which the value increments or decrements.

Eg
```
for(i=1; i<n; i++)
{
    for(j=1; j<m; j++)
    {
        printf("Job for GF");
    }
}
```

The time complexity in this case will be $O(n \cdot m)$

→ In multiplication of two matrices of same size the time complexity will be $O(n^3)$

DATE : / /
PAGE NO. :

Eg
```
for(i=1; i<n; i*2)
{
    printf("Hello JV");
}
```

let n = 16

So the time complexity in this case will be $O(\log_2 n)$ as the Hello JV will be printed 4 times.

@

Eg
```
for(i=n; i>1; i/2)
{
    printf("Hello JV");
}
```

let n = 16

Hence the loop will run 4 time
So the time complexity will be $O(\log_2 n)$

for i/K

The complexity will be $O(\log_K n)$

Eg
```
for(i=1; i<n; i+2)
{
    for(j=1; j<n; j*3)
    {
        printf("Job for GF");
    }
}
```

DATE: / /
PAGE NO.:

Time complexity in this case will be $O\left(\frac{n}{2} \log_3 n\right)$

24/8/23

Solu" of a problem

- Iterative
- Recursive
  - we divide the bigger problem into smaller part & then solve that smaller part

Iterative

eg

```
for (i=1; i^2<n; i++)
{
    printf ("Time is money");
}
```

Time complexity will be $O(\sqrt{n})$

as $i^2 < n$

$i < \sqrt{n}$

→

DATE : / /
PAGE NO. :

eg
```
A()
{
  i=1; S=1;
  while (S<=n)
  {
    i++;
    S=S+i;
    printf ("Hello");
  }
}
```

Time complexity will be $O(\sqrt{n})$ as

i	1	2	3	4	5	6	- - -	k
S	1	3	6	10	15	21		$\frac{k(k+1)}{2}$

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

$$k^2 > n$$

$$k > \sqrt{n}$$

eg
```
A()
{
  for (i=1; i<=n; i++)
  {
    for (j=1; j<i; i++)
    {
      for (k=1; k<=100; k++)
      printf ("Hello"); }
  }
}
```

Time complexity will be $O(n^2)$ as

i	1	2	3	4	- - -	n
j	1	2	3	4	- - -	n
k	100	2×100	3×100	4×100	- - -	n×100

So $1\times100 + 2\times100 + 3\times100 - - - - n\times100$

$= 100(1+2+3+4+ - - n)$

$= 100\left(n\left(\frac{n+1}{2}\right)\right)$

$= O(n^2)$

Eg
```
A()
{
  for(i=0; i<=n; i++)
  {
    for(j=1; j<=i; j++)
    {
      for(k=1; k<=n/2; k++)
      {
        printf("Hello");
      }
    }
  }
```

Time complexity will be $O(n^4)$ as

DATE : / /
PAGE NO. :

i	1	2	3	4	- - - -	n
j	1	4	9	16	- - -	n^2
k	$\frac{n}{2}$	$4 \times \frac{n}{2}$	$9 \times \frac{n}{2}$	$16 \times \frac{n}{2}$		$n^2 \times \frac{n}{2}$

$$\frac{n}{2}(1+4+9+16+ \cdots + n^2)$$

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

$$= O(n^4)$$

Eg

```
A()
{
  for(i = n/2 ; i <= n ; i++)  --> n/2
  {
    for(j = 1 ; j <= n/2 ; j++)  --> n/2
    {
      for(k = 1 ; k <= n ; k = k*3)  --> log3 n
      {
        printf("Hello");
      }
    }
  }
}
```

Time complexity will be $O\left(\frac{n^2}{4}\log_3 n\right)$ as

$$\frac{n}{2} \times \frac{n}{2} \times \log_3 n = \frac{n^2}{4}\log_3 n$$

eg
```
A()
{
while (n>1)
{
   n = n/2;
   Hello;
}
}
```

Time Complexity will be $O(\log_2 n)$ as this problem can be written as.

```
for (i = n; i>1; i = i/2)
{
    printf ("Hello");
}
```

eg
```
A()
{
  for (i = 1; i<=n; i++)
  {
    for (j = 1; j<=n; j+i)
    {
        printf ("Hello");
    }
  }
}
```

Time Complexity will be $O(n \log n)$ as

DATE : / /
PAGE NO. :

i	1	2	3	—	—	—	n
j	n	$\frac{n}{2}$	$\frac{n}{3}$				$\frac{n}{n}$

$$n + \frac{n}{2} + \frac{n}{2} + -- \frac{n}{n}$$

$$n\left\{1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right\}$$

$$= n \log n$$

Eg
```
int n = 2^2^k
for (i = 1; i <= n; i++)
{
    j = 2;
    while (j <= n)
    {
        j = j^2
        printf ("Hello");
    }
}
```

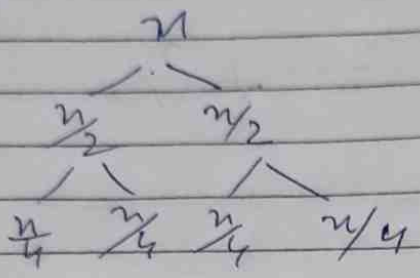
Time complexity will be $O(n(K+1))$
as $= O(n(\log \log n))$

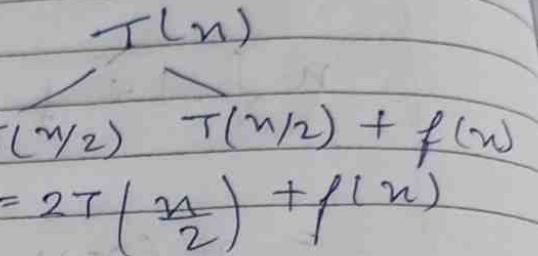
K	1	2	3
n	4	8	16
j	2,4	2,4,8	2,4,8,16
	= 2	3	4

DATE : / /
PAGE NO. :

$n = 2^{2^k}$ can be written as
$\log n = \log 2^k$
$\log (\log n) = 2^k$

Time complexity of Recursion





$= 2T\left(\frac{n}{2}\right) + f(n)$

$= 2^2 T\left(\frac{n}{2^2}\right) + f(n)$

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

29/08/23

Recursion -

- → Iterative
- → Substitution
- → Tree
- → Master Theorem

Recursion (Iterative way) -

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

DATE : / /
PAGE NO. :

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

Put (2) & (3) in (1)

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

· at K^{th} term.

