UNIT-1 : FUNDAMENTALS OF ALGORITHM ANALYSIS

* Algorithm:

An algorithm is a sequence of logical steps to accomplish a specific task with the following essential properties:

(i) **Input:**

Zero or more

(ii) **Output:**

Atleast one item or element or object should be generated from the algorithm as output.

(iii) **Definiteness:**

Every line in an algorithm should be precise and unambiguous.

For example, let $y = \log_a n$ is a statement in an algorithm where a, n are integers. Then, the conditions on the variables a, n should be defined. (like 'n' should not be negative, 'a' should not be '1', etc.)

(iv) **Finiteness:**

Every algorithm must terminate after certain number of iterations.

(v) **Effectiveness:**

Every step in an algorithm should be able to carried-out with pen and paper, then only search statements will be considered as effective. Non-effective statements cannot be part of algorithm.

For example,

$$y = a^{b^{cd^{e...}}}$$

This is not an effective statement.

Scanned with OKEN Scanner

08/07/2024 Monday

* Algorithm specifications:

→ An algorithm doesnot have a specific syntax.

So, we have some general specifications. They are:

- Comments in an algorithm preceeds/~~of~~ start with '//'.
- Any block (module/sub-task) in an algorithm will be written in between '{ }'.
- Assignment of values to the ~~values~~ variables should be done using ':='.
- Multi-dimensional arrays are always indexed and accessed using '[ ]'.
- Irrespective of the language, loops in algorithms like for, while, do-while, etc. are represented as:

```
for i = V1 to Vn          (control variable; V1 = starting value, Vn = ending value)
{
    S1
    S2
    :
    Sn
}
```

```
while condition
{
    S1
    S2
    :
    Sn
}
```

- In ~~the~~ algorithms, conditional statements are specified as:

```
if condition then S1
          else S1 ----
```

- In algorithm, no need to write specific name/syntax.

```
Algorithm sum(-, -)        (keyword; sum = method-name; (-, -) = parameters)
{
    S1
    S2
    :
    Sn
}
```

* Asymptotic notation:

- Performance of an algorithm is measured using two parameters - time and space.
- Time complexity:
  ~~The time take~~ The amount of time required by an algorithm in memory.
- Space complexity:
  The amount of space required by an algorithm in memory.
- Both time and space complexities of an algorithm should be optimal.
- There are no exact or specific metrics/value available to measure performance of an algorithm.

* Asymptotic notation is a notation used to measure the performance of the algorithms. The term performance refers to time and space complexity (requirement). To measure such performance of algorithms, asymptotic notations are used.

Eg: Algorithms to find sum of 'n' natural numbers:

Algorithm-1:

```
① s=0
② for i=1 to n
③ {
④     s=s+i;
⑤ }
⑥ print s
```

'6' lines

less time and space complexities
More optimal

Algorithm-2:

```
① s=0
② i=1
③ while i<=n
④ {
⑤     s=s+i
⑥     i=i+1
⑦ }
⑧ print s
```

'8' lines

More time and space complexities
less optimal

Scanned with OKEN Scanner

* There are three types in asymptotic notation. They are:

- Big-O notation (O)
- Theta notation (Θ)
- Omega notation (Ω)

* Big-O notation:

Let, $f(n), g(n)$ be two functions

$f(n) \leq cg(n)$. $\forall$ $n \geq n_0$ for all 'c' is a constant,

then, $\boxed{f(n) = O(g(n))}$

Eg: (i) $f(n) = 5n$ → function for liner of code

$5n \leq \frac{6}{c} \frac{n}{g(n)}$

$\therefore \boxed{f(n) = O(n)}$

(ii) $f(n) = 3n+4$

$3n+4 \leq \frac{8}{c} \frac{n}{g(n)}$ → $(3+4=7 \Rightarrow 8>7$, i.e, $8=7+1)$

$\therefore \boxed{f(n) = O(n)}$

(we should take a minimal no which is greater than or equal to sum of coefficients (i.e, '7' here))

09/07/2024 Tuesday

* Big-O notation:

Let, $f(n)$ and $g(n)$ are two functions. If we can write $f(n) \leq c \cdot g(n)$ $\forall$ $n \geq n_0$ and $n \in z$ where c, n_0 are arbitrary constants, then

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

Consider the following polynomials and represent in Big-O notation.

Eg: (i) $f(n) = 2n^2+3n+2$

$2n^2+3n+2 \leq 8.n^2$, $n \geq 1$

$\hookrightarrow [8 > 7 (= 2+3+2)]$

$c = 8$

$n_0 = 1$

$\therefore f(n) = 2n^2+3n+2 = O(n^2)$

Scanned with OKEN Scanner

(ii) $f(n) = 6n+20$

$6n+20 \leq (26).n \quad (=20+6), \; n \geq 1$

$\therefore f(n) = 6n+20 = O(n)$

(iii) $f(n) = 5n^3+4n^2+2n+6 \quad [=5+4+2+6]$

$5n^3+4n^2+2n+6 \leq (17).n^3, \; n \geq 1$

we can also take any number >17.

$\therefore$ ~~$O(n)$~~ $\boxed{f(n) = O(n^3)}$

* For a polynomial, $f(n) = a_n n^m + a_{n-1} n^{m-1} + a_{n-2} n^{m-2} + \ldots + a_0$, we can write $f(n) = O(n^m)$.

* Any polynomial of order 'm' can be expressed as $O(n^m)$.

* Theta Notation (θ):

Let, $f(n)$ and $g(n)$ are two functions, if we can write $c1.g(n) \leq f(n) \leq$ ~~$g(n)$~~ $c2.g(n) \; \forall \; n \geq n_0$, where c, n_0 are two arbitary constants, then, $f(n) = \theta(g(n))$.

Eg: Consider the following polynomials and express it in θ-notation

(i) $f(n) = 2n^2+3n+2$

$2n^2 \leq 2n^2+3n+2 \leq 9n^2, \; n \geq 1$

$c1 = 2$

$c2 = 9$

$g(n) = n^2$

$n_0 = 1$

$\therefore \boxed{f(n) = \theta(n^2)}$

Scanned with OKEN Scanner

(ii) $f(n) = 6n + 20$

$6n \leq 6n + 20 \leq 26n \quad , n \geq 1$

$c_1 = 6$

$c_2 = 26$

$n_0 = 1$

$g(n) = n$

$\therefore \boxed{f(n) = \Theta(n)}$

(iii) $f(n) = 5n^3 + 4n^2 + 2n + 6$

$5n^3 \leq 5n^3 + 4n^2 + 2n + 6 \leq 17.n^3 \quad , n \geq 1$

$c_1 = 5$

$c_2 = 17$

$n_0 = 1$

$g(n) = n^3$

$\therefore \boxed{f(n) = \Theta(n^3)}$

* Omega Notation (Ω):

Let, $f(n)$ and $g(n)$ are two functions, if we can write $f(n) \geq c.g(n) \ \forall \ n \geq n_0$, where c, n_0 are two arbitary constants, then, $f(n) = \Omega(g(n))$.

Eg: Consider the following polynomials and express them in Ω-notation.

(i) $f(n) = 2n^2 + 3n + 2$

$2n^2 + 3n + 2 \geq n^2$

and also, $2n^2 + 3n + 2 \geq n$

But, we have to choose the maximal one.

So, $2n^2 + 3n + 2 \geq n^2 \quad , n \geq 1$

$\therefore \boxed{f(n) = \Omega(n^2)}$

Scanned with OKEN Scanner

The above polynomial can be expressed as $\Omega(n^2)$, $\Omega(n)$ also, according to the definition, but among them, maximal is $\Omega(n^2)$. Hence, given polynomial is equal to $\Omega(n^2)$.

(8) ~~$f(n) = 6(2n + n^2)$~~, express it in O, Θ notations.

$f(n) = 6.2^n + n^2$

Sol: $6.2^n + n^2 \leq 7.n^2$

$n=1 \Rightarrow 13 \leq 7$ ✓

$n=2 \Rightarrow 28 \leq 28$ ✓

$n=3 \Rightarrow 57 \leq 63$ ✓

$n=4 \Rightarrow 112 \leq 112$ ✓

$n=5 \Rightarrow 217 \leq 175$ ✗

So, lets try $6.2^n + n^2 \leq 7.2^n$

$n=1 \Rightarrow 13 \leq 14$ ✓

$n=2 \Rightarrow 28 \leq 28$ ✓

$n=3 \Rightarrow 57 \leq 56$ ✗

Now, let's try with $\boxed{6.2^n + n^2 \leq 10.2^n}$

$n=1 \Rightarrow 13 \leq 20$ ✓

$n=2 \Rightarrow 28 \leq 40$ ✓

$n=3 \Rightarrow 57 \leq 80$ ✓

$n=4 \Rightarrow 112 \leq 160$ ✓

$\therefore \boxed{f(n) = O(2^n)}$

Scanned with OKEN Scanner

* Plotting of time and space complexities using graph:

Eg: Sample dataset : ~~f(n) = n~~

f(n) =>

n	$\log_2 n$	n^2	nlogn
2	1	4	2
4	2	16	8
8	3	64	24
16	4	256	64
32	5	1024	160

10/07/2024 Wednerday
10M may come

* Merge Sort:

- This algorithm is a recursive procedure or module with two arguments, i.e, index of the first element, last element namely low, high.

```
MergeSort(low, high)
{
    if (low < high)
    {
        mid = ⌊(low + high)/2⌋;
        Mergesort(low, mid);
        Mergesort(mid+1, high);
        Merge(A, low, mid, high);
    }
}
```

Scanned with OKEN Scanner

- The algorithm Merge will take two sorted sub-arrays and combines them into single sorted array.

```
Merge (A, low, mid, high)
{
    h=low;
    i=low;
    j=mid+1;
    while( low(h≤mid) and (j≤high))
    {
        if (a[h] ≤ a[j])
        {
            b[i]=a[h];
            h=h+1;
        }
        else
        {
            b[i]=a[j];
            j=j+1;
        }
        i=i+1;
    }
    i=i+1;
}
```

15/07/2024 Monday

* Performance Analysis:

- Two ~~measure or~~ analyze the performance of an algorithm, time and space complexity are the metrics.
- Algorithm performance can be estimated prior (or priori) or post (or posterior).
- For example, consider an algorithm for finding sum of 'n' natural

Scanned with OKEN Scanner

numbers, In a non recursive and recursive versions can be written as follows:

Non-recursive version:

```
sum(a,n)
{
  s=0.0
  for i=1 to n do
    s=s+a[i]
  return s
}
```

S/E	Frequency	Total steps
0	0	0
0	0	0
1	1	1
1	n+1	n+1
1	n	n
1	1	1
0	0	0
		2n+1

S/E → steps for execution, that means no. of steps required by the control inorder to execute that particular statement.

