14/01/25

Page No.
Date: / /

* **Prerequisites:**
  Programming
  Data Structures

* **Algorithm**

It is a step by step procedure to solve a computation problem.

- **Characteristics:**

Input : Every algorithm may have certain inputs.

Output : Every algorithm must generate certain output.

Definiteness : Every inst^n within an algorithm must be unambiguous.

Finiteness : Every inst^n should execute a finite no. of times.

Effectiveness : Each inst^n must be effectively solve a part of the problem.

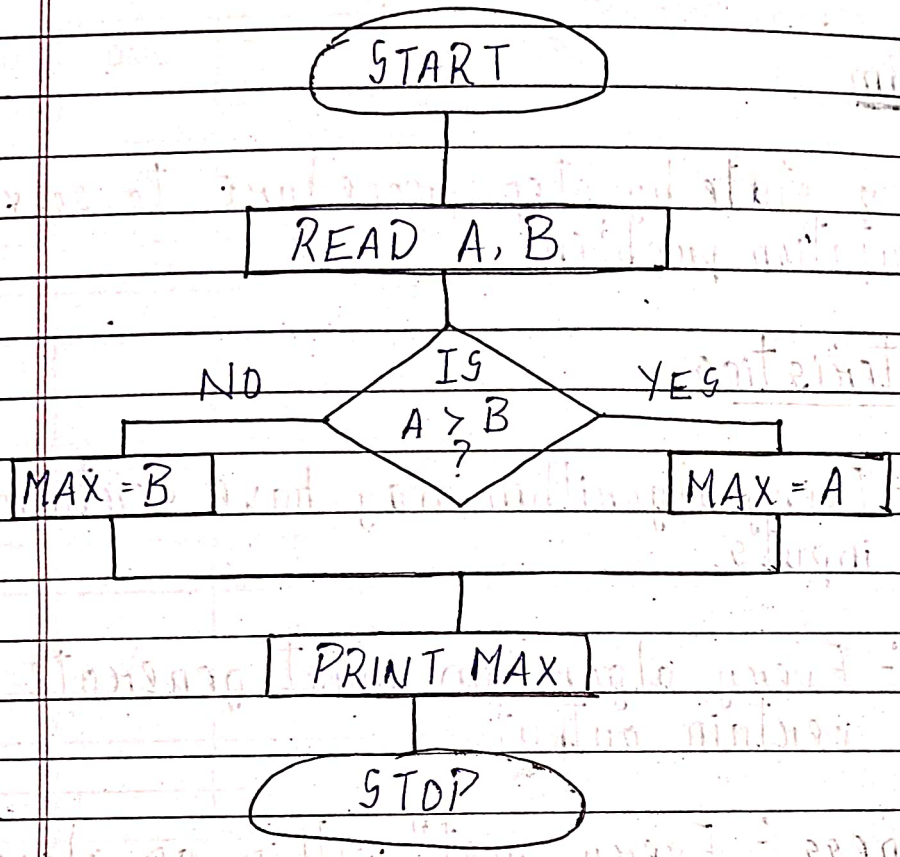
- **Tools to Represent an Algorithm**

1) Flowchart
2) Pseudo code
3) High level Language (C / C++ / JAVA...)

Page No.
Date: / /

To find the max^m bet^n 2 no.s

Flowchart



Pseudocode

```
Algo max (a, b)

begin
    if (a > b)
        max = a
    else
        max = b
    display (max)
end
```

Page No.
Date: / /

Techniques / Tools to Write an Algorithm

1) Iterative Technique
2) Divide & Conquer Technique
3) Greedy Technique
4) Dynamic Programming
5) Branch & Bound
6) Backtracking
7) Graph algorithm
8) Sorting algorithm
9) Soft Computing Technique
10) Machine learning & Deep learning.

17/01/25

How to Measure An Algorithm.

1) Time Complexity : The time required to execute an algorithm. Asymptotic Notations are used to measure it.

2) Space Complexity : The space required to execute an algorithm.
Units : Asymptotic Notations

Time & Space Trade Off

$$\text{Time} \propto \frac{1}{\text{Space}}$$

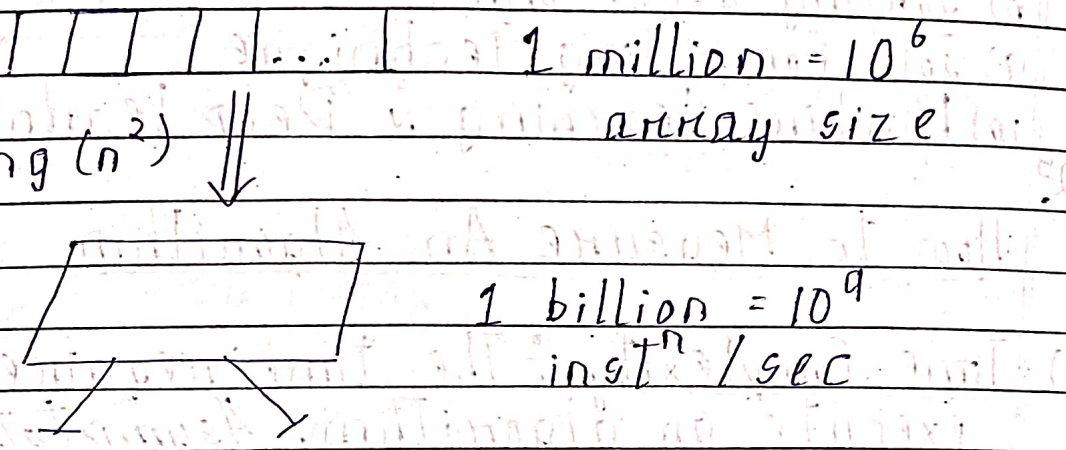
If you want to execute algorithm faster, then you may have to provide more space to the algorithm & vice-versa.

Page No.
Date: / /

Now a days, we are concentrating only on the time complexity of the algorithm as the space is cheap.

Problem: An array contains 1 million data elements. A machine is running 1 billion inst^n/sec. Using a sorting algorithm whose running time is n^2. The machine is sorting the array. How much time required for its execution.

A:



n = 1 million = 10^6

$n^2 = (10^6)^2 = 10^{12}$

Machine is executing

10^9 inst^n $\longrightarrow$ 1 sec

1 inst^n $\longrightarrow$ $\frac{1}{10^9}$ sec

10^{12} inst^n $\longrightarrow$ $\frac{1}{10^9} \times 10^{12} = 10^3$ sec

$= \frac{1000}{60} = 16.66$ min

Page No.
Date: / /

If the sorting algorithm is merge sort having running time $n \log_2 n$.

A- $n \log_2 n = 10^6 \log_2 10^6$

10^9 inst^n $\longrightarrow$ 1 sec

1 inst^n $\longrightarrow$ $\frac{1}{10^9}$

$10^6 \log_2 10^6$ inst^n $\longrightarrow$ $\frac{1}{10^9} \times 10^6 \log_2 10^6$

$= \frac{\log_2 10^6}{10^3} = \frac{20}{1000}$

$= 0.02$ sec

$\log_2 10^6 = \log_2 (1000 \times 1000)$
$= \log_2 (2^{10} \times 2^{10})$
$= \log_2 (2^{20})$
$= 20 \log_2 2$
$= 20$