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

after dividing the problem to the smallest part $T(n-k)$ will be equal to $T(1)$
i.e $n - k = 1$

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

$n - k = 1$
$n = k + 1$
$k = n - 1$

$T(n) = n - 1 + T(1)$ as $T(1) = 1$ from the ques.
$T(n) = n - 1 + 1$
$= n$

DATE : / /
PAGE NO. :

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

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

put eq (2) & (3) in (1)

$T(n) = n + n-1 + T(n-2)$
$T(n) = n + n + n - 1 - 2 + T(n-3)$
$T(n) = 3n - 3 + T(n-3)$
$T(n) = n + (n-1) + (n-2) + T(n-3)$
at k^{th} term

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

at last stage
$T(n-(k+1))$ will become equals to $T(1)$

So $n-(k+1) = 1$
$n - k - 1 = 1$
$n = k + 2$
$k = n - 2$

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

DATE : / /
PAGE NO. :

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

$= 2 + 1$

$= 3$

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

So $\frac{n(n+1)}{2} = O(n^2)$

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

$T(n) = T(n-1) + \log n$ — (1)

$T(n-1) = T(n-2) + \log(n-1)$ — (2)

$T(n-2) = T(n-3) + \log(n-2)$ — (3)

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

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

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

at k^{th} term.

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

$\log a + \log b = \log ab$

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

PAGE NO. :

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

$n - k - 1 = 1$

$n - k = 2$

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

$T(n) = T(2) + \log(n!)$

$T(2) + \log(n^n)$

$T(2) = n \log n$

$= O(n \log n)$

(4) $T(n) = \begin{cases} T\left(\frac{n}{2}\right) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$

$T(n) = T\left(\frac{n}{2}\right) + 1$ — (1)

$T(n-1) = T\left(\frac{n-1}{2}\right) + 1$ — (2)

$T(n-2) = T\left(\frac{n-2}{2}\right) + 1$ — (3)

put eq^n (2) & (3) in (1)

$T(n) =$

DATE : / /
PAGE NO. :

$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$ — (2)

$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$ — (3)

put eq^n (2) & (3) in (1)

$T(n) = 1 + 1 + T\left(\frac{n}{4}\right)$

$T(n) = 1 + 1 + 1 + T\left(\frac{n}{8}\right)$

$= 3 + T\left(\frac{n}{8}\right)$

$= 3 + T\left(\frac{n}{(2)^3}\right)$

for k^{th} term

$T(n) = T\left(\frac{n}{2^k}\right) + k$

as $T\left(\frac{n}{2^k}\right) = 1$

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

$n = 2^k$

$\log n = k$

Complexity of Binary search

$T(n) = T(1) + \log n = 1 + \log n$

$= O(\log n)$

DATE : / /
PAGE NO. :

(5) $T(n) = \begin{cases} 27\left(\frac{n}{2}\right)+1 & n>1 \\ 1 & n=1 \end{cases}$

$T(n) = 27\left(\frac{n}{2}\right)+1$ ——— (1)

$T\left(\frac{n}{2}\right) = 27\left(\frac{n}{4}\right)+1$ ——— (2)

$T\left(\frac{n}{4}\right) = 27\left(\frac{n}{8}\right)+1$ ——— (3)

put eq^n (2) & (3) in (1)

$T(n) = 27\left(27\left(\frac{n}{4}\right)+1\right)+1$

$T(n) = 27\left\{27\left[27\left(\frac{n}{8}\right)\right]\right\}+1+1$

$T(n) = 8T\left(\frac{n}{8}\right)+3$

$T(n) = 2^2 T\left(\frac{n}{4}\right)+2+1$

$T(n) = 2^2\left\{27\left(\frac{n}{8}\right)+1\right\}+2+1$

$T(n) = 2^3 T\left(\frac{n}{2^3}\right)+4+2+1$

at k^{th} term

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

at last stage $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$\log n = k$$

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

$$T(n) = n + \underbrace{2^0 + 2^1 + 2^2 + \cdots - 2^{k-1}}_{\hookrightarrow \text{GP}}$$

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

$a = 1 \qquad r = 2 \qquad n = k - 1$

$$S_n = \frac{1\left(2^{k-1} - 1\right)}{2 - 1}$$

$$S_n = 1(2^{k-1} - 1)$$

$$S_n = 2^{k-1} - 1$$

$$S_n = n - 1$$

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

$$= \frac{2n + n - 2}{2}$$

$$Tn = \frac{3n}{2} - 1$$

→ Complexity of Merge sort

DATE : / /
PAGE NO. :

⑥ $$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \text{— ①}$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \quad \text{— ②}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} \quad \text{— ③}$$

put eq^n ② & ③ in ①

$$T(n) = 2\left\{2T\left(\frac{n}{4}\right) + \frac{n}{2}\right\} + n$$

$$T(n) = 2^2 T\left(\frac{n}{4}\right) + n + n$$

$$T(n) = 2^2\left\{2T\left(\frac{n}{8}\right) + \frac{n}{4}\right\} + 2n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + n + 2n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

at k^{th} term

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

DATE : / /
PAGE NO. :

at last $\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = k$

$T(n) = n\,T(1) + n \log n$

$T(n) = n + n \log n$

$= O(n \log n)$

↳ complexity of merge sort & quick sort

(7) $T(n) = \begin{cases} 3T\left(\frac{n}{4}\right) + n & n > 1 \\ 1 & n = 1 \end{cases}$