Frequency → Total no. of times or iterations that can be made by the statements.

Total no. of steps → It is the product of S/E and frequency.

∴ The time complexity of above algorithm = 2n+3

'for' & 's=s+a[i]' lines, Each lines need 'n' steps

One step each for s, n, i initialization

Recursive version:

Consider the recursive version of 'n' numbers.

Rsum(a,n) → It is the recursive sum

```
RSum(a,n)
{
    if (n<=0) then
        return;
    else
        return RSum(a,n-1)+a[n];
}
```

S/t	Frequency		Total steps	
	n=0	n>0	n=0	n>1
—	—	—	0	0
—	—	—	0	0
1	1	1	1	1
—	—	—	0	0
1	1	1-x	1	1+x
			2	2+x

$$\therefore t_{RSum} = \begin{cases} 2 & , \text{if } n=0 \\ 2+x & , \text{if } n>0 \end{cases}$$

16/07/2024 Tuesday

Q) Evaluate the time complexity of the following algorithms:

(1)
```
A()
{
    int i,j,n,k;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            for(k=1;k<=100;k++)
            {
                printf("CSE-A");
            }
        }
    }
}
```

Sol: n=5

i=1	i=2	i=3	i=4	i=5
j=1	j='2' times	j='3' times	j='4' times	j='5' times
100	2×100	3×100	4×100	5×100

Scanned with OKEN Scanner

$100 + 2\times100 + 3\times100 + 4\times100 + 5\times100$

$= 100(1+2+3+\cdots+n)$

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

$= 50(n^2+n)$

$= 50n^2 + 50n$

$\therefore$ Time complexity $= O(n^2)$

(iii)
```
A()
{
    int i,j,k,n;
    for(i=1 ; i<=n/2 ; i++)
    {
        for (j=n/2 ; j<n ; j++)
        {
            for (k=1 ; k=n/2 ; k++)
            {
                pf("CSE-A")
            }
        }
    }
}
```

Sol: $n = 16$

$i = 1$ to 8 → $i=1$ $i=2$ $i=3$ ... $i=8$

$j = 8$ to 16 $j=8$ time $j=8$ times $j=8$ times ... $j=8$ times

$k = 1$ to 8 $k=8$ time $k=8$ times $k=8$ times ... $k=8$ times

$n=16$ $n=32$ $n=64$ ... n

$8\times8\times8$ → 'k' loop '8' times $16\times16\times16$ $32\times32\times32$ ... $\frac{n}{2}\times\frac{n}{2}\times\frac{n}{2}$

'i' loop '8' times, 'j' loop '8' times

$\frac{n^3}{8}$ ⇒ Time complexity $= O(n^3)$

(iii) A()

```
{
    for (i=1; i<=n; i=2*i)
    {
        Pf("CSE-A");
    }
}
```

Sol: n=16

i=1	2	3	.....	k
2	4	6	.....	2k

$2k > n \Rightarrow$ loop terminates

$k > \frac{n}{2}$

$O(n/2)$

∴ Time complexity = $O(n)$

Pg-58 TB

a) Find out the time complexity of the following algorithm ~~for i=1 to n~~ for i = 1 to n ~~do~~

```
{
    for j=1 to i do
    {
        for k=1 to j do
        {
            Pf("CSE-A")
        }
    }
}
```

Sol: $i=1$ $i=2$ (1' time) $i=3$ ..... $i=n$

$j=1$ time $j=$ '2' times $j=$ '3' times ..... $j=$ 'n' times

$k=1$ time $k=$ '4' times $k=$ '9' times ..... $k=$ 'n²' times

$n=8$ $n=16$ $n=32$ ..... n

$8\times8\times8$ $16\times16\times16$ $32\times32\times32$ ..... $n\times n\times n$

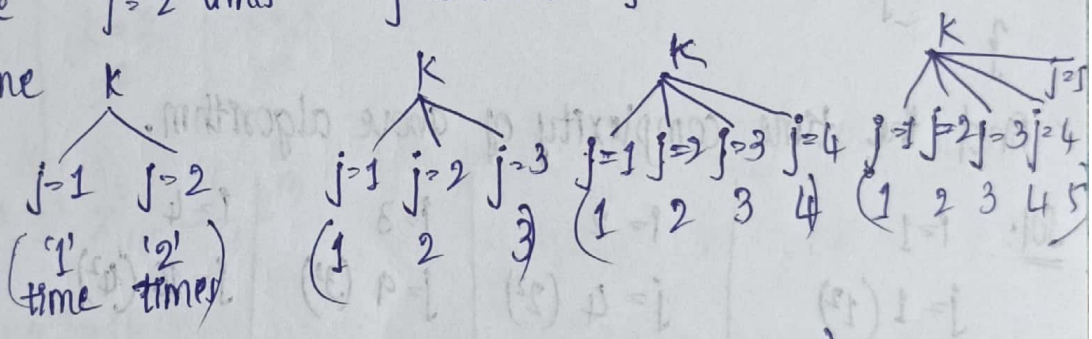
$\therefore$ Time complexity $= O(n^3)$

$n=5$

$i=1$ $i=2$ $i=3$ $i=4$ $i=5$

$j=$ '1' time $j=$ '2' times $j=$ '3' times $j=$ '4' times $j=$ '5' times

$k=$ '1' time



$1+(1+2)+(1+2+3)+\underbrace{(1+2+3+4)}_{\text{sum of } (n-1) \text{ terms}}+\underbrace{(1+2+3+4+5)}_{\text{sum of 'n' terms}}$

22/07/2024 Monday

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

~~$\therefore$ Time complexity $= O(n^3)$~~

$\sum \frac{n(n+1)}{2}$

$= \sum \frac{n^2}{2} + \sum \frac{n}{2}$

$= \frac{1}{2}\left(\sum n^2 + \sum n\right) = \frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right]$

~~The qtv~~ $\therefore$ Time complexity $= O(n^3)$

$\therefore$ The given algorithm is order n^3.

Scanned with OKEN Scanner

Q)
```
A()
{
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=i^2; j++)
        {
            for(k=1; k<=n/2; k++)
            {
                pf("CIE-A");
            }
        }
    }
}
```

Find the time complexity of above algorithm.

Sol:

$i=1$	$i=2$	$i=3$	$i=4$
$j=1\ (1^2)$	$j=4\ (2^2)$	$j=9\ (3^2)$	$j=16\ (4^2)$
$k=\frac{n}{2}*1^2$	$k=\frac{n}{2}*2^2$	$k=\frac{n}{2}*3^2$	$k=\frac{n}{2}*4^2$

$$= \frac{n}{2}*1^2 + \frac{n}{2}*2^2 + \frac{n}{2}*3^2 + \frac{n}{2}*4^2 + \cdots$$

$$= \frac{n}{2}\left[1^2+2^2+3^2+4^2+\cdots\right]$$

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

$$\cong O(n^4)$$

Q) Find time complexity of below algorithm:

```
A()
{
    for(i=n/2; i<=n; i++)
    {
```

```
for(i=1; i<=n/2; i++)
{
    for(k=1; k<=n; k=k*2)
    {
        printf("Crt-A");
    }
}
}
```

Sol: k-loop:

k=1,	k=2	k=3
2^1	2^2	2^3

$1, 2^1, 2^2, 2^3, \dots, 2^m$

loop works upto : $2^m = n$

$\Rightarrow m = \log_2 n$

$i \rightarrow \frac{n}{2}$ times

$j \rightarrow \frac{n}{2}$ times for each i

$k \rightarrow \log_2 n$

$= \frac{n}{2} \times \frac{n}{2} \times \log_2 n$

$= \frac{n^2}{4} \log_2 n$

$= \frac{1}{4} \times n^2 \log_2 n$

∴ Time complexity = $O(n^2 \log n)$

Scanned with OKEN Scanner

Q) Find time complexity of below algorithm:

```
A()
{
    for(i=1; i<=m; i++)          ----> (m+1)
    {
        for(j=1; j<=n; j++)      ----> m*(n+1)
        {
            c[i,j] = a[i,j]+b[i,j];   ----> mn
            // a,b,c are matrices of order m x n
        }
    }
}
```

Calculate in how many no. of steps the algorithm will be executed and also the time complexity.

Sol: In the above algorithm, while calculating steps executed by each line.

In for loop,

In general, control variable running upto to last value+1.

That means when the control variable reach beyond the specified limit we will exit the loop.

Similarly, the inner for loop will be running.

∴ Total no. of steps taken by the control to complete the algorithm is minimum $(2mn+2m+1)$ indicating the time complexity $O(mn)$.

$$(m+1)+[m*(n+1)]+mn$$
$$= m+1+mn+m+mn$$
$$= 2mn+2m+1$$

Scanned with OKEN Scanner

24/07/2024 Wednesday

Q) Find the time complexity of the following iterative algorithm using asymptotic notation:

```
A()
{
  int i,j;
  for(i=1 ; i<=n ; i++)
  {
    for(j=1 ; j<=n ; j++)
    {
      printf("CSE-A");
    }
  }
}
```

Sol:

i=1	i=2	i=3
j → 'n' times	j → 'n/2' times	j → 'n/3' times

$$n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{k}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k}\right)$$

$$= n \log n$$

$$\Rightarrow O(n \log n)$$

Q) Find the time complexity of the following iterative algorithm:

```
A()
{
  i=1, j=1;
  while (s<=n)
  {
```

```
        i++;
        s=s+i;
        pf("CSE-A");
    }
}
```

Sol:

i=1	i=2	i=3	i=4
s=2	s=4	s=7	s=11
	2+2	2+2+3	2+2+3+4
	s+i	s+i	s+i

$s > n$

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

$k^2 + k > 2n$

$k^2 > 2n - k$

$k > \sqrt{2n-k}$

$\therefore$ Time complexity $= O(\sqrt{n})$

Assume that the loop will be iterating upto 'k' times. After 'k' no. of iterations, the value of 's' will be (sum of first 'k' natural numbers + 1). As '1' is a constant, we can ignore in evaluation of time complexity and therefore at kth iteration the value of 's' can be taken as $\frac{(k+1)(k+2)}{2}$.

The above algorithm will terminate when this value of 's' becomes '>n'. Therefore, $\frac{k(k+1)}{2} > n$ is the terminating condition. ('k' is the no. of times/iterations algorithm is running). Therefore,

Scanned with OKEN Scanner

The time complexity of above algorithm can be written as $O(\sqrt{n})$.