20/01/25 **Bubble Sort & its Running Time Analysis**

Bubble Sort $\longrightarrow$ Simplest sorting algorithm

Worst Case

0	1	2	3	4	5
12	10	8	6	4	2

Iteration 1

Step 1 : 10 12 8 6 4 2

Step 2 : 10 8 12 6 4 2

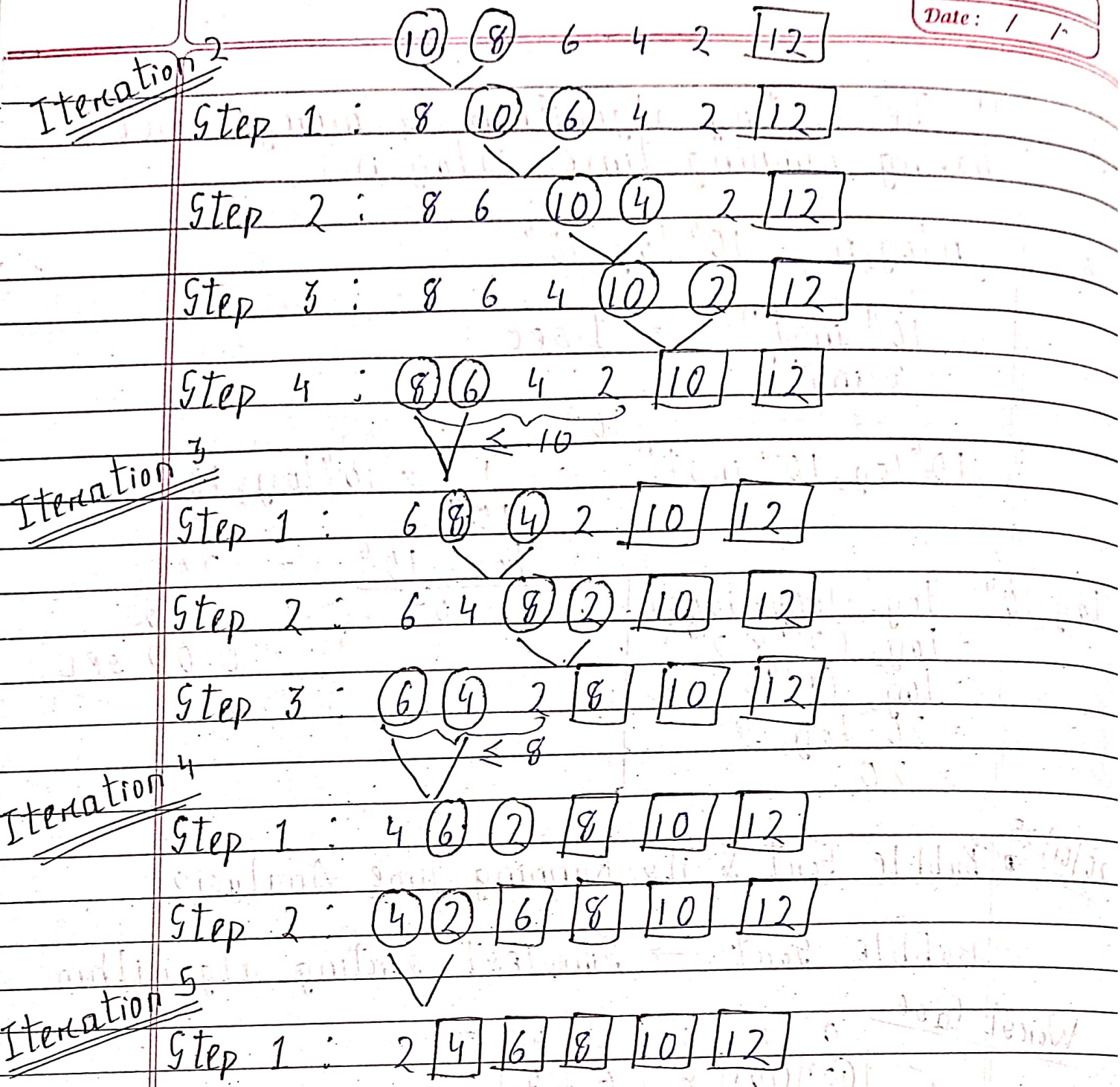
Step 3 : 10 8 6 12 4 2

Step 4 : 10 8 6 4 12 2

Step 5 : 10 8 6 4 2 | 12 |

≤ 12

Page No.
Date: / /



∴ Sorted elements : 2 4 6 8 10 12

Elements = 6
Iteration = 5

1st iteration ⟶ 5
2nd " ⟶ 4
3rd " ⟶ 3
4Th " ⟶ 2
5Th " ⟶ 1

Page No.
Date: / /

NOTE If the array has n no. of elements
No. of iterations = n-1

1st iteration ⟶ (n-1)
2nd " ⟶ (n-2)
3rd " ⟶ (n-3)
⋮
(n-1)th iteration ⟶ 1 step

∴ The total no. of steps :
$= (n-1) + (n-2) + (n-3) + \dots + 1$
$= \frac{(n-1)(n-x+x)}{2}$
$= \frac{n(n-1)}{2}$ $(= \frac{n^2}{2} - \frac{n}{2} \simeq n^2$ (quadratic)

Best Case

2	4	6	8	10	12

Iteration 1

Step 1 : 2 (4) (6) 8 10 12 swapped = false
Step 2 : 2 4 (6) (8) 10 12 swapped = false
Step 3 : 2 4 6 (8) (10) 12 swapped = false
Step 4 : 2 4 6 8 (10) (12) swapped = false
Step 5 : 2 4 6 8 10 12 swapped = false

Best case : n
Worst case : n^2

Page No.
Date : / /

NOTE We do not have to go for 2nd iteration as the flag swapped is false in each step

∴ The bubble sort running time varies from n to n^2.

```
Algo bubblesort (A, n)
//A is the array, n is the size of the array
{
    boolean swapped
    for (i = 0 ; i < (n - 1) ; i++)
    {
        swapped = false
        for (j = 0 ; j < (n - i - 1) ; j++)
        {
            if (A[j] > A[j+1])
            {
                swapped = true
            }
        }
        if (swapped == false)
        break
    }
}
```

28/01/25

Page No.
Date: / /

Bubble Sort C Program

```
void BubbleSort (int A[10], int n)
{
 int i, j, swapped, temp;
 for (i=0; i<n-1; i++)
 {
   swapped = 0;
   for (j=0; j<n-i-1; j++)
   {
     if (A[j] > A[j+1])
     {
       temp = A[j];
       A[j] = A[j+1];
       A[j+1] = temp;
       swapped = 1;
     }
     if (swapped == 0)
       break;
   }
 }
}

#include <stdio.h>
int main()
{
  int B[10], n, i;
void Bubblesort (int [10], int) //Func^n prototype
  //Read the no. of elements in the array
    printf ("Enter the no. of elements: ");
    scanf ("%d", &n);
  //Read the array element
```

```
printf("Enter the %d no. of array elements : ",n);
for(i=0; i<n; i++)
    scanf("%d", &B[i]);
//Before sorting print the array
printf("The unsorted array is : \n");
for(i=0; i<n; i++)
    printf("%d\n", B[i]);
BubbleSort(B, n);
//After sorting the array
printf("The sorted array is : \n");
for(i=0; i<n; i++)
    printf("%d\n", B[i]);
}
```

- **Linear Search & its Running Time Analysis**

0	1	2	3	4	5	6
5	2	7	3	9	8	16

element = 17

Best Case

1, $O(1)$ or constant → Running Time

If the element is present at the 1st location.

Worst Case

$O(n)$ or linear

If the element is present at the last location or not present in the array

* If the element is not present, then $(n+1)$ no. of searches is required.

size = n

If the element is present at the last index, n no. of searches.

Page No.
Date: / /

Avg. case OR Expected Running Time

$$E(x) = x_1P_1 + x_2P_2 + x_3P_3 + \dots + x_nP_n$$

$x_i \longrightarrow$ The pos^n of the element

$$= 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \dots n \cdot \frac{1}{n}$$

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

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

$P_i \longrightarrow$ probability that the element is present at the ith pos^n.

24/01/25

Algo linearSearch (A, n, element)
// A is the array, n is the size of the array and element is the item you are searching in

```
{
  for (i=0 ; i<n ; i++)
  {
    if (A[i] == element)
      return i
  }
  printf ("The element is not found")
}
```

Page No.
Date: / /

PROBLEM: How many times loop will be executed?

i) `for (i = 0; i <= 6; i++);`

A- 7 Loop will be run one more time

ii) `for (i = 1; i <= n; i++);`

A- n (n-1+1)

iii) `for (i = 0; i <= n-1; i++)`

A- n

iv)
```
for (i = 1; i < n; i++)
    x = x + 1
```

A- n times
(n-1) times

v)
```
for (i = 2; i < n-3; i++)
    x = x+5;    (in loop)
    y = y+5;    (outside loop)
```

A- (n-4) Times
(n-5) Times
1 time

vi) `for (i = 1; i <= 1024; i = i*2)`

A- 12 times

for (i = 1 ; i <= 5 ; i++) //5
for (j = 1 ; j <= 4 ; j++) //25
x = x + 1 //20

Page No.
Date: / /

vii) for (i = 1 ; i <= n ; i = i * n)

A- $\log_2 n$

viii) for (i = 1 ; i <= n ; i = i * 3)

A- $\log_3 n$

Eg. for (i = 1 ; i <= 243 ; i = i * 3)

7 times

ix) for (i = 1 ; i <= n ; i = i * 2)
for (j = 1 ; j <= n ; j = j * 3)
x = x + 1

A $\log_2 n$
$\log_2 n \times \log_3 n$
$\log_2 n (\log_3 n - 1)$

27/01/25

Binary Search & its Running Time Analysis

EXAMPLE- Searching a word in a dictionary

Advantage Running time is $\log_2 n$ in worst case

Disadvantage The array must be in sorted order.