$T(n) = 3T\left(\frac{n}{4}\right) + n$ — (1)

$T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{16}\right) + \frac{n}{4}$ — (2)

$T\left(\frac{n}{16}\right) = 3T\left(\frac{n}{64}\right) + \frac{n}{16}$ — (3)

put eq~ (2) & (3) in (1)

$T(n) = 3\left\{3T\left(\frac{n}{16}\right) + \frac{n}{4}\right\} + n$

DATE : / /
PAGE NO. :

$$T(n) = 3^2 T\left(\frac{n}{16}\right) + \frac{3n}{4} + n$$

$$T(n) = 3^2\left\{3T\left(\frac{n}{64}\right) + \frac{n}{16}\right\} + \frac{3n}{4} + n$$

$$T(n) = 3^3 T\left(\frac{n}{64}\right) + \frac{3^2 n}{16} + \frac{3n}{4} + n$$

$$T(n) = 3^3 T\left(\frac{n}{64}\right) + \frac{9n}{16} + \frac{3n}{4} + n$$

$$T(n) = 3^3 T\left(\frac{n}{64}\right) + n\left(\frac{9}{16} + \frac{3}{4} + 1\right)$$

$$3^3 T\left(\frac{n}{64}\right) + n\,\frac{9+12+16}{16}$$

$$3^3 T\left(\frac{n}{64}\right) + n\left(\frac{37}{16}\right)$$

at k^{th} term :

$$3^k T\left(\frac{n}{4^k}\right) + n\left(\frac{37}{16}\right)$$

at last

$T(n) \; 3^3 T\left(\frac{n}{64}\right) + \frac{9n}{16} + \frac{3n}{4} + n$

at k^{th} term.

$$T(n) = 3^k T\left(\frac{n}{4^k}\right) + \left\{ n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots + \left(\frac{3}{4}\right)^{k-1} n \right\}$$

$$T\left(\frac{n}{4^k}\right) = 1$$

$$n = 4^k$$

$$\log n = k$$

$$T(n) = 3^{\log_4 n} T(1) + \left\{ n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots + \left(\frac{3}{4}\right)^{k-1} n \right\}$$

$$S_n = \frac{a(1-r^n)}{1-r}$$

∞ for infinity

$$S_n = \frac{a}{1-r}$$

DATE : / /
PAGE NO. :

$a = 1 \quad r = \frac{3}{4}$

$S_n = \frac{1}{1-\frac{3}{4}} = \quad = 4$

$T(n) = 3^{\log_4 n} + 4n$

$T(n) = n^{\log_4 3} + 4n$

$= n + 4n$

$= 5n$

(8) $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & n > 1 \\ 1 & n = 1 \end{cases}$

$T(n) = 2T\left(\frac{n}{2}\right) + n$ — (1)

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$ — (2)

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$ — (3)

put (2) & (3) in (1)

$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$

DATE : / /
PAGE NO. :

$$T(n) = 4T\left(\frac{n}{4}\right) + \frac{2n}{n} + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + n + 2n$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