* Recursive algorithm:

Eg: Estimate the time complexity of the following recursive algorithm:

```
A(n)
{
    if (n>1)
        A(n/2) + A(n/2);
}
```

Sol:

$$T(n) = 2T\left(\frac{n}{2}\right) \quad n>1$$

$$= c \quad n=1$$

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

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

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

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

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

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

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

Scanned with OKEN Scanner

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

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

$n = 2^k$

$= 2^k T(1) + C(1+2+2^2+\dots+2^k)$

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

$= 2^k C + C(2^k-1)$

$= C(2^{k+1}-1)$

$= C(2n-1)$ $\quad [\because 2^k = n \Rightarrow 2^{k+1} = 2n]$

$\therefore$ Time complexity $= O(n)$

Q) Consider the following recurrence relation to find out the time complexity of an algorithm and express it using asymptotic notation.

Sol: $T(n) = n + T(n-1) \quad n > 1$

$= 1 \quad n = 1$

$T(1) = 1$

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

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

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

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

Scanned with OKEN Scanner

Terminating condition of the

$n-(k+1) = 1$

$n-k-1 = 1$

$n-k = 2$

$k = n-2$

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

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

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

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

∴ Time complexity $= O(n^2)$

Q) $T(n) = 1 + T(n-1)$, $n>1$

$= 1$, $n=1$

Find out time complexity using Big-O notation.

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

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

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

$= 1+1+1+ \cdots +1+T(k)$

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

Terminating condition is:

$n-(k+1) = 1$ $\quad$ $n-k = 1$

$n-k-1 = 1$ $\quad$ $k = n-1$

$k = n-2$

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

$= k(1) + T(1)$

$= k+1$

$= n-1+1$

$= n$

$\therefore$ Time complexity $= O(n)$

Q) Estimate time complexity of:

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

Assume $n = 2^k$ ← If not mentioned in question assume suitable one)

Sol:

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

$$= 2\left[2T\left(\frac{n}{2^2}\right) + c.\frac{n}{2}\right] + cn$$

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

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + c.\frac{n}{4}\right] + cn$$

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

At k^{th} iteration,

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

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

$$= n.T(1) + 2^k.kcn$$

Scanned with OKEN Scanner

$= n(a) + 2^k \cdot kcn$

$= na + 2^k kcn$

$= na + \log_2 n \cdot cn$

$= an + cn\log_2 n$

$\therefore$ Time complexity $= O(n \log n)$

26/07/2024 Friday

10M Ques

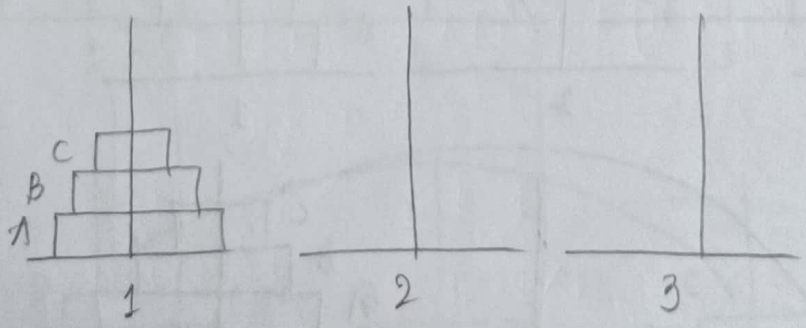
Towers of Hanoi Algorithm:

- There are 'n' no. of discs given and '3' towers - source, destination, auxiliary were available.
- These 'n' no. of discs are arranged on source tower in the decreasing order of their size.
- The problem is to move these 'n' discs on to destination tower and arrange them as in source tower subjected to the following condition/constraints:

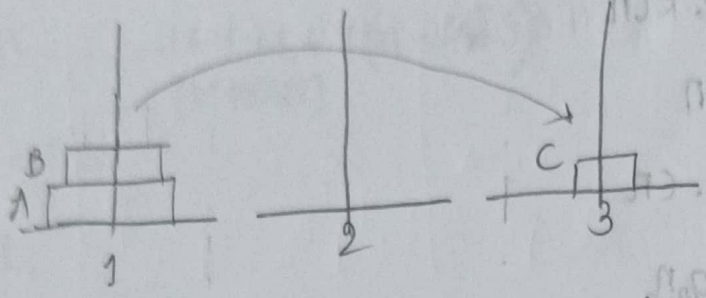
(i) At a time only one disc is allowed to move.

(ii) A smaller disc cannot be under a bigger disc/a bigger disc cannot be over a smaller disc.

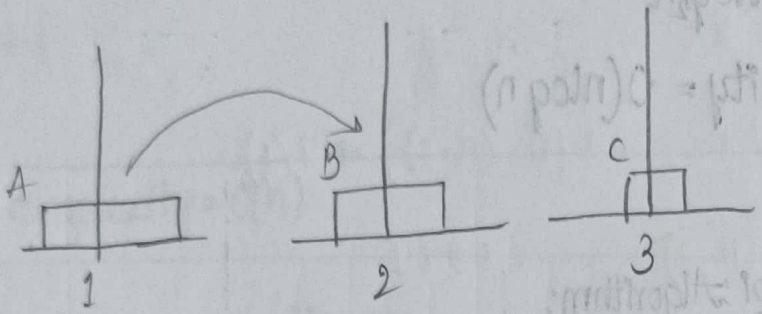
For example, let $n=3$, then the no. of moves are as follows:



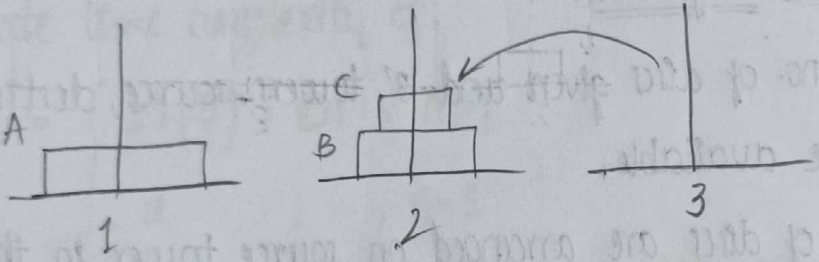
①



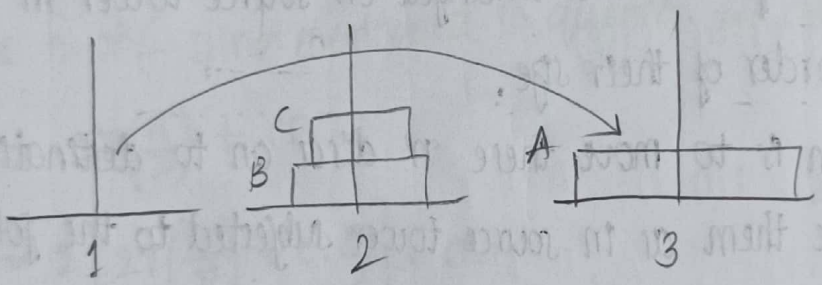
②



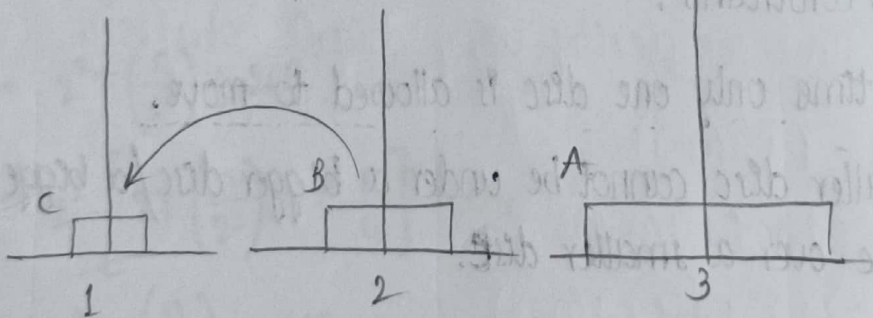
③



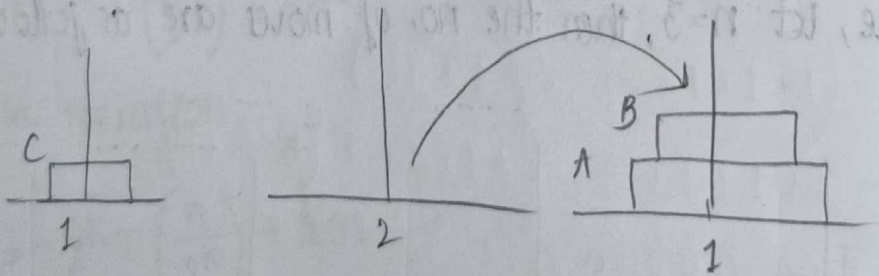
④



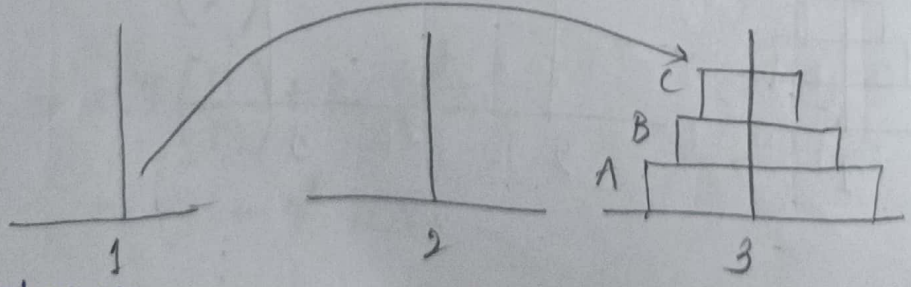
⑤



⑥

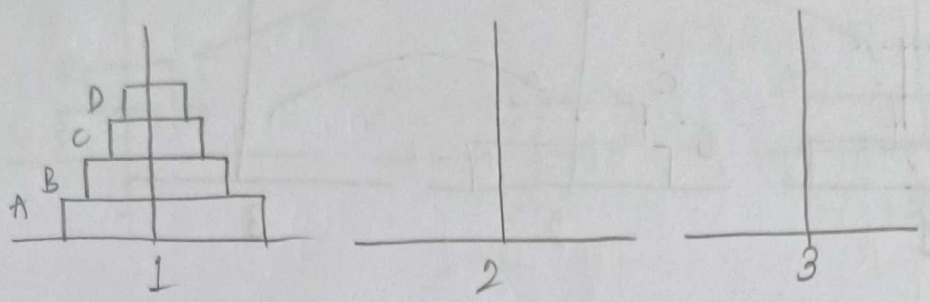


⑦

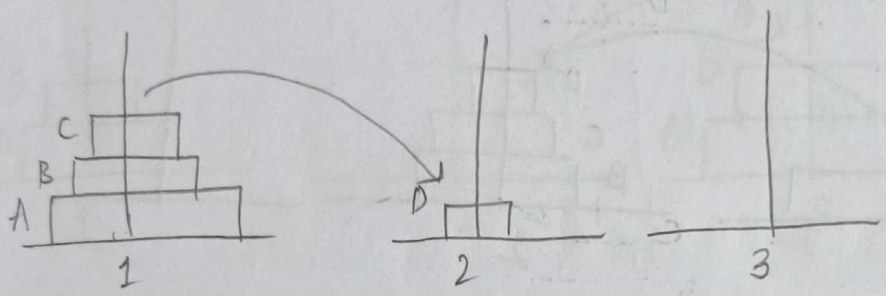


∴ Total '7' moves.