Page No.
Date: / /

0	1	2	3	4	5	6	7
2	4	6	10	15	18	25	35

Item = 25 ?

Search 25 in the array. If found return its index

Step 1 beg = 0
end = 7
mid = $\frac{beg + end}{2}$ = 3 → 10

If (A[3] == 25) ? NO
Since (25 > A[3])
beg = mid + 1
= 3 + 1 = 4

STEP 2 beg = 4
end = 7
mid = $\frac{beg + end}{2}$ = 5 → 18

If (A[5] == 25) ? NO
Since (25 > A[5])
beg = mid + 1
= 5 + 1 = 6

STEP 3 beg = 6
end = 7
mid = $\frac{beg + end}{2}$ = 6.

If (A[6] == 25) ? YES

∴ The searched item 25 is present at index 6.

Page No.
Date: / /

0	1	2	3	4	5	6	7
2	4	8	12	15	17	19	25

Item = 18 ?

STEP 1 beg = 0
end = 7
mid = $\frac{beg + end}{2}$ = 3 → 12

If (A[3] == 18) ? NO
Since (18 > A[3])
beg = mid + 1
= 3 + 1 = 4

STEP 2 beg = 4
end = 7
mid = $\frac{beg + end}{2}$ = 5 → 17

If (A[5] == 18) ? NO
Since (18 > A[5])
beg = mid + 1
= 5 + 1 = 6

STEP 3 beg = 6
end = 7
mid = $\frac{beg + end}{2}$ = 6 → 19

If (A[6] == 18) ? NO
Since (18 < A[6])
end = mid - 1
= 6 - 1 = 5

STEP 4 beg = 6
end = 5

∴ Since (beg > end)
The searched item 18 is not present in the array.

Algo Binary Search (A, n, item)
// A is the sorted array, n is the size of the array, item is the element's index searched for.

```
{
  beg = 0, end = n-1
  while (beg <= end)
  {
    mid = (beg + end) / 2
    if (A[mid] == item)
        return (mid)
    else if (item > A[mid])
        beg = mid + 1
    else
        end = mid - 1
  }
  display (item not found)
}
```

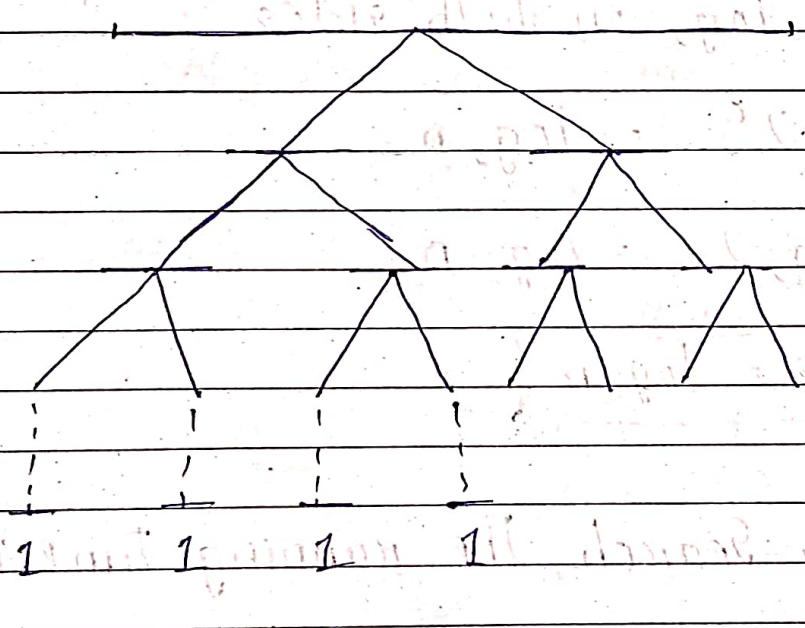
Binary Search Program

```
int BinarySearch (int B[10], int n, int item)
{
  int beg, mid, end;
```

```
beg = 0;
end = n-1;
while (beg <= end)
{
    mid = (beg + end)/2;
    if (A[mid] == item)
        return (mid);
    else if (item > A[mid])
        beg = mid + 1;
    else
        end = mid - 1;
}
return (-1);
}
```

28/01/25

Running Time



Let the pos^n size = n

Page No.
Date: / /

After 1 division (requires only 1 unit of time)
The problem size = $\frac{n}{2}$

After 2 division (requires 2 unit of time)
The problem size = $\frac{n}{4} = \frac{n}{2^2}$

.
.
.

let after ith division the searched element is found (in the worst case)

ith division (requires i unit of time)
problem size = $\frac{n}{2^i}$

$\therefore \frac{n}{2^i} = 1$

$\Rightarrow 2^i = n$

Taking $\log_2$ on both sides

$\log_2 2^i = \log_2 n$

$\Rightarrow i \log_2 2 = \log_2 n$

$\Rightarrow \boxed{i = \log_2 n}$

For Binary Search, the running time for

Best case = 1 unit
Worst case = $\log_2 n$

Page No.
Date: / /

Asymptotic Notations

Asymptotic Notations are used to measure the time as well as space complexity of algorithms.

These notations gives the information how a func^n behaves for the larger ~~group~~ value of the input to the algorithm.

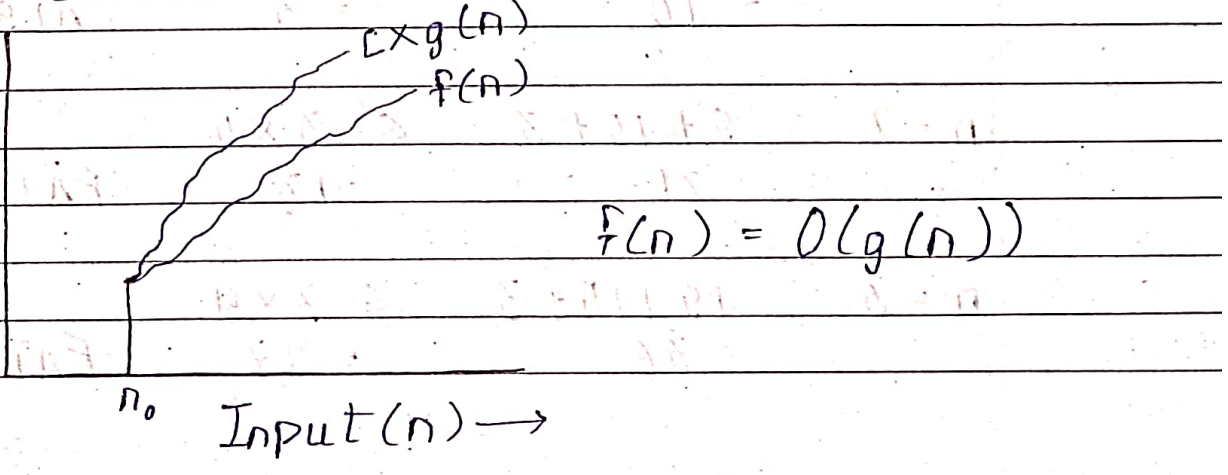
Different Asymptotic Notations are :-

1) Big-Oh Notation (O)
2) Big Omega Notation (Ω)
3) Theta Notation (Θ)
4) Little Oh Notation (o)
5) Little omega Notation (ω)

1) Big Oh Notation (O) [atmost]

Def^n: If $f(n)$ & $g(n)$ are 2 func^{n}s $f(n) = O(g(n))$ ($f(n)$ is said to be big Oh of $g(n)$) if $\exists$ a const c such that

$$f(n) \leq c * g(n) \quad \forall \quad n \geq n_0$$



Page No.
Date: / /

Class	Student 1	Student 2
I	85	83
II	90	87
III	92	93
IV	90	94
V	87	95
VI	93	94
VII	91	96
VIII	87	89
IX	90	91
X	91	92

std $\geq$ 3 $\Rightarrow$ $n_0 = 3$

Student 1 = O (Student 2) $\forall$ std $\geq$ 3
since student 1 has scored less marks than student 2 from std 3.