at k^{th} term

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

at last $\frac{n}{2^k} = 1$

$$n = 2^k$$

$$\log n = k$$

$$T(n) = nT(1) + n\log n$$

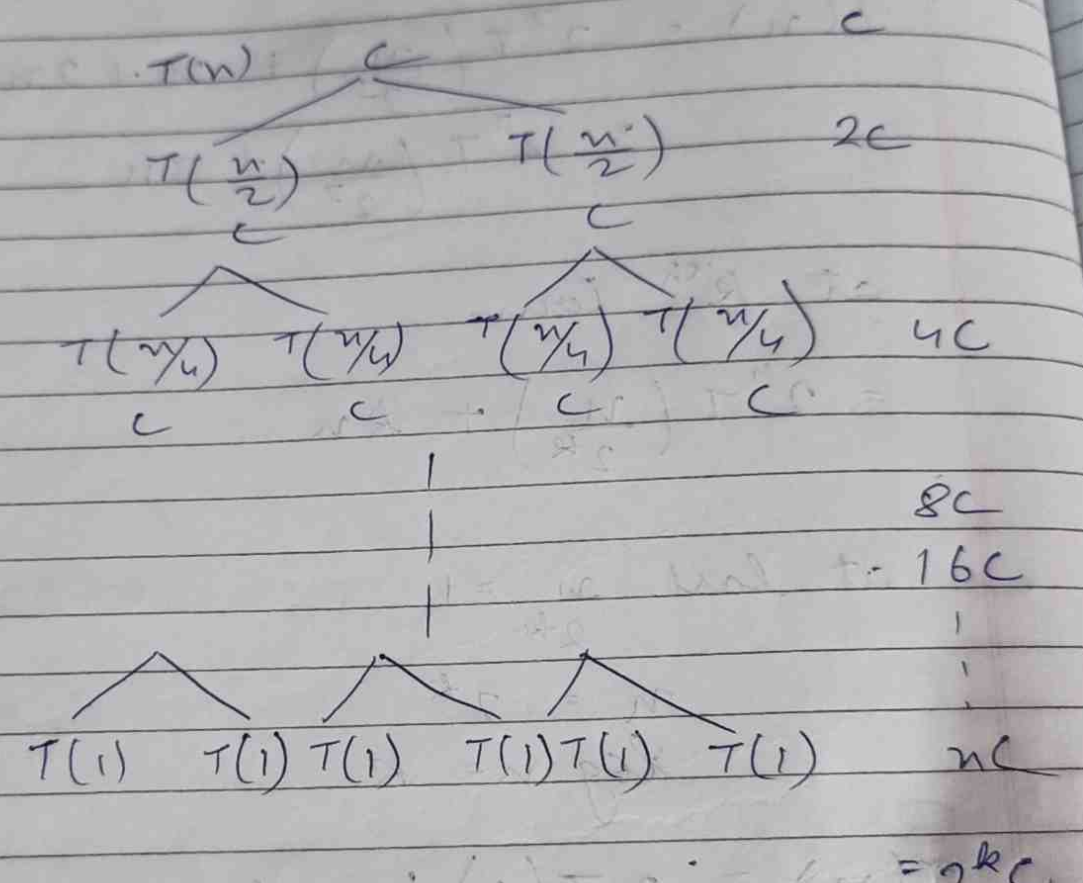
$$= n + n\log n$$

$$= O(n\log n)$$

4/9/23

Recursive tree method -

① $T(n) = \begin{cases} 2T(\frac{n}{2}) + c & \\ c & n = 1 \end{cases}$



$= 2^k c$

$c + 2c + 4c + 8c + \cdots nc$

$c[1 + 2 + 4 + 8 + \cdots 2^k]$

- GP - $S_n = a\left[\frac{r^n - 1}{r - 1}\right]$

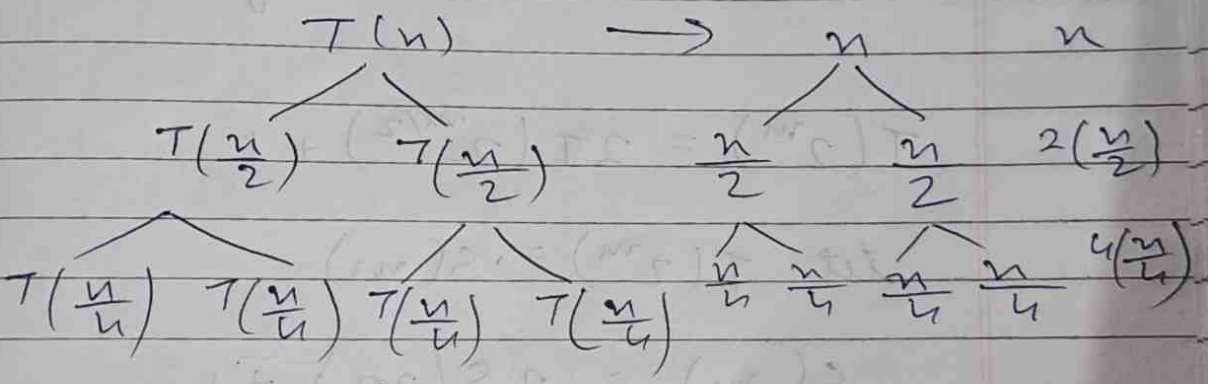
$= \frac{1[2^{k+1}-1]}{2-1}$

$= \frac{2 \cdot 2^k - 1}{1}$

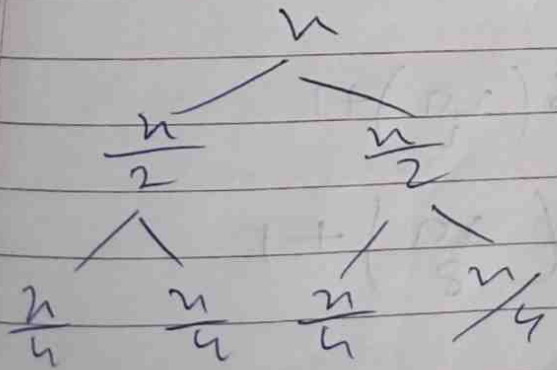
$= 2n - 1$

$= O(n)$

(2) $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & n > 1 \\ 1 & n = 1 \end{cases}$



~~$2^0 n + 2^1 n + 2^2 n$~~



$n = 2^0 n = n$

$2\left(\frac{n}{2}\right) = 2^1\left(\frac{n}{2}\right) = n$

$4\left(\frac{n}{2}\right) = 2^2\left(\frac{n}{2}\right) = n$

$\vdots$

kn

DATE : / /
PAGE NO. :

$T = \left(\frac{n}{2^k}\right)$

as last $T\left(\frac{n}{2^k}\right) = 1$

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

$n = 2^k$

$\log_2 n = k$

(3) Substitute method

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

let $n = 2^m$ ← $n^{\frac{1}{2}} = 2^{m/2}$, $\log n = m$

$T(2^m) = 2T(2^{m/2}) + 1$

let $T(2^m) = S(m)$

$S(m) = 2S\left(\frac{m}{2}\right) + 1$

~~at k^{th} term.~~

$S\left(\frac{m}{2}\right) = 2S\left(\frac{m}{4}\right) + 1$

$S\left(\frac{m}{4}\right) = 2S\left(\frac{m}{8}\right) + 1$

DATE : / /
PAGE NO. :

$S(n) = 2\left[2S\left(\frac{n}{4}\right)+1\right]+1$

$S(n) = 4S\left(\frac{n}{4}\right)+3$

$S(n) = 4\left[2S\left(\frac{n}{8}\right)+1\right]+3$

$S(n) = 2^3 S\left(\frac{n}{2^3}\right)+7$

at k^{th} term.