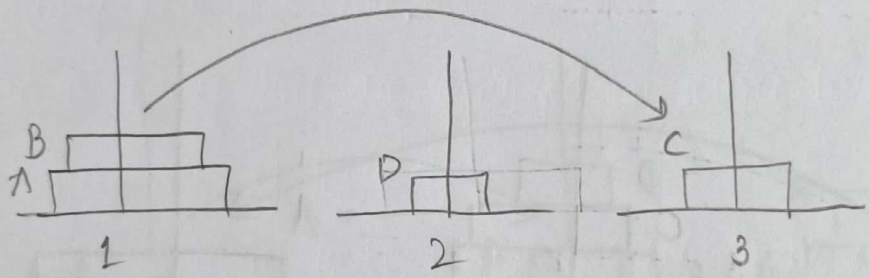
Eg: If n=4



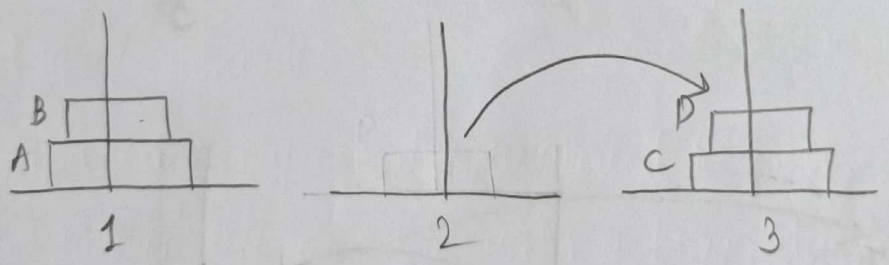
①



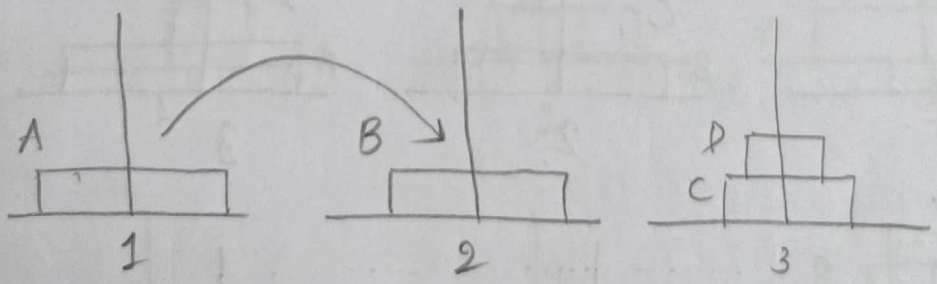
②



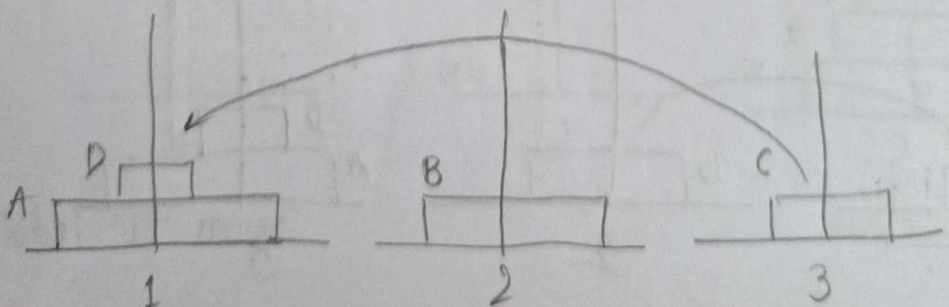
③



④

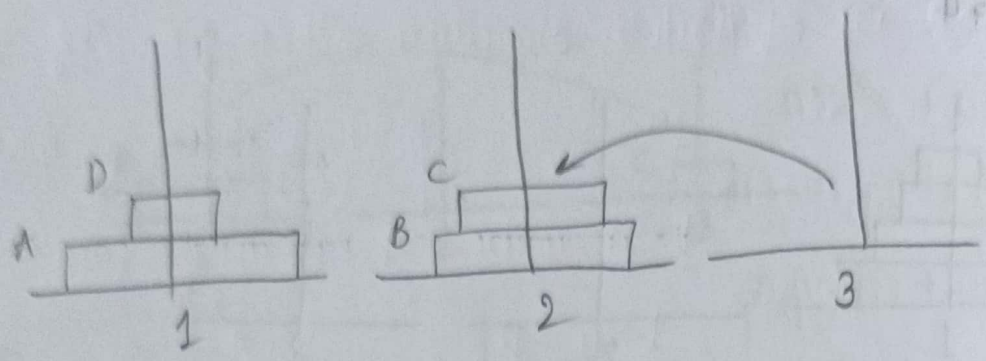


⑤

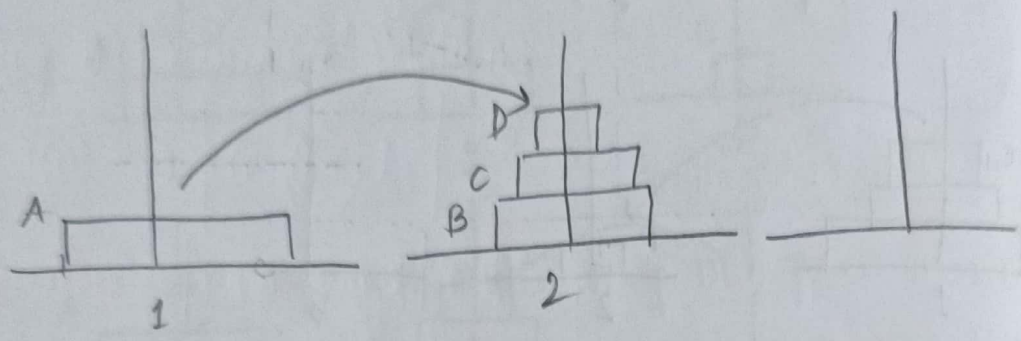


Scanned with OKEN Scanner

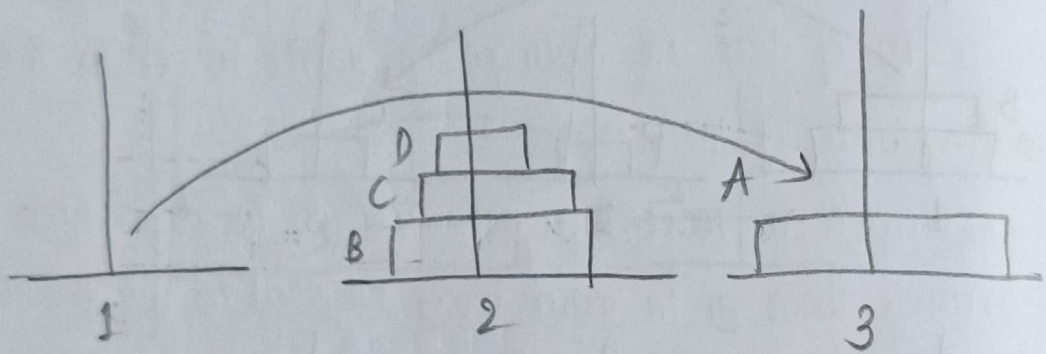
⑥



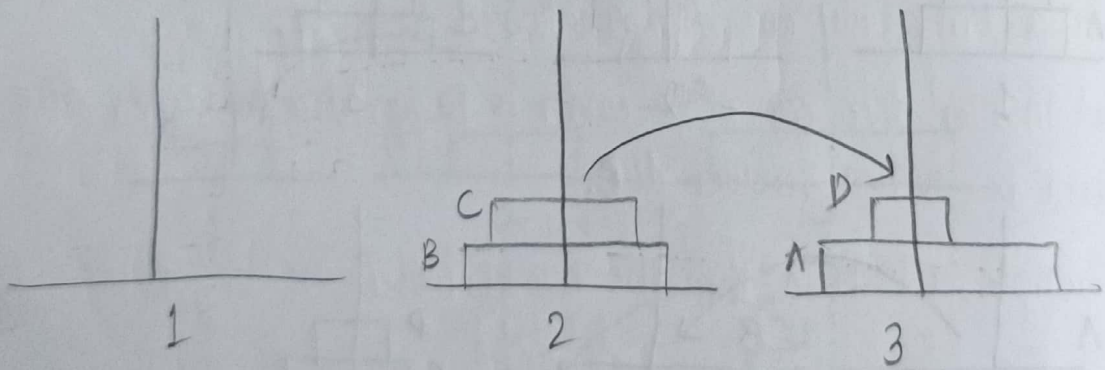
⑦



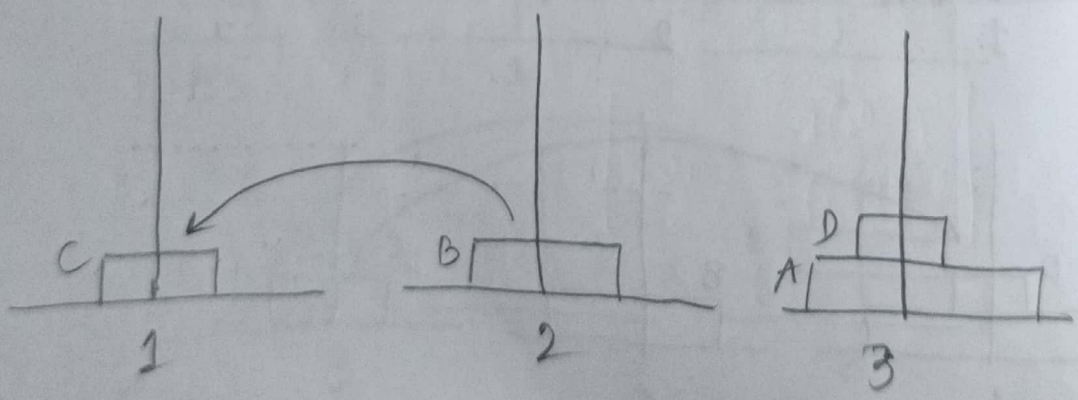
⑧



⑨

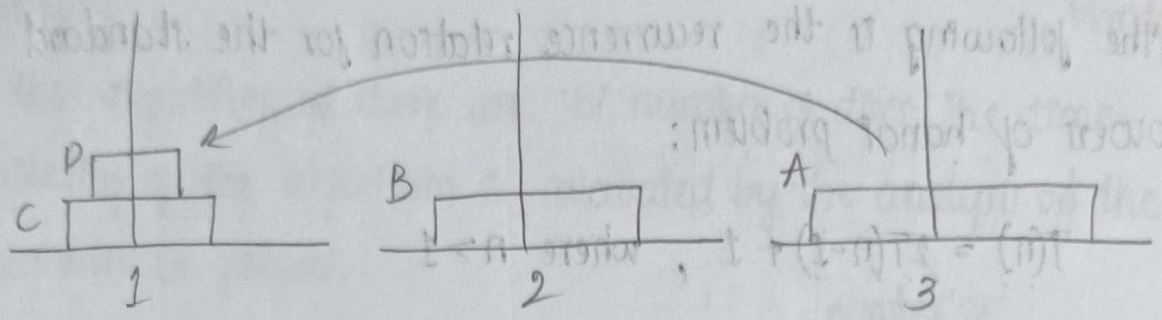


⑩

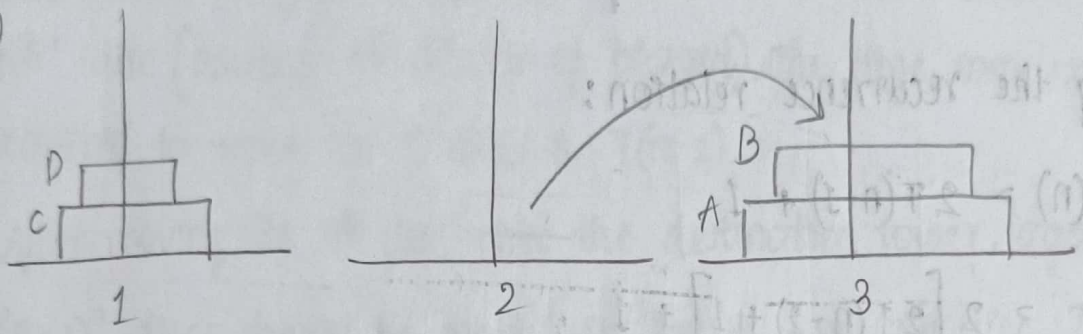


Scanned with OKEN Scanner

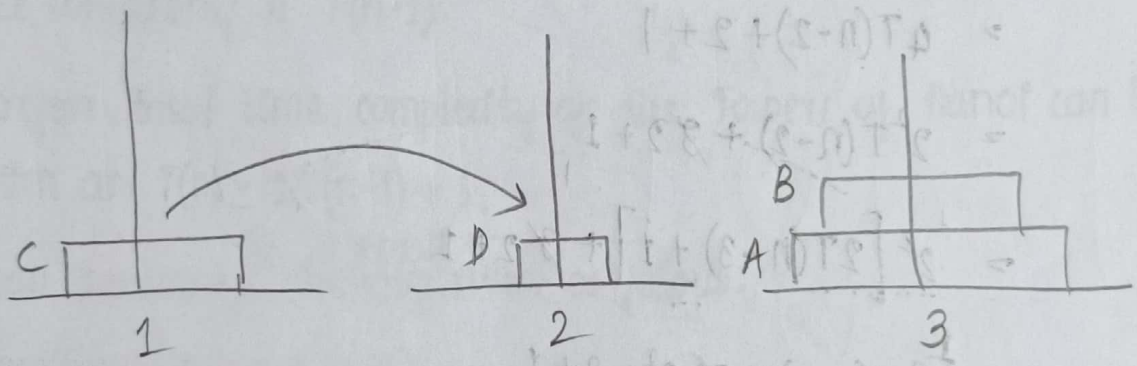
(11)



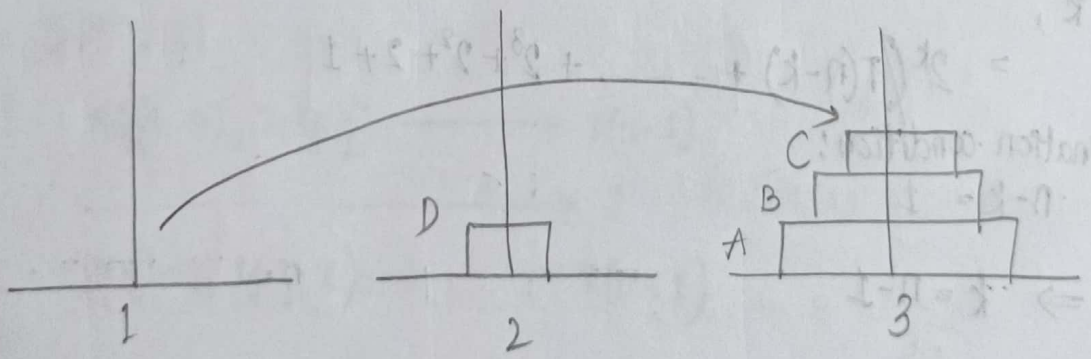
(12)



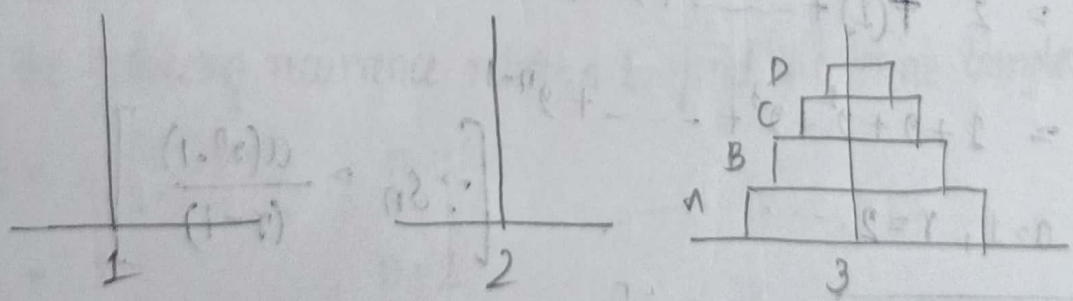
(13)



(14)



(15)



∴ Total '15' moves.

Scanned with OKEN Scanner

* The following is the recurrence relation for the standard towers of hanoi problem:

$$T(n) = 2T(n-1) + 1, \text{ where } n > 1$$
$$= 1, \text{ if } n = 1$$

Solving the recurrence relation:

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

for 'k',

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

Termination condition:

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

$a = 1, r = 2$

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

$$\therefore \boxed{T(n) = 2^n - 1}$$

Scanned with OKEN Scanner

29/07/2024 Monday

- In this algorithm, let there are 'n' number of discs. The time complexity of the algorithm is calculated by the analysis of the algorithm as follows.

That means, the time required to move 'n-1' number of discs above the 'nth' disc (assume nth disc is of biggest). The time complexity required to move '(n-1)' discs is T(n-1).

After moving the nth disc onto the destination tower, again '(n-1)' discs should be moved on top of the nth disc. This time complexity is T(n-1).

Therefore, total time complexity of the Towers of Hanoi can be written as $T(n) = 2T(n-1) + 1$.

We can summarize the recursive algorithm as follows:

```
Algorithm TH(n, x, y, z)
{
    if(n >= 1)
        TH((n-1), x, y, z) ------> T(n-1)
        ----------------   ------> 1