Problem: Represent $f(n) = 2n^2 + 5n + 3$ in terms of O Notation.

A

$$2n^2 + 5n + 3 \leq 3n^2$$

$n = 1$ $\quad 2 + 5 + 3 \leq 3 \times 1$
$= 10 \quad = 3$ FALSE

$n = 2$ $\quad 8 + 10 + 3 \leq 3 \times 4$
$= 21 \quad = 12$ FALSE

$n = 3$ $\quad 18 + 15 + 3 \leq 3 \times 9$
$= 36 \quad = 27$ FALSE

Page No.
Date: / /

$n = 4$ $\quad 32 + 20 + 3 \leq 3 \times 16$
$= 55 \qquad = 48$ FALSE

$n = 5$ $\quad 50 + 25 + 3 \leq 3 \times 25$
$= 78 \qquad = 75$ FALSE

$n = 6$ $\quad 72 + 30 + 3 \leq 3 \times 36$
$= 105 \qquad = 108$ TRUE

$$2n^2 + 5n + 3 = O(n^2) \quad \forall \; n \geq 6 \;\&\; c = 3$$

NOTE Big Oh (O) gives the asymptotic upper bound. That upper bound may or may not be asymptotically tight.

For eg. if $f(n) = n^2 \Rightarrow f(n) = O(n^2)$
$= O(n^3)$
$= O(n^4)$
$\vdots$
$= O(n^{100})$

All are correct. But $f(n) = O(n^2)$ is more informative.

If $f(n) = n^2$
$\Rightarrow f(n) = O(n)$ WRONG

* If $f(n) = O(n^\alpha)$
$\Rightarrow f(n) = O(n^{\alpha+\epsilon})$
But if $f(n) = O(n^\alpha)$
$\not\Rightarrow f(n) = O(n^{\alpha-\epsilon})$ WRONG

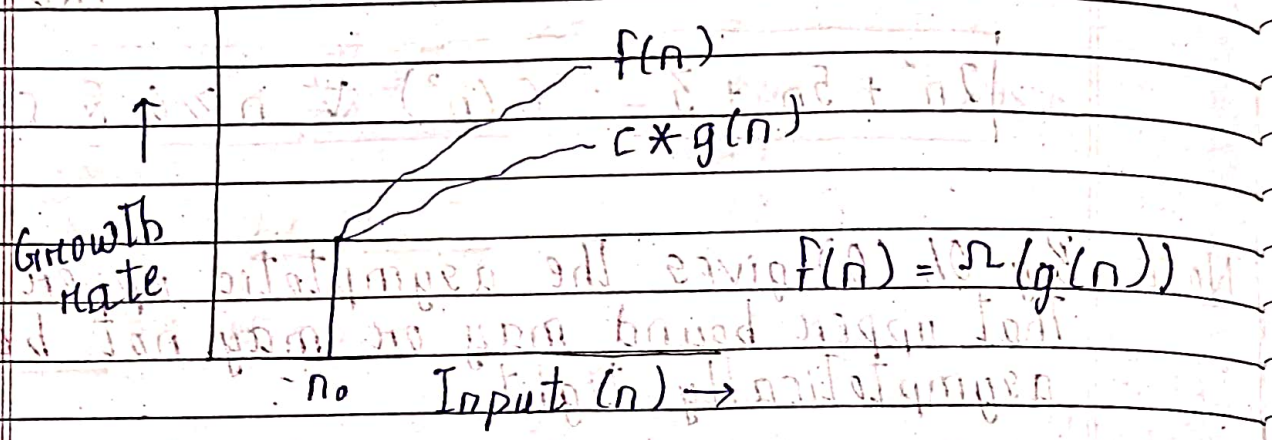
31/01/25

Page No.
Date: / /

2) Big Omega Notation (Ω) [at least]

Def": If $f(n)$ & $g(n)$ are two func"s
$f(n) = \Omega(g(n))$
if $\exists$ a const. c such that

$$f(n) \geq c * g(n) \quad \forall \quad n \geq n_0$$



NOTE Big Omega (Ω) gives the asymptotic lower bound. That lower bound may or may not be asymptotically tight.

Problem Write the Big Omega notation for the following code:

```
k = 5
for (i = 1 ; i <= n ; i++)
    for (j = 1 ; j <= n ; j++)        } n² times
        k = k + 1

for (m = 1 ; m <= n ; m++)
    for (t = 1 ; t <= n ; t++)        } n² times
        k = k + 2
```

```
for(i = 1 ; i <= n ; i++)
    t = t + k
```
} n times

```
t = t + 1
t = t + 2
t = t + 1
```
} 3 times

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

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

$n = 1 \quad 2 + 1 + 3 \geq 2$

$6 \geq 2$ TRUE

$\boxed{2n^2 + n + 3 \geq 2n^2 \quad \forall \quad n \geq 1 \ \& \ c = 2}$

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

$n = 1 \quad 2 + 1 + 3 \geq 1$

$6 \geq 1$ TRUE

$\boxed{2n^2 + n + 3 \geq \Omega(n^2) \quad \forall \quad n \geq 1 \ \& \ c = 1}$

NOTE For the func^n

$f(n) = n^2$, $f(n) = \Omega(n^2)$

$= \Omega(n^{1.9})$

$= \Omega(n^{1.5})$

$\vdots$

$= \Omega(n^{0.1})$

All are correct, but $f(n) = \Omega(n^2)$ is more informative.

* Let $f(n) = \Omega(n^{\alpha})$
$\Rightarrow f(n) = \Omega(n^{\alpha-\epsilon})$

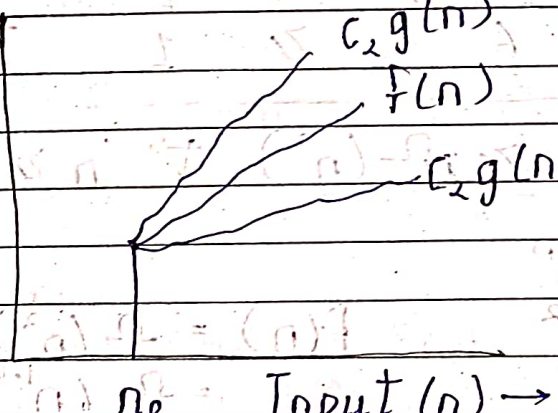
[e.g. if $f(n) = \Omega(n^3)$, Then we may say,
$f(n) = \Omega(n^{2.9})$
$f(n) = \Omega(n^2)$
$f(n) = \Omega(n)$
} All are correct

* Let $f(n) = \Omega(n^{\alpha-\epsilon})$
$\not\Rightarrow f(n) = \Omega(n^{\alpha})$ WRONG

3) Theta (Θ) Notation

Def^n: If $f(n)$ and $g(n)$ two func^{n}s.
$f(n) = \Theta(g(n))$ [$f(n)$ is theta of $g(n)$]
if $\exists$ two const.s c_1 & c_2 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall\ n \geq n_0$$



Problem: Represent the func^n $f(n) = 5n^2 + 7n + 3$ in terms of Θ notation.