$= 2^k S\left(\frac{n}{2^k}\right) + k$

(2) $T(n) = 2T(\sqrt{n}) + \log n$

let $n = 2^m$

$T(2^m) = 2T(2^{m/2}) + \log 2^m$

let $T(2^m) = S(m)$

$S(m) = 2S\left(\frac{m}{2}\right) + \log 2^m$

$S(\frac{n}{2}) = 2S(\frac{n}{4}) + \log 2^n$

Master Theorem –

$T(n) = aT(\frac{n}{b}) + \theta(n^k \log^p n)$

$a \geq 1$; $b > 1$; $k \geq 0$ & p is a real no

↓

where a denotes no. of sub problems

Note → In binary search only the value of a will be equal to 1.

b denotes the coefficient of the no. of a elements in a sub problem.

→

Case 1 $(a > b^k)$ then
$T(n) = \theta(n^{\log_b a})$

Case 2 if $(a = b^k)$

i) if $p > -1$, then $T(n) = \theta(n^{\log_b a} \cdot \log^{p+1} n)$

ii) if $p = -1$, then $T(n) = \theta(n^{\log_b a}, \log\log n)$

iii) if $p < -1$, then $T(n) = \theta(n^{\log_b a})$

Case 3 $(a < b^k)$

i) if $(p \geq 0)$, then $T(n) = \theta(n^k \log^p n)$

ii) if $(p < 0)$, then $T(n) = \theta(n^k)$

Q1 $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

On comparing with $T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$

$a = 3 \; ; \; b = 2 \; ; \; k = 2 \; ; \; p = 0$

$a = b^k$

$3 < (2)^2 \; ; \; 3 < 4$

So

$(a < b^k)$ as $p = 0$

$T(n) = \theta(n^2)$

Q4

Q2 $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

$a = 4 \; ; \; b = 2 \; ; \; k = 2 \; ; \; p = 0$

$a = b^k$

$4 = (2)^2$

So $a = b^k$

as $p = 0$ So $p > -1$

4

~~$T(n) = \theta(n \log$~~

$T(n) = \theta(n^{\log_2 4} \cdot \log^{0+1} n)$

$= \theta(n^{\log_2 4} \cdot \log n)$

$= \theta(n^{\log_2 (2)^2} \log n)$

$= \theta(n^2 \log n)$

Q3 $T(n) = T\left(\frac{n}{2}\right) + n^2$

$a = 1 \; ; \; b = 2 \; ; \; k = 2 \; ; \; b = 0$

$a = b^k$

$1 < (2)^2$

So $a < b^k$

4 as $b = 0$ So $p \geq 0$

$T(n) = \theta(n^2)$

DATE : / /
PAGE NO. :

Q4. $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \log n$

$a = \sqrt{2}$; $b = 2$, $k = 0$; $p = 1$

$a < b^k$

$\sqrt{2} = 1$

So. $a > b^k$

$T(n) = \theta(n^{\log_2 \sqrt{2}})$

$= \theta(\sqrt{n})$

Q5. $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

$a = 2 \quad b = 2 \quad k = 1 \quad p = 1$

$a = b^k$

So, $a = b^k$

as $p = 1$ so $p > -1$

$T(n) = \theta(n^{\log_2 2} \cdot \log^2 n)$

$T(n) = \theta(n \log^2 n)$

Q6 $T(n) = 27\left(\frac{n}{4}\right) + n^{.51}$

$a = 2$; $b = 4$; $k = \cdot$; $p = 0$

$a < b^k$

So $p \geq 0$

DATE : / /
PAGE NO. :

Q10

$T(n) = \theta(n^{1.51})$

(7) $T(n) = 27\left(\frac{n}{2}\right) + n\log^{-1} n$

$a = 2 \quad b = 2 \quad k = 1 \quad p = -1$

$a = b^k$

So $p = -1$

$T(n) = \theta(n^{\log_b a} \cdot \log\log n)$

$T(n) = \theta(n^{\log_2 2} \cdot \log\log n)$

$T(n) = \theta(n \log\log n)$

(8) $T(n) = \cdot 5T\left(\frac{n}{2}\right) + \frac{1}{n}$

$a = 0.5 \qquad k = -1$

In this we can't apply master theorem as a should be $a \geq 1$ & $k \geq 0$

(9) $T(n) = 7T\left(\frac{n}{2}\right) + n^2$

$a = 7 \quad b = 2 \quad k = 2 \quad p = 0$

$a > b^k$

$T(n) = \theta(n^{\log_2 7})$

DATE : / /
PAGE NO. :

Q10 $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

$T(n) = 64T\left(\frac{n}{8}\right) + n^2 \log^{+1} n^{-1}$

$a = 64 \quad b = 8$

→ Do not follow master theorem

Q11 $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$

$a = 2 \quad b = 4 \quad k = \frac{1}{2} \quad p = 0$

as $a = b^k$

$p = 0 \qquad p > -1$

$T(n) = \theta(n^{\log_4 2} \log n)$

$\theta = (\sqrt{n} \log n)$

5/9/

$T(n) = 27\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$a = 2 \quad b = 2 \quad k = 1 \quad p = -1$

$a = b^k$

& $p = -1$

~~$T(n) = \theta(n \log$~~

$T(n) = \theta(n^{\log_2 2} \log \log n)$

$= \theta(n \log \log n)$

$T(n) = 37\left(\frac{n}{2}\right) + 2n^{1.5}$

$a = 3 \quad b = 2 \quad k = 1.5 \quad p = 0$

~~$a \not< b \quad a < b$~~ $a > b^k$

~~$T(n) = \theta(n^{1.5})$~~

$T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_2 3})$

DATE : / /
PAGE NO. :

Searching —

1) Linear Search
2) Binary Search

① Linear Search.

Time complexity in worst case = O(n)
Time Complexity in best case = O(1)
In this searching we compare the no. to be found with all the no. of the array one by one to search the location of that no.