        TH:(n-1, y, z, x) ------> T(n-1)
}
```

Q) Solve the following recurrence relation to find the time complexity:

$$T(n) = 2T\left(\frac{n}{2}\right) + 3 \quad , \quad n > 2$$

$$= 2 \quad , \quad n = 2$$

Scanned with OKEN Scanner

Sol:

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

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

$$= 2^2 T\left(\frac{n}{2^2}\right) + 6 + 3$$

$$= 2^2 T\left(n/2^2\right) + 3^2$$

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

$$= 2^3 T\left(n/2^3\right) + 12 + 9$$

$$= 2^3 T\left(n/2^3\right) + 21$$

$$= 2^3\left[2T\left(\frac{n}{2^4}\right) + 3\right] + 21$$

$$= 2^4 T\left(\frac{n}{2^4}\right) + 45$$

For 'k' iterations,

$$= 2^k T\left(\frac{n}{2^k}\right) + 3\underbrace{\left[1 + 2^1 + 2^2 + 2^3 + \cdots + 2^{k-1}\right]}_{\text{'k' terms}}$$

[∵ Last term in G.P. $t_n = ar^{n-1}$]

Terminating condition is,

$$\frac{n}{2^k} = 2 \Rightarrow n = 2^{k+1} \Rightarrow k+1 = \log_2 n \Rightarrow k = \log_2 n - 1$$

$$\Rightarrow \frac{n}{2} T(2) + 3\left[\frac{1(2^k - 1)}{(2-1)}\right]$$

$$= \frac{n}{2}(2) + 3(2^k - 1)$$

$$= n + 3\left(\frac{n}{2}\right) - 3 = \frac{5n}{2} - 3$$

∴ Time complexity = $O(n)$

Scanned with OKEN Scanner

Q) $T(n) = 2T\left(\frac{n}{2}\right) + C \quad , \quad n > 1$

$= C \quad , \quad n = 1$

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

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

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

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

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

For 'k' Iterations,

$= 2^k T\left(\frac{n}{2^k}\right) + C\left[1 + 2 + 2^2 + \ldots + 2^{k-1}\right]$

Terminating condition is,

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

$\Rightarrow \quad n T(1) + C\left[\frac{1(2^k - 1)}{(2-1)}\right]$

$= n(C) + C\left[2^k - 1\right]$

$= cn + C[n-1]$

$= cn + cn - C$

$= 2cn - C = c[2n-1]$

∴ Time complexity $= O(n) = \theta(n) = \Omega(n)$

31/07/2024 Wednesday

Q) Show that the following equalities are correct.