Page No.
Date: / /

$5n^2 \leq 5n^2 + 7n + 3 \leq 6n^2$

$n=1$ $5 \leq 15 \leq 6$ FALSE

$n=2$ $5 \times 4 \leq 20 + 14 + 3 \leq 6 \times 4$
$20 \leq 37 \leq 24$ FALSE

$n=3$ $5 \times 9 \leq 45 + 21 + 3 \leq 6 \times 9$
$45 \leq 68 \leq 54$ FALSE

$n=4$ $5 \times 16 \leq 80 + 28 + 3 \leq 6 \times 16$
$80 \leq 111 \leq 96$ FALSE

$n=5$ $5 \times 25 \leq 125 + 35 + 3 \leq 6 \times 25$
$125 \leq 163 \leq 150$ FALSE

$n=6$ $5 \times 36 \leq 180 + 42 + 3 \leq 6 \times 36$
$180 \leq 225 \leq 216$ FALSE

$n=7$ $5 \times 49 \leq 245 + 49 + 3 \leq 6 \times 49$
$245 \leq 297 \leq 294$ FALSE

$n=8$ $5 \times 64 \leq 320 + 56 + 3 \leq 6 \times 64$
$320 \leq 379 \leq 384$ TRUE

$\boxed{5n^2 + 7n + 3 = O(n^2) \quad \forall \; n \geq 8, \; c_1 = 5, \; c_2 = 6}$

NOTE Let $\lim_{n \to \infty} \frac{f(n)}{g(n)} = L$ where $L > 0$

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

03/01/25

$\frac{N^n}{D^n} = 0 \Rightarrow D^n >> N^n$ $\quad \frac{N}{D^n} = +ve\ no.$

$\frac{N^n}{D^n} = \infty \Rightarrow N^n >> D^n$ $\quad \Rightarrow N^n \simeq D^n$

Page No.
Date: / /

Problem: Prove That $3n^2 + 5n = \Theta(n^2)$

A-

$$\lim_{n\to\infty} \frac{3n^2+5n}{n^2}$$

$$= \lim_{n\to\infty} \frac{3n^2}{n^2} + \lim_{n\to\infty} \frac{5n}{n^2}$$

$$= \lim_{n\to\infty} 3 + \lim_{n\to\infty} \frac{5}{n}$$

$$= 3 + 0 = 3$$

Since $3 > 0$

$\therefore f(n) = \Theta(n^2)$

Problem: Prove or Disprove $3n^2 + 5n = \Theta(n^3)$

A-

$$\lim_{n\to\infty} \frac{3n^2+5n}{n^3}$$

$$= \lim_{n\to\infty} \frac{n^2(3n+5)}{n^3}$$

$$= \lim_{n\to\infty} \frac{3n}{n^2} + \lim_{n\to\infty} \frac{5}{n^2}$$

$$= 0 + 0$$

$$= 0$$

$\therefore 3n^2 + 5n^2 \neq \Theta(n^3)$

Limits

Q- Find $3x+2$ where $x = 2$

A- $3 \times 2 + 2 = 8$

Page No.
Date: / /

Q- Find $\lim_{x \to 2} 3x+2$ $\quad 3\times 2+2 = 8$

Q- Find $\frac{x^2-4}{x-2}$ where $x=2$ $\quad \frac{0}{0} = UD$

Q- Find $\lim_{x \to 2} \frac{x^2-4}{x-2}$

$$\lim_{x \to 2} \frac{(x-2)(x+2)}{(x-2)}$$

$$= \lim_{x \to 2} x+2 = 2+2 = 4$$

LHL $\lim_{x \to 2^-} \frac{x^2-4}{x-2}$

$$= \lim_{h \to 0} \frac{(2-h)^2-4}{2-h-2}$$

$$= \lim_{h \to 0} \frac{4-h^2-4h-4}{-h}$$

$$= \lim_{h \to 0} \frac{-h(h^2+4)}{-h} = 4$$

RHL $\lim_{x \to 2^+} \frac{x^2-4}{x-2}$

$$= \lim_{h \to 0} \frac{(2+h)^2-4}{2+h-2}$$

$$= \lim_{h \to 0} \frac{4+4h+h^2-4}{h}$$

$$= \lim_{h \to 0} \frac{h(h+4)}{h} = 4$$

Page No.
Date: / /

Problem: Let $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ $(a_k > 0)$. Prove that $f(n) = \Theta(n^k)$.

A-

$$\lim_{n\to\infty} \frac{a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0}{n^k}$$

$$= \lim_{n\to\infty} a^k + \lim_{n\to\infty} \frac{a_{k-1}}{n} + \lim_{n\to\infty} \frac{a_{k-2}}{n^2} + \dots + \lim_{n\to\infty} \frac{a}{n^{k-1}} + \lim_{n\to\infty} \frac{a_0}{n^k}$$

$$= \lim_{n\to\infty} a^k + 0 + 0$$

$$= a^k > 0$$

$\therefore f(n) = \Theta(n^k)$

Problem: Prove that $(n+a)^b = \Theta(n^b)$, where $b > 0$, a & b are const.s.

A-

$$\lim_{n\to\infty} \frac{(n+a)^b}{n^b}$$

$$= \lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^b = \lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^b$$

$$= \lim_{n\to\infty} 1 + \lim_{n\to\infty} \frac{1}{n^b} = \left(\lim_{n\to\infty} 1 + \lim_{n\to\infty} \frac{a}{n}\right)^b$$

$$= (1+0)^b$$

$$= 1 > 0$$

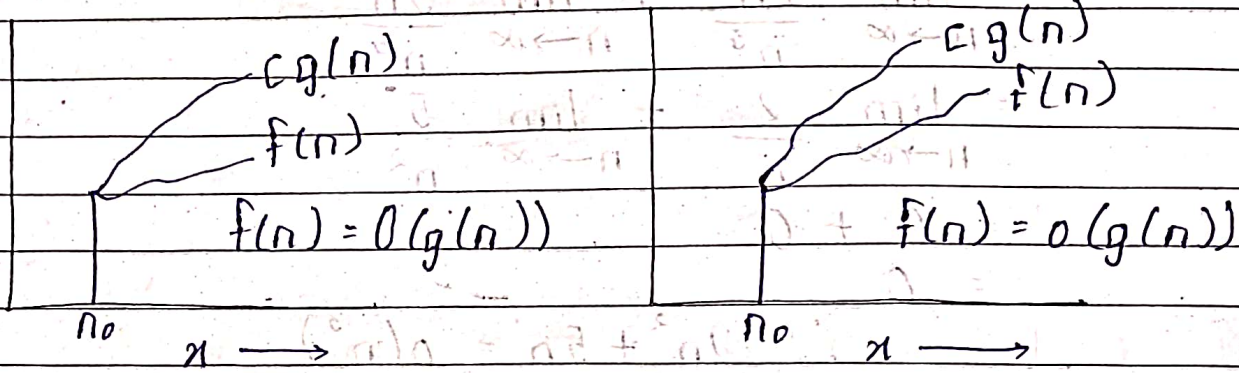
$\therefore f(n) = \Theta(n^b)$

04/02/25

Page No.
Date: / /

4) Little Oh (o) Notation

Big Oh (O) gives the asymptotic upper bound. That upper bound may or may not be asymptotically tight.
Little Oh (o) notation gives the asymptotic upper bound. That upper bound never be asymptotically tight.



Def^n : If f(n) & g(n) are the functions, f(n) = o(g(n)) if ∀ a const. c such that

$$f(n) < c * g(n) \quad \forall \quad n \geq n_0$$

For the func^n
$f(n) = n^2$

$f(n) = O(n^2)$ ← asymptotically tight
$= O(n^{2.1})$
$= O(n^3)$
$\vdots$
$= O(n^\infty)$
} asymptotically upper bound

But
for the func^n $f(n) = n^2$
$f(n) = o(n^2)$ is incorrect.
All others are like $o(n^{2.1})$, $o(n^3)$, $o(n^\infty)$ are correct.

NOTE Let $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Then $f(n) = o(g(n))$

Problem: Prove that $2n^2 + 5n = o(n^3)$

A- $\lim_{n\to\infty} \frac{2n^2+5n}{n^3}$

$= \lim_{n\to\infty} \frac{2n^2}{n^3} + \lim_{n\to\infty} \frac{5n}{n^3}$

$= \lim_{n\to\infty} \frac{2}{n} + \lim_{n\to\infty} \frac{5}{n^2}$

$= 0 + 0$

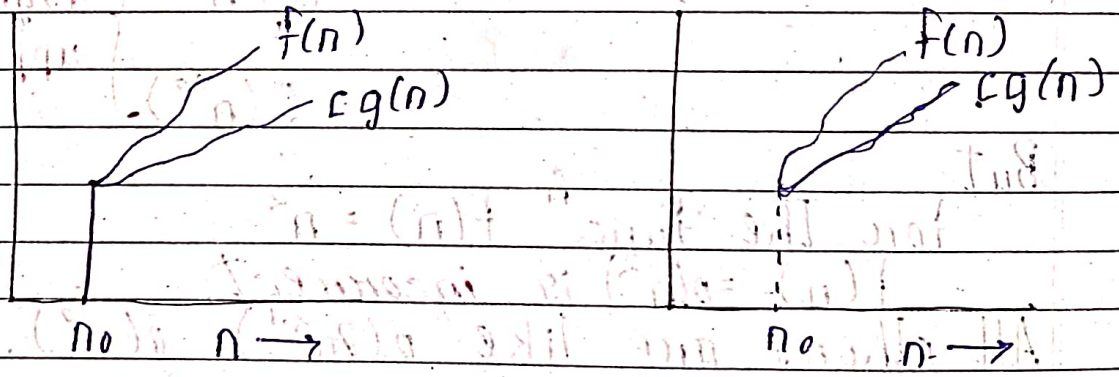
$= 0$

$\therefore 2n^2 + 5n = o(n^3)$

5) **Little Omega (ω) Notation**

Big Omega (Ω) gives the asymptotic lower bound. That lower bound may or may not be asymptotically tight.