Eg. | 5 | 3 | 7 | 8 | 9 | item = 8

a[0] == item
⋮
a[4] = item

→ initially Flag = 0

Algo →
```
for(i=0; i<n; i++)
{
    if(a[i] == item)
    {
        flag = 1;
        pos = i;
        break;
    }
}
```

DATE : / /
PAGE NO. :

(2) Binary Search

Note → This can only be implemented when the data is in sorted manner (already)

0	1	2	3	4	5	6	7	8
10	20	30	40	50	60	70	80	90

In this we search / more bi-directional

This of search follows divide & conquer approach

In this we divide the array in three indexes (low, mid, high) & compare the item to be found with the element of mid index.

L	M	H
0	4	8

Algo → while ($L \leq H$)

$M = \left\lfloor \frac{L+H}{2} \right\rfloor$

0	1	2	3	4	5	6	7	8
10	20	30	40	50	60	70	80	90

item = 80

```
if (a[mid] == item)
{
    flag = 1
    pos = mid
    break;
}
```

```
else if (a[mid] > item)
{
        H = mid - 1;
}
else {
        L = mid + 1;
    }
}
```

Now,

L	M	H	
0	4	8	$\frac{5+8}{2} = 6$
5	6	8	
7	7	8	

Time complexity will be $O(\log n)$

Algo of divide & conquer

```
Min_Max (a, i, j)
{
  if (i == j)
  {
    max = min = a[i];
    return (min, max);
  }
  else if (i = j-1)
  {
    if (a[i] < a[j])
      max = a[j];
      min = a[i];
    else
      max = a[i];
      min = a[j];
```

DATE : / /
PAGE NO. :

return (min, max);
}
else
{
mid = $\left[\frac{i+j}{2}\right]$
max_2, min_2 = DAC (a, i, mid);
max_1, min_1 = DAC (a, mid+1, j);

if ($max_1 < max_2$) then max = max_2
else max = max_1;

if ($min_1 < min_2$) then min = min_1;
else min = min_2;

return (max, min);
}
}

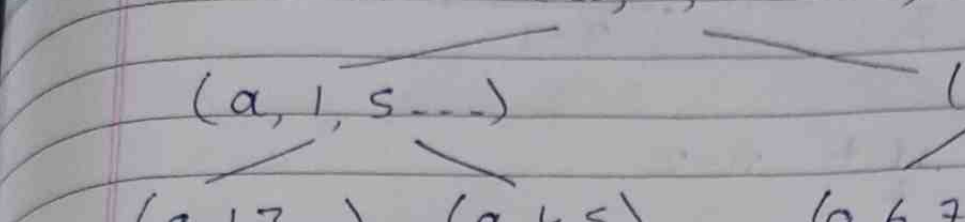
Here a = array
i = initial index value
j = last index value

DAC → Divide & Conquer.

Eg ① A[25, 30, 50, 70, 17, 23, 5, 90, 16]
1 2 3 4 5 6 7 8 9

(a, 1, 9 - -)

DATE : / /
PAGE NO. :



Q. $$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ 2T\left(\frac{n}{2}\right) + 2 & n > 2 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + 2\right] + 2$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 4 + 2$$

after k^{th} term

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + (2 + 2^2 + 2^3 + \cdots + 2^k)$$

$$\frac{n}{2^k} = 2 \quad = \quad n = 2^{k+1}$$

PAGE NO. :

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

$$= \frac{n}{2} + n - 2$$

$$= \frac{3n}{2} - 2 \rightarrow$$ Time Complexity of D & C

Fractional Knapsack

~~Greedy~~ Greedy Method

~~Algo~~ Greedy (a, n)

// a[1:n] contain n inputs

UNIT-2

DATE: / /
PAGE NO.:

Fractional Knapsack

Greedy Method

Algo Greedy (a, n)

// a[1:n] contain n inputs

{

sol^n = ∅ //

for i = 1 to n do

{

x = select (a)

if feasible (sol^n, x)

then sol^n = Union (sol^n, x)

}

return sol^n ;

}

Knapsack problem

$$\text{Max} \sum_{1 \le i \le n} p_i x_i$$

Subject to $\sum_{1 \le i \le n} w_i x_i$

and $0 \le x_i \le 1 \; ; \; 1 \le i \le n$

where p_i = profit
x_i = fraction
w_i = weight

Algo of Greedy Knapsnack

Algo Greedy Knapsnack (m, n)

// p[1:n], w[1:n] contains the profit & weight respectively of n items ordered such that $\frac{p[i]}{w[i]} \geq \frac{p[i+1]}{w[i+1]}$. M is knapsack capacity & x[1:n] is soln vector.

{
for i = 1 to n do
x[i] = 0.0; // initialize x

U = m;

for i = 1 to n do
{
if (w[i] > U) then break;
x[i] = 1.0;
U = U - w[i];
}
if (i ≤ n) then x[i] = $\frac{U}{w[i]}$
}

no. of items → Knapsack (capacity) → Profit → Weight

DATE: / /
PAGE NO.:

eg. $n = 3$; $M = 20$; $(P_1, P_2, P_3) = 25, 24, 15$
$(W_1, W_2, W_3) = 18, 15, 10$

x_1	x_2	x_3	w_1x_1	w_2x_2	w_3x_3	$\sum w_i x_i$	p_1x_1	p_2x_2	p_3x_3	$\sum p_i x_i$
1/2	1/3	6/10	9	5	6	20	12.5	8	9	29.5
0	0	0								
	1	1/2	0	15	5	20	0	24	7.5	31.5

② ③ ①
$\frac{24}{15}$, $\frac{15}{10}$, $\frac{25}{18}$

Huffman Code :—

```
Huffman Code (C)
{
    n = |C|          // C - character, n = no of diff characters
    Q = C            // Q - array / list of n characters
    for i = 1 to n-1
        allocate a new Z node
        Z.left = X = Extract-min(Q);
        Z.right = Y = Extract-min(Q);
        Z.Freq = X.Freq + Y.Freq;

        insert (Q, Z)
        return Extract-min;
}
```