(i) $\frac{6n^3}{\log n + 1} = O(n^3)$

(ii) $n^{1.0001} + n\log n = \Theta(n^{1.001})$

Sol: (i) $\frac{6n^3}{\log n + 1} \leq 3.n^3$

$c = 3,\ g(n) = n^3 \ \forall\ n \geq 2$

$n_0 = 2$

$n = 1$	$\frac{6n^3}{\log n+1} = 24$
$n = 2^2$	128
$n = 2^3$	768

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

$$\frac{6n^3}{\log n + 1} = O(n^3)$$

(ii) $n^{1.0001} + n\log n \leq n^{\log n}$

$n = 10$: $10^{1.0001} + 10\log_{10} 10 = 10 + 10^{1.0001} \leq 10^{1.001} + 10^{\log_{10} 10}$

$n = 10^2$: $(10^2)^{1.0001} + 10^{2\log_{10} 10^2}$

$n = 10$: $10^{1.001} \leq 10 + 10^{1.001} \leq 10 \times 10^{1.001}$

$10.02 \leq 20.02 \leq 100.2$

$n = 10^2$: $100^{1.001} \leq 100^{1.001} + 200 \leq 100 \times 100^{1.001}$

$100.46 \leq 300.46 \leq 1004.6 \ \forall\ n \geq 10$

$c_1 = 1,\ c_2 = 10$

Scanned with OKEN Scanner

31/07/2024 UNIT-II: Wednesday

DIVIDE AND CONQUER

- To solve any given problem 'p', first we try to solve it directly. If it is bigger than the usual, then, we try to subdivide 'p' into sub-problems or sub-tasks and solve the sub-problems solutions into required solution of the problem.

- The control strategy or abstraction of divide and conquer approach is written as follows:

~~DAnd~~ Let 'p' is the given problem.

```
DAndC(p)
{
   if p' is small then return S(p)
   else
      divide p' into sub-problems
        T1, T2, ....., Tn and solve
      combine (DAndC(T1), DAndC(T2), DAndC(T3), ....)
}
```

- The time complexity of the divide and conquer strategy can be written as following:

$$T(n) = \begin{cases} g(n) & \text{where 'p' is small} \\ T(n_1)+T(n_2)+\dots+T(n_x)+f(n), & \text{otherwise} \end{cases}$$

where 'g(n)' is a small time complexity, generally a constant.
'f(n)' is the time required to divide 'p' into sub-tasks and combine solutions of subtasks to get the final solution/answer of 'p'

• Generally, time complexity of divide and conquer is expressed as

$$T(n) = \begin{cases} T(1) & , n=1 \\ aT(n/b) + f(n) & , n>1 \end{cases}$$

where, 'n' is assumed as 'b^k'

Eg: $a=2, b=2, f(n)=n, T(1)=1$

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

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

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

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

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

For 'k' iterations,

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

Terminating condition is $T(1)=1$.

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$$

$$\Rightarrow nT(1) + n\left[\frac{1(2^k - 1)}{(2-1)}\right] = nT(1) + nk$$

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

$$= n[1 + \log n]$$

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

$$= n + n^2 - n$$

$$= n^2$$

Eg: $a=2, b=2, f(n)=n^2, T(1)=1$

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

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

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

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

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

For 'k' iterations,

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

Terminating condition:

$$T(1) = 1 \Rightarrow \frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$$

$$\Rightarrow 2^k T(1) + n^2\left[\frac{1\left(1 - \frac{1}{2^k}\right)}{\left(1 - \frac{1}{2}\right)}\right]$$

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

$$\Rightarrow n + 2n^2 - 2n$$

$$= 2n^2 - n$$

$\therefore$ Time complexity $= O(n^2)$

Scanned with OKEN Scanner

02/08/2024 Friday

* Time complexity of divide and conquer is:

$$\boxed{T(n) = n^{\log_b a}\,[T(1) + u(n)]}$$

where, $u(n) = \sum_{j=1}^{k} h(b^j)$

$$h(n) = \frac{f(n)}{n^{\log_b a}}$$

$$\Rightarrow u(n) = \sum_{j=1}^{k} h(b^j) = h(b^1) + h(b^2) + h(b^3) + \cdots + h(b^k)$$

$$= \frac{f(b)}{b^{\log_b a}} + \frac{f(b^2)}{b^{2\log_b a}} + \frac{f(b^3)}{b^{3\log_b a}} + \cdots + \frac{f(b^k)}{b^{k\log_b a}}$$

$h(n)$		$u(n)$
$\Theta(n^r)$	$r<0$	$O(1)$
$\Theta(\log n)^r$	$r\geq 0$	$\Theta(\log n)^{r+1}/r+1$
$\Omega(n^r)$	$r>0$	$\Theta(h(n))$

Q) What is the time complexity of the following recurrence relation:

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

(i) $a=1$, $b=2$, $f(n)=C$

(ii) $a=1$, $b=2$, $f(n)=cn$

Scanned with OKEN Scanner

$\log_b a = \log_2 1 = 0$

Sol: (i) $h(n) = \frac{f(n)}{n^0} = \frac{c}{1} = c(\log n)^0$

$= \theta(\log n)^0, \ t = 0$

$u(n) = \theta(\log n)^{0+1}/0+1$

$= \theta(\log n)$

$T(n) = [T(1) + \theta(\log n)] = \theta(\log n)$

(ii) $\log_2 1 = 0$

$h(n) = \frac{f(n)}{n^0} = f(n)$

$h(n) = cn = O(n) = \Omega(n)$

$u(n) = \theta(cn) = \theta(n)$

$T(n) = n^0 [T(1) + u(n)]$

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

$= \theta(n)$

(iii) $a = 2, \ b = 2, \ f(n) = n$

$\log_b a = \log_2 2 = 1$

$h(n) = \frac{f(n)}{n^1} = \frac{f(n)}{n} = \frac{n}{n} = 1 = \theta(\log n)^t, \ t = 0$

$u(n) = \theta(\log n)^{0+1}/0+1 = \theta(\log n)$

$T(n) = n[T(1) + \theta(\log n)] = n\theta(\log n) = \theta(n \log n)$

Scanned with OKEN Scanner

(iv) $a=4, b=2, f(n)=n^2$

$\log_b a = \log_2 4 = 2$

$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^2} = 1 = \theta(\log n)^l, \; l=0$

$u(n) = \theta(\log n)^{a+1}/0+1 = \theta(\log n)$

$T(n) = n^{\log_b a}[T(1) + \theta(\log n)] = n^2[T(1)+\theta(\log n)] = \theta(n^2 \log n)$

03/08/2024 Saturday

Q) What is the time complexity of the following recurrence relation:

$a = 28, b=3, f(n) = cn^3$

Sol: $\log_b a = \log_3 28 = 3.03$

$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{cn^3}{n^{\log_3 28}} = \frac{cn^3}{n^{3.03}} = cn^{-0.03} = O(n^{-0.03})$

$\Rightarrow u(n) = O(1)$

$T(n) = n^{\log_b a}[T(1) + O(1)] = n^{3.03}[T(1)+O(1)]$ ~~$= n^{3.03}T(1) + n^{3.03}O(1)$~~

$= k.n^{3.03}$

$\therefore T(n) = O(n^{\lceil 3 \rceil})$ → nearest integer greater than '3'

Q) Solve the following recurrence relation using substitution method:

$$T(n) = \begin{cases} 1, & n \le 4 \\ T(\sqrt{n}) + c, & n > 4 \end{cases}$$

Sol: $T(n) = T(\sqrt{n}) + c$

$= \left(T\left(\sqrt{\sqrt{n}}\right) + c\right) + c$

$= T\left(n^{1/4}\right) + c + c$

Scanned with OKEN Scanner

$T(n) = [T(n^{1/2^3}) + c] + c + c$

$= T(n^{1/2^3}) + 3c$

For 'k' iterations,

$= T(n^{1/2^k}) + k.c$

Terminating condition:

$n^{1/2^k} = 4 \Rightarrow \frac{1}{2^k} = \log_n 4 = \frac{1}{\log_4 n} \Rightarrow 2^k = \log_4 n$

$n^{2^{-k}} = 4$

$2^{-k} = \log_n 4$

$n^{1/2^k} \leq 4$

$\log_2 n^{1/2^k} \leq \log_2 4$

$\frac{1}{2^k} \log_2 n \leq 2$

$\log_2 n \leq 2^{k+1}$, consider $\log_2 n = 2^{k+1}$

$k+1 = \log_2 \log_2 n$

$k = \log_2 \log_2 n - 1$

$T(n) = c(\log_2(\log_2 n) - 1) + T(n)$

$= c.\log_2(\log_2 n) - c + 1$

$= c \log(\log_2 n)$

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

Scanned with OKEN Scanner

07/08/2024

* Binary Search Algorithm:

- In the algorithm, recursive version takes the inputs a, f, l, x.

```
Binarysearch (a, f, l, x)
{
    // 'a' contains an array of element in sorted order
    // 'f' is the starting ~~elem~~ index
    // 'l' is the index of the last element
    // 'x' is the element to be searched
    if (l == f)  // Problem is small
    {
        if (x == a[f])
            return f;
        else
            return 0;
    }
    else  // Problem is big
    {
        mid = [(f+l)/2]
        if (x == a[mid])
            return mid;
        else if (x < a[mid])
            Binarysearch (a, f, mid-1, x);
        else if (x > a[mid])
            Binarysearch (a, mid+1, l, x);
        else
            return 0;
    }
}
```

Scanned with OKEN Scanner

Eg: Consider the following set of values:

-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151.

x = 151,

x = 151	f	l	mid	
	1	14	7	a[7] = 54 ⇒ 151 > 54
	8	14	11	a[11] = 125 ⇒ 151 > 125
	12	14	13	a[13] = 142 ⇒ 151 > 142
	14	14	14	a[14] = 151 ⇒ 151 = 151

Therefore, return 14.

08/08/2024 Thursday

* Binary search Algorithm using iteration:

Iterative version:

```
Binarysearch (a,n,x)
{
  low = 1, high = n;
  while (low <= high)
  {
     mid = ⌈(low+high)/2⌉
     if x = a[mid] then return mid;
     else if x < a[mid] then high = mid-1;
     else if x > a[mid] then low = mid+1;
     else return 0;
  }
}
```

Scanned with OKEN Scanner

Eg: -15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 123, 134, 142, 151

x = 36

low := 1, high := 14

mid := 7

low := 1, high := 6

mid := 3

a[3] = 0

x = 36

low := 4, high : 6

mid : 5, a[5] = 9, x = 36

low = 6, high = 6

mid = 6, a[3] = 23, x = 36

low = 7, high := 6

* Time Complexity:

Let, the input elements or input keys be 'n', your search may terminate after generating '2^k' no. of mid values, whose value cannot exceed 'n'. Therefore, maximal value of $2^k = n \Rightarrow k = \log_2 n$

When binary search is successful, the time complexity of the algorithm can be taken as $O(\log n)$

When the search is unsuccessful, you can consider $O(\log n)$ as the time complexity.