Little omega (ω) gives the asymptotically lower bound. That lower bound never be asymptotically tight.



Page No.
Date: / /

Defⁿ: If $f(n)$ & $g(n)$ are 2 funcⁿs $f(n) = \omega(g(n))$ if ∄ a const. c such that

$$\boxed{f(n) > c * g(n) \ \forall \ n \geq n_0}$$

NOTE Let $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Then $f(n) = \omega(g(n))$

Theorem

Let $\lim_{n \to \infty} \frac{f(n)}{g(n)} = L$

i) if $L > 0$, Then $f(n) = \Theta(g(n))$
ii) if $L = 0$, Then $f(n) = o(g(n))$
iii) if $L = \infty$, Then $f(n) = \omega(g(n))$

- **Stirling's Approximation**

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

$$\simeq \left(\frac{n}{e}\right)^n$$

Problem: Prove that $\log_2 n! = \Theta(n \log_2 n)$.

LHS $\log_2 n! = \log_2 \left(\frac{n}{e}\right)^n$

$= n \log_2 \frac{n}{e}$

$= n (\log_2 n - \log_2 e)$

Page No.
Date: / /

Now;

$$\lim_{n \to \infty} \frac{n(\log_2 n - \log_2 e)}{n \log_2 n}$$

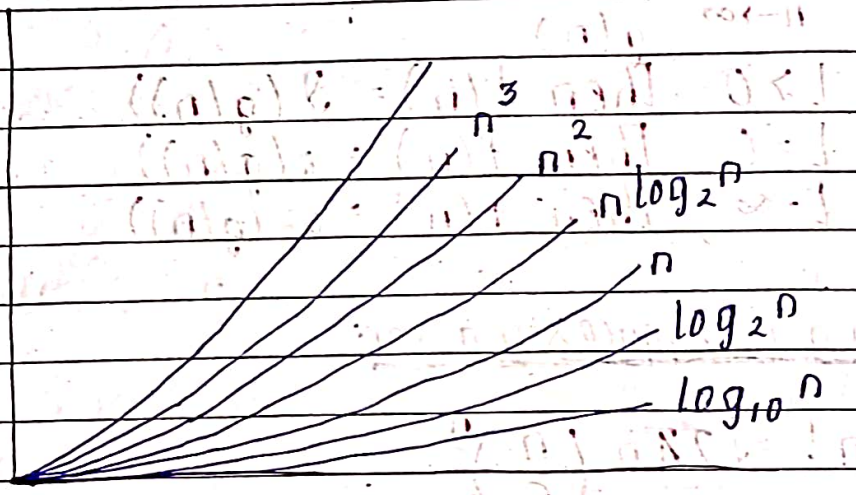
$$= \lim_{n \to \infty} \frac{\log_2 n}{\log_2 n} - \lim_{n \to \infty} \frac{\log_2 e}{\log_2 n}$$

$$= 1 - 0$$

$$= 1 > 0$$

$$\therefore f(n) = \Theta(n \log_2 n)$$

Growth Rate of Func^{n}s



7/02/25

Page No.
Date: / /

Problem: Prove that $\sum_{i=1}^{n} \log_2 i = O(n \log_2 n)$

A- We proved that

$\log_2 n! = O(n \log_2 n)$ ——— (1)

LHS $\sum_{i=1}^{n} \log_2 i$

$= \log_2 1 + \log_2 2 + \ldots + \log_2 n$

$= \log_2 (1 \cdot 2 \cdot 3 \cdots n)$

$= \log_2 n!$

$= O(n \log_2 n)$ (∵ From (1))

(Proved)

Problem: Prove that $n! = o(n^n)$.

A- Method 1

$\lim_{n \to \infty} \frac{n!}{n^n} = \lim_{n \to \infty} \frac{(n/e)^n}{n^n}$ (∵ By stirling approximatio

$= \lim_{n \to \infty} \frac{n^n}{e^n} \times \frac{1}{n^n}$

$= \lim_{n \to \infty} \frac{1}{e^n} = 0$

∴ $n! = o(n^n)$

Method 2

$\lim_{n \to \infty} \frac{n!}{n^n} = \lim_{n \to \infty} \frac{n(n-1)(n-2)\ldots 1}{n^n}$

$= \lim_{n \to \infty} \frac{n(n(1-1/n))(n(1-2/n))\ldots n \cdot 1/n}{n^n}$

$= \lim_{n \to \infty} \frac{n^n (1-1/n)(1-2/n)\ldots 1/n}{n^n}$

$= \lim_{n \to \infty} (1 - \frac{1}{n}) \cdot \lim_{n \to \infty} (1 - \frac{2}{n}) \cdots \lim_{n \to \infty} \frac{1}{n}$

$\Theta \Leftrightarrow O \;\&\&\; \Omega$

Page No.
Date: / /

$= (1-0)(1-0) \cdots 0$

$= 0$

$\therefore n! = o(n^n)$

Problem: Is $2^{n+1} = O(2^n)$?

A- $\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2}{2^n} = 2$

$\therefore 2^{n+1} = \Theta(2^n)$

$\Leftrightarrow 2^{n+1} = O(2^n) \;\&\&\; \Omega(2^n)$

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

Problem: Arrange the following func^{n}s in the ascending order of their growth rates.

$\log_{10} n,\ \log_2 n,\ \log_3 n,\ n,\ (\log_2 n)^2$

A- $\log_{10} n \leq \log_3 n \leq \log_2 n < (\log_2 n)^2 < n$

Comparison bet^n n & $(\log_2 n)^2$

If $\lim_{n\to\infty} \frac{n}{(\log_2 n)^2} = 0$, Then $n < (\log_2 n)^2$

If $\lim_{n\to\infty} \frac{n}{(\log_2 n)^2} = \infty$ Then $n > (\log_2 n)^2$

$\lim_{n\to\infty} \frac{n}{(\log_2 n)^2}$ $\left(\frac{\infty}{\infty} \text{ form}\right)$

$\frac{d}{dx}(x^2) = 2x$

$\frac{d}{dx}(\log_e x) = \frac{1}{x}$

$\frac{d}{dx}(\log_a x) = \frac{1}{x}\log_a e$

Page No.