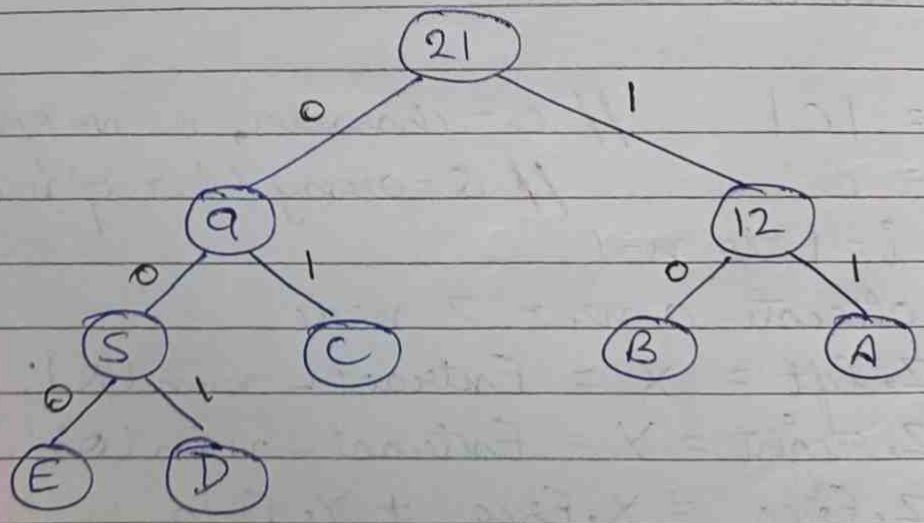
Q ABCDBBAACDDEEBAACCBAA

		Freq.	
000	A	7	→ no of repeat characters
001	B	5	
010	C	4	
011	D	3	
100	E	2	

$21 \times 3 = 63$

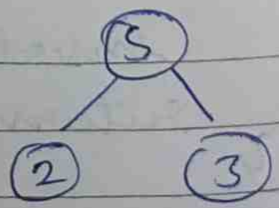
21 is the no. of characters in a string
3 is the no of bits used to transfer the data

Now construct a tree :



Step 1- freq. E + freq. D

$2 + 3 = 5$



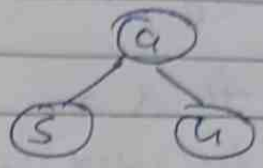
Step 2

Step 3

DATE : / /
PAGE NO. :

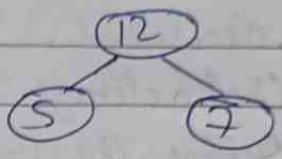
Step 2 freq. E + freq. D + freq. C

$5 + 4 = 9$



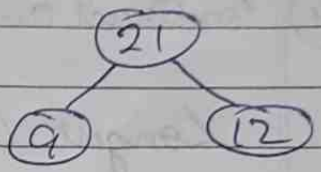
Step 3 freq. B + freq. A

$5 + 7 = 12$

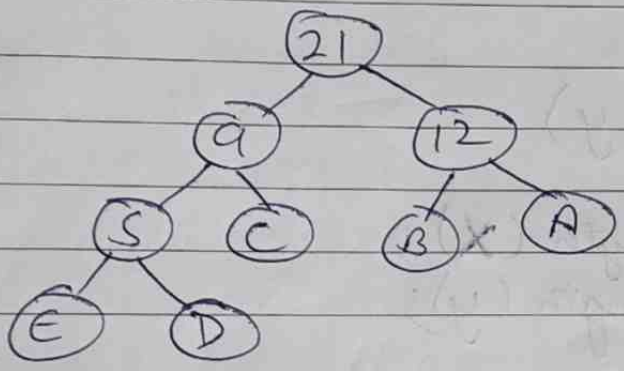


Step 4 freq. E + freq. D + freq. C + freq B + freq A

$= 9 + 12 = 21$



final –



E	000	3 × 2	= 6
D	001	3 × 3	= 9
C	01	2 × 4	= 8
B	10	2 × 5	= 10
A	11	2 × 7	= 14
			47

DATE : / /
PAGE NO. :

19/9/23 Dynamic programming Approach

It is an algorithm design method that can be used when the solu^n to a problem can be viewed as the result of a sequence of decisions. There are 4 steps -

Step ① Characterize the structure of optimal solution
Step ② Define the value of an optimal solu^n recursively
Step ③ Compute the value of an optimal solu^n in bottom-up approach
Step ④ Construct an optimal solu^n from computed info

Longest common ~~Sgt~~ Subsequence (LCS) -

Algo -

```
LCS (X, y)
{
    m = length (X);
    n = length (y);

    for i = 1 to m
        do c[i, 0] ← 0;

    for j = 0 to n
        do c[0, j] ← 0;

    for i = 1 to m
    {
        do for j ← 1 to n
        {
            do { if (Xi = yi)
```

then c[i,j] ← c[i-1, j-1] + 1;
b[i,j] ← "↖"

else if c[i-1, j] ≥ c[i, j-1]
c[i,j] ← c[i-1, j]
b[i,j] ← "↑"

else c[i,j] ← c[i, j-1]
b[i,j] ← "←"

}
}
return c & b;
}

Algo of print LCS

Print LCS (b, X, i, j)
{
if (i = 0 or j = 0)
then return;

if b[i,j] = "↖"
then print LCS (b, X, i-1, j-1);
print Xi;
else if b[i,j] = "↑"
then print LCS (b, X, i-1, j);

else print LCS (b, X, i, j-1);
}

TUT
19/9/23

Q. X → MOOR Y → MORL

	j	0	1	2	3	4
i	X	Yj	M	O	O	R
0	Xi	0	0	0	0	0
1	M	0	1↖	1←	1←	1←
2	O	0	1↑	2↖	2↖	2←
3	R	0	1↑	2↑	2↑	3↖
4	L	0	1↑	2↑	2↑	3↑