09/08/2024 Friday

* Breadth First Search (BFS):

- In graphs, starting from a source node visiting or exploring nodes in the next level after completion of visiting all the nodes in the current level is the general strategy of Breadth first search algorithm.

Scanned with OKEN Scanner

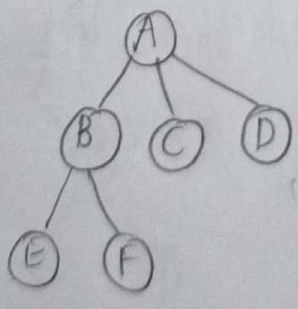
That means, to visit/explore nodes in level 'N', all the nodes in level 'N-1' need to be visited mandatory.

```
BFS (V)
{
   //'q' is a queue containing nodes to be visited, once visited they
   //can be deleted from the queue 'q'
   //'v' is a node under processing/exploration
   u := v
   visited[v] := 1
   repeat
   {
      for all vertices 'w' adjacent to 'u' do
      {
         if (visited[w] = 0) then
         {
            add 'w' to 'q'
            visited[w] = 1
         }
      }
      if 'q' is empty then return
      delete next element 'u' from 'q'
   }until (false);
}
```

10/08/2024 Saturday

* Representation of a graph using adjacency list and matrix:

Eg:



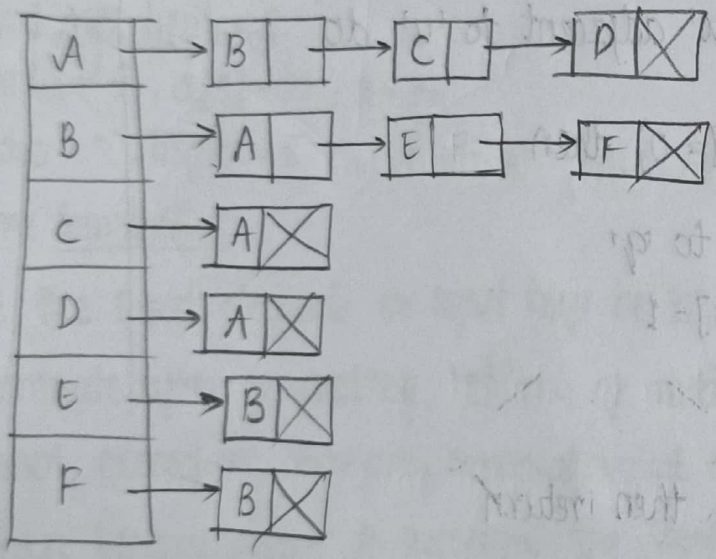
Scanned with OKEN Scanner

Adjacency matrix:

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	0
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	0	0	0	0

6×6

Adjacency list:



* Depth first search (DFS):

Algorithm:

```
DFS(v)
{
    visited[v] = 1
    repeat
    {
        for all vertices 'w' adjacent 'v'
        if(visited[w]=0)
```

```
    {
        DFS(w)
    }
}until(false);
}
```

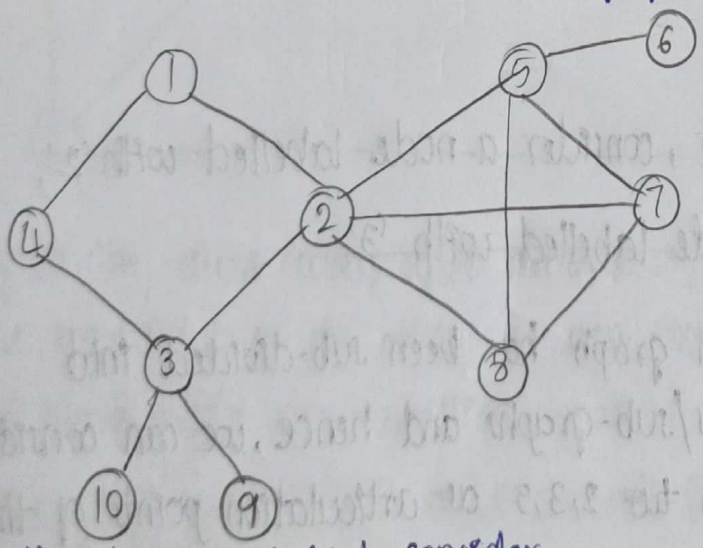
14/08/2024 Wednesday

* Bi-Connected Components / Graphs:

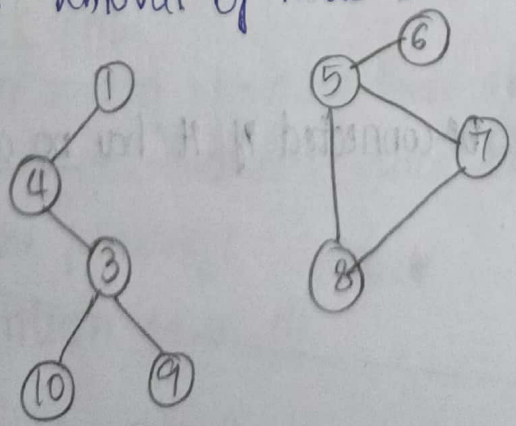
* Articulation points:

A node in ~~graph~~ a graph 'G' is said to be an articulation point if removal of ~~that~~ node is dividing the graph 'G' into two or more non-empty sub-components / graphs.

For example, consider the following graph:

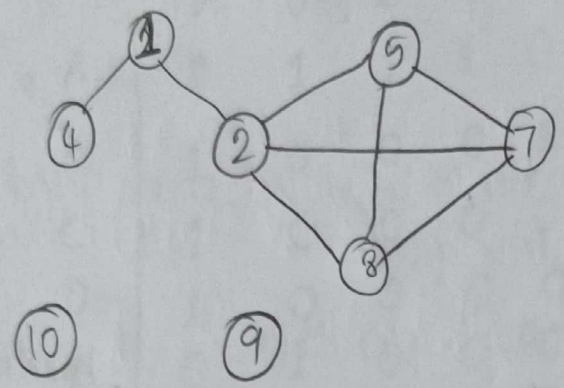


In the above graph 'G1', consider,
After removal of node '2'

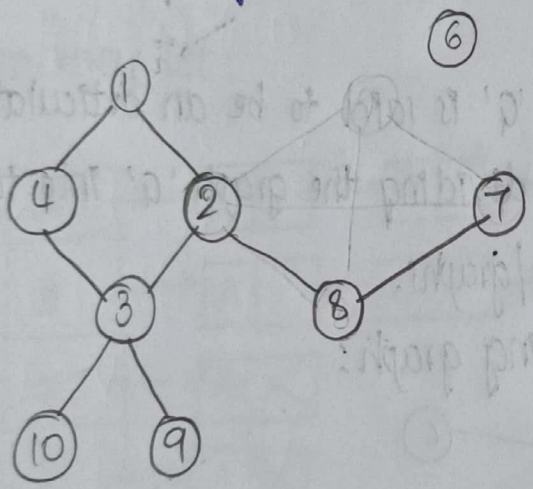


These two sub-graphs are non-empty.

After removal of node '3'



After removal of node '5'



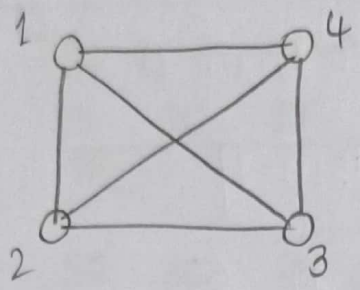
~~In the above graph 'G', consider a node labelled with '2', after removal of the node labelled with '3'~~

In all above cases, given graph has been sub-divided into non-empty sub-components/sub-graphs and hence, we can consider the nodes labelled with ~~two~~ 2,3,5 as articulation points of the graph.

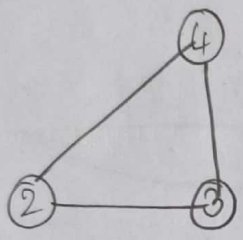
* Bi-connected Graph:

A graph 'G' is said to be bi-connected if it has no articulation points.

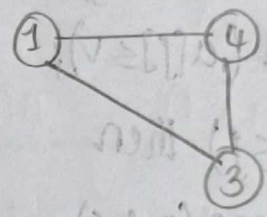
Eg: Consider the following graph 'G2',



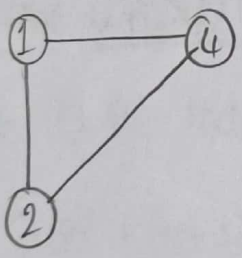
After removal of '1',



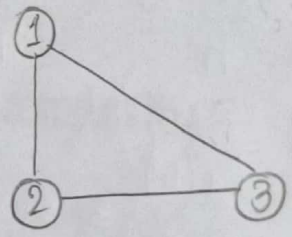
After removal of '2',



After removal of '3',



After removal of '4',



In all the above cases, after removal of nodes 1,2,3,4, we got one empty sub-graph and one non-empty subgraph. So, none of the above nodes are auxiliary points.

∴ The given graph 'G2' is a bi-connected graph.