Date: / /

Applying L' Hospital Rule

$$\lim_{n\to\infty} \frac{d/dn(n)}{d/dn(\log_2 n)^2} = 1$$

$$\frac{d}{dn}(\log_2 n)^2 = 2\log_2 n \frac{d}{dn}(\log_2 n)$$

$$= 2\log_2 n \cdot \frac{1}{n}\log_2 e$$

Now; $\lim_{n\to\infty} \frac{1}{2\log_2 n \frac{1}{n}\log_2 e}$

$= \frac{1}{2\log_2 e} \lim_{n\to\infty} \frac{n}{\log_2 n}$ (Applying L'Hospital Rule)

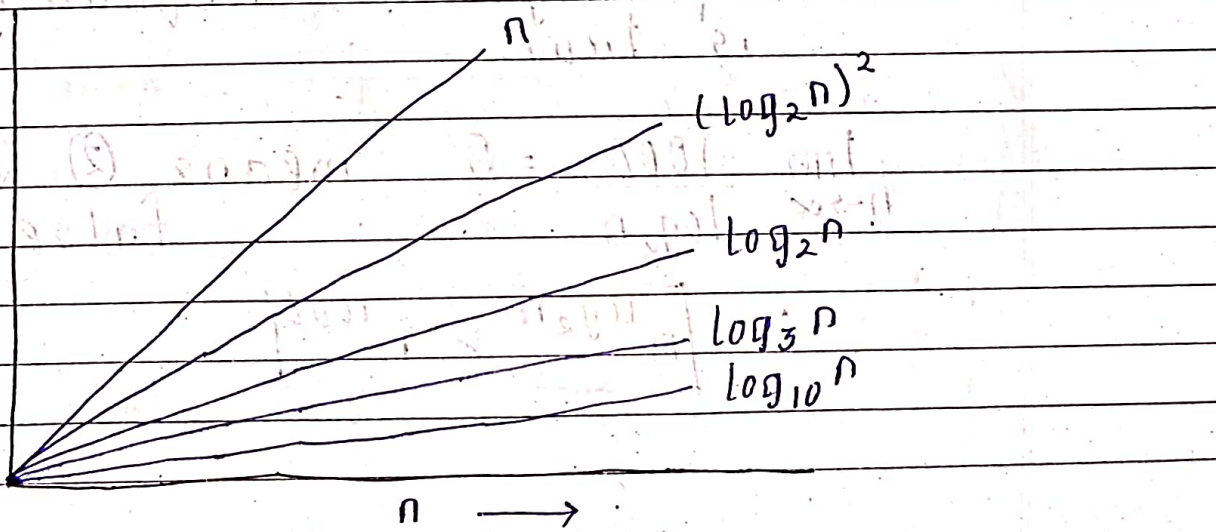
$= \frac{1}{2\log_2 e} \lim_{n\to\infty} \frac{1}{1/n \log_2 e}$

$= \frac{1}{2(\log_2 e)^2} \lim_{n\to\infty} n = \infty$

$\therefore n > (\log_2 n)^2$

The arrangement will be -

$\log_{10} n$, $\log_3 n$, $\log_2 n$, $(\log_2 n)^2$, n.



10/02/25

Page No.
Date: / /

Problem: Arrange the following func^{n}s in their ascending order of growth rates

$n!, 2^n, n^{\log_2 n}, n^{1000}, \log_2 n, 10, \log_{10} n$

(exponential; logarithm)

A- $10, \log_{10} n, \log_2 n$

$2^n < n!$

Comparison bet^n $n^{\log_2 n}$ & n^{1000}

Let $n^{\log_2 n} < n^{1000}$ —— (1)

Taking $\log_2$ on both sides

$\Rightarrow \log_2 n \log_2 n < 1000\, n \log_2 n$ ($\because \log m^n = n \log m$)

$\Rightarrow \log_2 n < 1000$ —— (2)

If $\lim_{n \to \infty} \frac{1000}{\log_2 n} = 0$,

inequality (2) & inequality (1) is false.

If $\lim_{n \to \infty} \frac{1000}{\log_2 n} = \infty$,

inequality (2) & inequality (1) is true.

$\lim_{n \to \infty} \frac{1000}{\log_2 n} = 0$, means (2) & (1) is false.

$\therefore \boxed{n^{\log_2 n} > n^{1000}}$

Page No.
Date: / /

Comparison bet^n $n^{\log_2 n}$ & 2^n

Let $n^{\log_2 n} < 2^n$ ——— (3)

Taking $\log_2$ on both sides

$\log_2 n^{\log_2 n} < \log_2 2^n$

$\Rightarrow \log_2 n \times \log_2 n < n \log_2 2$

$\Rightarrow (\log_2 n)^2 < n$ $(\because \log_m m = 1)$ ——— (4)

If $\lim_{n \to \infty} \frac{(\log_2 n)^2}{n} = 0$

inequality (4) & inequality (3) is false

If $\lim_{n \to \infty} \frac{(\log_2 n)^2}{n} = \infty$

inequality (4) & inequality (3) is true.

$\lim_{n \to \infty} \frac{(\log_2 n)^2}{n}$

Applying L' Hospital Rule

$= \lim_{n \to \infty} \frac{d/dn (\log_2 n)^2}{\frac{d}{dn}(n)}$

$\frac{d}{dn}(\log_2 n)^2 = 2 \cdot \log_2 n \frac{d}{dn}(\log_2 n)$

$= 2 \log_2 n \cdot \frac{1}{n} \log_2 e$

Page No.
Date: / /

$$\lim_{n\to\infty} \frac{2\log_2 e \log_2 n}{n}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{\log_2 n}{n}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{d/dn(\log_2 n)}{d/dn(n)}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{1}{n}\log_2 e$$

$$= 2(\log_2 e)^2 \lim_{n\to\infty} \frac{1}{n}$$

$$= 0$$

$$\therefore \boxed{n^{\log_2 n} < 2^n}$$

$2^n < n!$

$n^{\log_2 n} > n^{1000}$

$n^{\log_2 n} < 2^n$

$$\therefore \boxed{10 < \log_{10} n < \log_2 n < n^{1000} < n^{\log_2 n} < 2^n < n!}$$

11 to 2 12 5

Insertion Sort & Running Time Analysis

Eg. Arranging the playing cards

Algo Insertion Sort (A, n)
for j ← 1 to (n-1)
do
key ← A[j]
i ← j-1
// Insert A[j] into the sorted array A[0...j-1]

```
while ((i > -1) && (A[i] > key))
{
    A[i+1] ← A[i]
    i = i - 1
}
A[i+1] ← key
}
```

0	1	2	3
5	3	1	6

Iteration 1

j = 1
key = A[j] = 3
i = 1 - 1 = 0
while ((0 > -1) (T) && (A[0] > key) (T))
{
    A[1] = A[i] = A[0]
    i = -1
}
while (-1 > -1) (F)
{
    No operation
}
A[0] = 3

3	5	1	6
sorted			

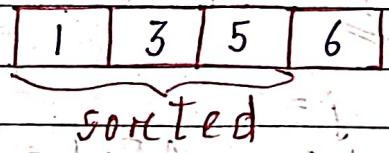
Iteration 2

j = 2
key = A[2] = 1
i = 2 - 1 = 1
while ((1 > -1) (T) && (A[1] > key) (T))
{
    A[2] ← A[i] = A[1]
    i = 1 - 1 = 0
}

while ((0 >= -1) && (A[0] > key))
(T) (T)
{
A[i] ← A[0]
i = 0 - 1 = -1
}

while (-1 >= -1)
(F)
{
No operation
}

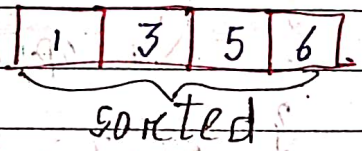
A[0] = 1



Iteration 3

j = 3
key = A[3] = 6
i = 3 - 1 = 2
while ((2 >= -1) && (A[2] > key))
(T) (F)
{
No operation
}

A[3] = 6



$\sum_{j=1}^{n} t_j = t_1 + t_2 + \dots t_{n-1}$

$j = 1$, let while loop running t_1 times

Page No.
Date: / /

	Cost	Time
for $j \leftarrow 1$ to $(n-1)$	C_1	n
{		
$key \leftarrow A[j]$	C_2	$n-1$
$i = j - 1$	C_3	$n-1$
while $((i > -1) \&\& (A[i] > key))$	C_5	$\sum_{j=1}^{n} t_j$
{		
$A[j+1] \leftarrow A[i]$	C_6	$\sum_{j=1}^{n-1} t_j - 1$
$i = i - 1$	C_7	"
}		
$A[i+1] \leftarrow key$	C_8	$n-1$

$$T_n = C_1 n + C_2(n-1) + C_3(n-1) + C_5 \sum_{j=1}^{n} t_j + C_6 \sum_{j=1}^{n-1} (t_j - 1) + C_7 \sum_{j=1}^{n-1} (t_j - 1) + C_8 (n-1)$$

18/02/25

Best case

$$\sum_{j=1}^{n-1} t_j = t_1 + t_2 + \dots + t_{n-1}$$

$t_1 \rightarrow$ how many times the while loop will run when $j = 1$.