Post matric (c)

Now by print LCS i = j = 4

firstly b[i, j] = ↑

so

b, X, i-1, j

= b, X, 3, 4

Then b[i, j] = ↖

so

b, X, i-1, j-1

= b, X, 2, 3 = M

Then b[i, j] = ↖

so

b, X, i-1, j-1

= b, X, 1, 2 = O

Then b[i, j] = ←

so

b, X, i, j-1

= b, X, 1, 1 = R

DATE : / /
PAGE NO. :

Time complexity = O(m, n)

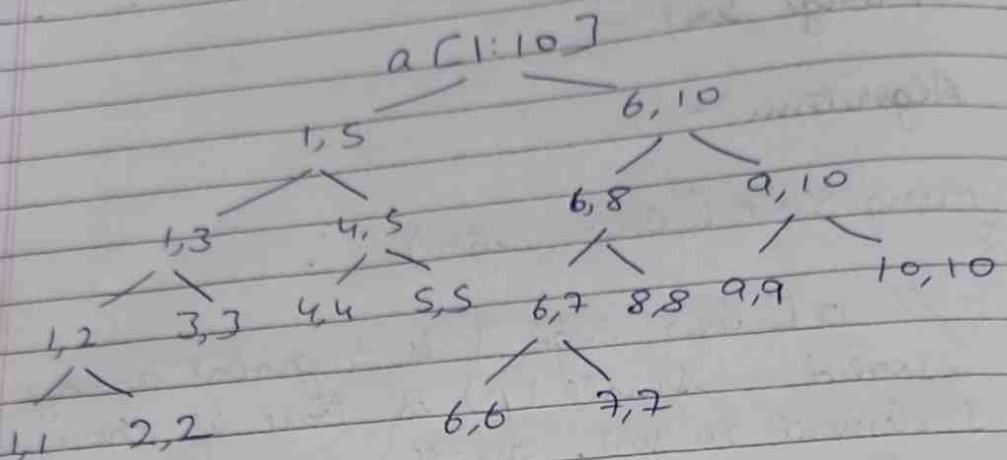
TUT
19/9/23

Merge Sort -

Algorithm -

```
MergeSort (low, high)
{
  // a[low, high] is a global array to be
  sorted. Small(p) is true if there is only
  1 element to sort. In this case this is already
    sorted

  if (low < high)   // if there are more than
                       one element
  // divide p into subproblems
  // find where to split the set

  mid = [ (low + high) / 2 ]

  // solve the subproblems

  MergeSort (low, mid);
  MergeSort (mid+1, high);
  Merge (low, mid, high);
}
```

Eg. a[1:10] = 310, 285, 179, 652, 351, 423, 861, 254, 450, 520



Algo of Merge (low, mid, high)

Merge (low, mid, high)
{
// a[low, high] is a global array containing 2 sorted subset in a[low, mid] & a[mid+1, high]. The goal is to merge these two sets into a single set residing in a[low, high]. b[ ] is an auxiliary global array

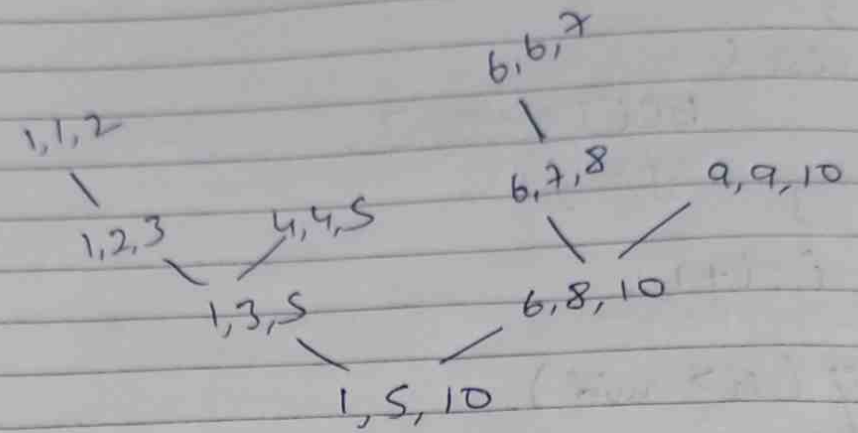
~~h = 100;~~ h = low;
~~i = 1000;~~ i = low;
j = mid + 1;
while ((h ≤ mid) and (j ≤ high))
do
{
if (a[h] ≤ a[j])
then {

```
        b[i] = a[h];
        h = h+1;
    }
    else {
            b[i] = a[j];
            j = j+1;
        }
        i = i+1;
    }
    if (h > mid)
    then
        for (k = j to high) do
        {
            b[i] = a[k];
            i = i+1;
        }
    else
        for (k = h to mid) do
        {
            b[i] = a[k];
            i = i+1;
        }
    for k = low to high
        do { a[k] = b[k];
    }
```

Eg- a[1:10]

→

Eg a[1:10]



Space complexity of ~~auxiliary array~~ merge is
= 2n

Q. a[1:10] = 310, 285, 179, 652, 351, 423, 861, 254, 450, 520

A. merge (1, 1, 2)
(l, m, h)

b | 285 | 310 | | |
1 2 3 4

from algo of merge
h = 1 ; i = 1 ; j = 1+1 = 2

while (1 ≤ 1) & (2 ≤ 2)
h ≤ mid j ≤ high

a[1] = 310 a[2] = 285

b[2] = a[1]

285, 310, 179, 652, 351, 423, 861, 254, 450, 520

New merge goes to (1, 2, 3)
(l, m, h)