21/08/2024 Wednesday

* Quick Sort:

- In this sorting algorithm, there are '2' major components, one is partition and another is quick sort. The module partition works like the following:

```
Partition (a,m,p)
{
    v= a[m], i=m;
```

```
j=p;
repeat
{
    repeat
      i=i+1
    until (a[i]≥v);
    repeat
      j=j-1
    until (a[j]≤v);
    if (i<j) then
       swap(a,i,j)
}until(i≥j)
a[m]=a[j]
a[j]=v
return j;
}
```

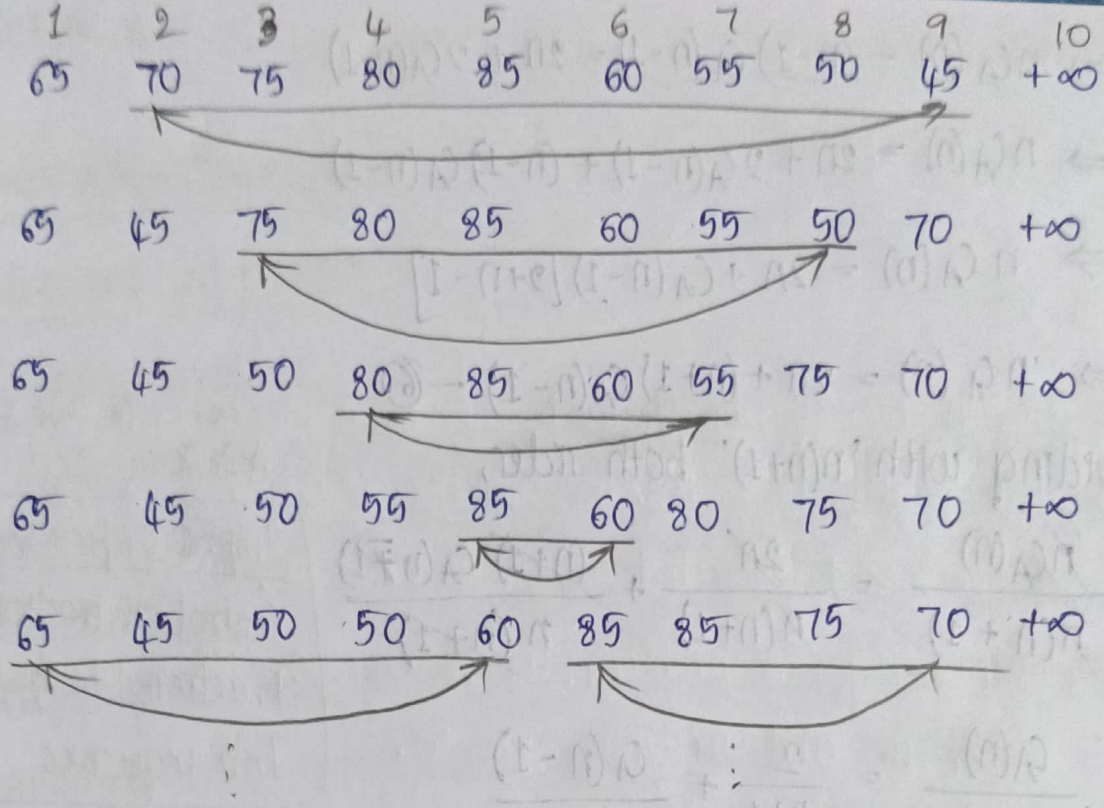
This algorithm returns the index where pivot element fit enough to divide given array into sub-arrays where the subarrays contain smaller elements than pivot and bigger elements than pivot.

Consider the following example, an array of '9' elements and apply partition algorithm on this:

array = [65, 70, 75, 80, 85, 60, 55, 50, 45]

Put last element as '+∞'.

Scanned with OKEN Scanner



Saturday

24/08/2024

* Time Complexity of Quick sort:

Let $C_A(n)$ indicating avg-case time complexity:

$$C_A(n) = (n+1) + \frac{1}{n}\sum_{1\le k\le n}[C_A(k-1) + C_A(n-k)] \quad -①$$

$$C_A(0) = C_A(1) = 0$$

$$nC_A(n) = n(n+1) + \sum_{1\le k\le n}[C_A(k-1) + C_A(n-k)] \quad -②$$

If we write $(n-1)^{th}$ iteration

$$(n-1)C_A(n-1) = (n-1)n + \sum_{1\le k\le n}[C_A(k-1) + C_A(n-k-1)] \quad -③$$

$$nC_A(n) = n(n+1) + 2\{C_A(0) + C_A(1) + C_A(2) + \dots + C_A(n-2) + C_A(n-1)\} \quad -④$$

$$(n-1)C_A(n-1) = (n-1)(n) + 2\{C_A(0) + C_A(1) + C_A(2) + \dots + C_A(n-1) + C_A(n-2)\} \quad -⑤$$

Scanned with OKEN Scanner

④ - ⑤ $\Rightarrow n C_A(n) - (n-1) C_A(n-1) = 2n + 2C_A(n-1)$

$\Rightarrow n C_A(n) = 2n + 2C_A(n-1) + (n-1) C_A(n-1)$

$\Rightarrow n C_A(n) = 2n + C_A(n-1)[2+n-1]$

$\Rightarrow n C_A(n) = 2n + (n+1) C_A(n-1)$ — ⑥

Dividing with 'n(n+1)' both sides,

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

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

$$\frac{C_A(n-1)}{n} = \frac{2}{n} + \frac{C_A(n-2)}{(n-1)} \quad \text{— ⑧}$$

☆☆☆

Imp. Ques from Unit-III
Control Abstraction
Fractional knapsack
Job sequence

28/08/2024 Wednesday

$$\frac{C_A(n)}{n+1} \leq 2 \log n$$

$$C_A(n) \leq 2(n+1) \log n$$

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

Scanned with OKEN Scanner

25/08/2024 Wednesday

UNIT - III

~~* Greedy Paradm:~~

* Control Abstraction:

* Greedy Paradm:

In this method, some constraints will be given to solve given problem along with an objective function which is to be minimised or maximised. Any solution of the problem satisfying the constraints and whose objective function is being minimized or maximized is called a feasible solution. A problem can have multiple feasible solutions, but, among them only one can be optimal solution.

Therefore, we can write the general abstraction / control abstraction of greedy method as follows:

```
Greedy (a, n)
{
    solution = ϕ
    x = select (a)
    if feasible (solution, x)
    {
        solution = Union (solution, x)
    }
    return solution
}
```

29/08/2024 Thursday

* Fractional Knapsack:

Eg: $w_i \Rightarrow 18, 15, 10$

$P_i \Rightarrow 25, 24, 15$

$m = 20$

- There are 'n' no. of objects in the problem along with associated weights ('w_i') and profits ('P_i') $(1 \le i \le n)$. The objective maximize $\sum P_i x_i$, where x_i $0 \le x_i \le 1$ subjected to $\sum w_i x_i \le m$.

Feasible solutions for a problem: let us exercise with $n = 3$ and profits 25, 24, 15 and weights 18, 15, 10.

The feasible solutions of the above problem can be as follows:

Fractions	Profit
$(1, \frac{2}{15}, 0)$	~~$18 + \frac{2}{15} \times 15$~~ $25 \times 1 + \frac{2}{15} \times 24 = 28.2$
$(1, 0, \frac{2}{10})$	$25 + 0 + \frac{1}{5} \times 15 = 28$
$(\frac{5}{18}, 1, 0)$	$\frac{5}{18} \times 25 + 24 + 0 = 30.94$
$(0, 1, \frac{1}{2})$	$0 + 24 + \frac{1}{2} \times 15 = 31.5$
$(0, \frac{10}{15}, 1\phi)$	$0 + \frac{10}{15} \times 24 + 15 = 31$
$(\frac{10}{18}, 0, 1)$	$\frac{10}{18} \times 25 + 0 + 15 = 28.89$

Scanned with OKEN Scanner

So far, ~~the optimal~~ maximum profit is 31.5 and the corresponding solution is $(0, 1, \frac{1}{2})$.

At this point, this solution is looking to be optimal but no algorithm specified to conclude such optimal value. Therefore, the following greedy algorithms or versions can be adopted to decide optimal value of the profit.

~~Case~~ (i): Select objects in the non-increasing (decreasing) order of profits.

(ii) Select objects in the ~~weight in the~~ non-decreasing order of w (weights)

(iii) Select objects in non-increasing order of $\frac{p}{w}$

So in our above example:

(i) $\left(1, \frac{2}{15}, 0\right) \Rightarrow$ Profit $= 25 + \frac{2}{15} \times 24 + 0 = 28.2$

(ii) $\left(0, \frac{2}{3}, 1\right) \Rightarrow$ Profit $= 0 + \frac{2}{3} \times 24 + 15 = 31$

(iii) $\frac{P_1}{w_1} = \frac{25/18}{25} = 1.38$ (3)

$\frac{P_2}{w_2} = \frac{24/15}{24} = 1.6$ (1)

$\frac{P_3}{w_3} = \frac{15}{10} = 1.5$ (2)

So, $\left(0, 1, \frac{1}{2}\right) \Rightarrow$ Profit $= 0 + 24 + \frac{1}{2} \times 15 = 31.5$

Hence by selecting the objects in the non-~~decr~~ increasing order of $\frac{p}{w}$, the optimal solution is 31.5.

Scanned with OKEN Scanner

Eg: $m = 15$

$P_i \Rightarrow 10, 5, 15, 7, 6, 18, 3$

$w_i \Rightarrow 2, 3, 5, 7, 1, 4, 1$

$\frac{P_1}{w_1} = \frac{10}{2} = 5$ ① — 2

$\frac{P_2}{w_2} = \frac{5}{3} = 1.6$ ⑥ — $\frac{2}{3}$

$\frac{P_3}{w_3} = \frac{15}{5} = 3$ ④ — 5

$\frac{P_4}{w_4} = \frac{7}{7} = 1$ ⑦

$\frac{P_5}{w_5} = \frac{6}{1} = 6$ ① — 1

$\frac{P_6}{w_6} = \frac{18}{4} = 4.5$ ③ — ~~4~~

$\frac{P_7}{w_7} = \frac{3}{1} = 3$ ⑤ — ~~1~~

So, $\left(1, \frac{2}{3}, 1, 0, 1, 1, 1\right)$

1 2 3 4 5 6 7

$\Rightarrow$ Profit $= 10 + \frac{2}{3} \times 5 + 15 + 0 + 6 + 18 + 3 = 55.33$

30/08/2024 Friday

* Algorithm:

```
GreedyKnapsack(
{
    for i = 1 to n do
        x[i] = 0
    U = m
```

```
for i:= 1 to n do
{
    if (w[i] > 0) then break
    x[i] := 1;
    U = U - w[i];
}
if (i ≤ n) then x[i] = U / w(i)
}
```

* Job Sequencing with Deadlines:

- In these problems, 'n' no. of jobs along with deadlines will be given and every job should be completed on or before the deadline. Each job is associated with a profit.
- Our problem is finding feasible solutions in which all jobs completed in deadline and cummulative profit will be maximal.

Every job will complete its execution in one unit of time.

- For example, consider n = 4 jobs and their associated profits and deadlines are:

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

S.No.	Sequence	Profit
[1, 2]	J2, J1	100+10 = 110
[2,3]	J_2, J_3	10+15 = 25
[1,3]	J_1, J_3 (or) J_3, J_1	100+15 = 115
[1,4]	J_4, J_1	27+100 = 127
[3,4]	J_4, J_3	27+15 = 42

Scanned with OKEN Scanner

[1]	J_1	100
[2]	J_2	10
[3]	J_3	15
[4]	J_4	27

In the above, sequence [4,1] found to be with optimal profit.
Therefore, optimal solution for this job sequencing problem is [4,1].