$= 1 + 1 + \dots 1$ ($n-1$ times)

$= (n-1)$

$$\sum_{j=1}^{n-1} (t_j - 1) = (t_1 - 1) + (t_2 - 1) + \dots + (t_{n-1} - 1)$$

$$= (1-1) + (1-1) + \dots + (-1-1)$$

$$= 0 + 0 + \dots + 0$$

$$= 0$$

Page No.
Date: / /

$\therefore T_n = C_1 n + C_2(n-1) + 0 + C_5(n-1) + C_6 \times 0 + C_7 \times 0 + C_8(n-1) + C_3(n-1)$

$= C_1 n + C_2(n-1) + C_5(n-1) + C_8(n-1) + C_3(n-1)$

$= n[C_1 + C_2 + C_3 + C_5 + C_8] + [-C_2 - C_5 - C_3 - C_8]$

$= D_1 n + D_2$

$= O(n)$

$D_1 = C_1 + C_2 + C_3 + C_5 + C_8$

$D_2 = -C_2 - C_3 - C_5 - C_8$

Worst Case

$$\sum_{j=1}^{n-1} t_j = t_1 + t_2 + \ldots + t_{n-1}$$

$= 2 + 3 + \ldots + n$

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

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

$$\sum_{j=1}^{n-1} (t_j - 1) = (t_1 - 1) + (t_2 - 1) + \ldots + (t_{n-1} - 1)$$

$= (2-1) + (3-1) + \ldots + (n-1)$

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

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

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

$\therefore T_n = C_1 n + C_2(n-1) + C_3(n-1) + C_5\left(\frac{n^2}{2} - \frac{n}{2} - 1\right)$

$+ C_6\left(\frac{n^2}{2} - \frac{n}{2}\right) + C_7\left(\frac{n^2}{2} - \frac{n}{2}\right) + C_8(n-1)$

$= n^2\left(\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2}\right) + n\left(C_1 + C_2 + C_3 + \frac{C_5}{2} - \frac{C_6}{2} - \frac{C_7}{2} + C_8\right) + \left[-C_2 - C_3 - C_5 - C_8\right]$

$= D_3 n^2 + D_4 n + D_5$
$= O(n^2)$

$D_3 = C_5/2 + C_6/2 + C_7/2$
$D_4 = C_1 + C_2 + C_3 + C_5/2 - C_6/2 - C_7/2 + C_8$
$D_5 = -C_2 - C_3 - C_6 - C_8$

Divide & Conquer Technique

The problems which are solved using D & C technique have the following 3 steps:

i) **Divide** :- Divide the problem P into subproblems $P_1, P_2, \ldots P_k$ which are similar in nature as P.

ii) **Conquer** :- Solve the subproblems $P_1, P_2, \ldots P_k$ recursively.

iii) **Continue** :- Combine the solved sub problems $P_1, P_2, \ldots P_k$ to get the sol^n of the original problem P.

NOTE Every D & C problems are recursive in nature.
i.e. D & C ⇒ Recursive
Recursive ≠ D & C ✗

Page No.
Date: / /

Generalized Algorithm

```
Algo D&C (P)
// P is the problem
{
  if (small (P))
     return (solution (P))
  else
     Divide P into P1, P2, ... PK
     combine (D&C(P1), D&C(P2), ... D&C(PK))
}
```

Here small is a boolean valued func^n which returns TRUE if the problem is small enough which doesn't need to be divided further, else returns FALSE

Well known D & C Problems

1) Merge Sort
2) Binary Search
3) Quick Sort
4) Matrix multiplication using Strassen's algorithm

Analyzing D & C Problems

The running time of the divide & conquer problems can be analyzed by writing a recurrence relation of the form:

Page No.
Date: / /

$$T(n) = aT\left(\frac{n}{b}\right) + \underbrace{D(n)}_{\text{Division time}} + \underbrace{C(n)}_{\text{Comb time}}$$

$\frac{n}{b}$ → size of each sub-problem

a → no. of subproblems partitioning in the next iteration.

Binary Search :-

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

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

21/02/25

Recurrence OR Recurrence Relation

A recurrence is a relation in which the nth term of the series can be represented in the form of its previous terms.

Eg. i) $S_1 = 1, 2, 3, \ldots$	ii) $S_2 = 1, 3, 5, \ldots$
$a_n = a + (n-1)d$	$a_n = a + (n-1)d$
$= 1 + (n-1)1$	$= 1 + (n-1)2$
$= 1 + n - 1$	$= 1 + 2n - 2$
$= n$ //	$= 2n - 1$ //

In A.P. the nth term is expressed as :- $a_n = a + (n-1)d$

Page No.
Date: / /

Problem: S = 1, 3, 5, 9, ...
Write the recurrence to represent the series.

A-
$a_n = a_{n-1} + 2, \quad n \geq 2$
$a_1 = 1$

Problem: The recurrence is given as
$a_n = a_{n-1} + 3, \quad n \geq 2$
$a_1 = 2$
Find the 1st 4 terms.

A- Given $a_1 = 2$

n = 2 $a_2 = a_1 + 3$
$= 2 + 3$
$= 5$

n = 3 $a_3 = a_2 + 3$
$= 5 + 3$
$= 8$

n = 4 $a_4 = a_3 + 3$
$= 8 + 3$
$= 11$

Problem: Find the recurrence of the Fibonacci series:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

A-
$a_1 = 0$
$a_2 = 1$
$a_n = a_{n-1} + a_{n-2}, \quad n \geq 3$

Page No.
Date: / /

Problem: Find the recurrence of
$S = \{3^1, 3^2, 3^3, \ldots\}$

A-
$a_1 = 3$
$a_n = 3 \cdot a_{n-1}, \quad n \geq 2$

Problem: Write the recurrence relation to find the no. of multiplication required to get $n!$

A-
$m(n) = m(n-1) + 1$
where $m(0) = 0$

Verify for $n = 4$

$m(4) = m(3) + 1$
$= \underline{m(3-1) + 1} + 1$
$= m(2) + 2$
$= \underline{m(2-1) + 1} + 2$
$= m(1) + 3$
$= \underline{m(1-1) + 1} + 3$
$= m(0) + 4$
$= 4$ //

Recurrence

Linear (Difference eq^n)

Non-Linear (D & C problems)

Methods to solve Recurrence

1) Substitution — Forward, Backward
2) Master's Theorem
3) Recursion Tree
4) Using generating func^n

1) Substitution Method

Problem: The recurrence is given as-

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

$$T(1) = 1$$

} Merge sort

Solve it.

A-

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

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

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

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

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

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

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

Page No.
Date: / /

After k^{th} iterations

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn \quad \text{---------- (1)}$$

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

$\Rightarrow 2^k = n$

$\Rightarrow \log_2 2^k = \log_2 n$

$\Rightarrow k \log_2 2 = \log_2 n$

$\Rightarrow \boxed{k = \log_2 n}$

By putting $k = \log_2 n$ in eq^n (1)

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n * n$$

$$= n T\left(\frac{n}{n}\right) + \log_2 n * n$$

$$= n \log_2 n + n$$

$$\approx n \log_2 n$$

$2^{\log_2 n} = y$

$\Rightarrow \log 2^{\log_2 n} = \log_2 y$

$\Rightarrow \log_2 n \log_2 2 = \log_2 y$

$\Rightarrow \log_2 n = \log_2 y$

$\Rightarrow \boxed{n = y}$

24/02/25

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

Ans: n^3

Hint: $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$

2) $$T(n) = \begin{cases} T(n-1) + n^3 & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

Ans: n^4

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

Page No.
Date: / /

Problem: $T(n) = \begin{cases} T(n-1) + \frac{1}{n} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$

A-

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

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

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

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

$$\vdots$$

After kTh iteration

$$= T(n-k) + \underbrace{\frac{1}{n-k+1} + \ldots + \frac{1}{n-1} + \frac{1}{n}}_{k \text{ terms}} \quad \text{---(1)}$$

Putting

$$n - k = 1$$

$$\Rightarrow \boxed{k = n-1}$$

Putting this value in eq^n (1)

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

$$= T(1) + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-1} + \frac{1}{n}$$

$$= 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \quad (H.P)$$

$$= \log_e n$$

Page No.
Date: / /

2) Master's Theorem

If a recurrence is of the form

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

Then only we can use Master's Theorem.

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

$T(n) = 2T(n-3) + n^2$ ✗

Def^n : If $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, $a \geq 1$, $b > 1$ and $\left(\frac{n}{b}\right)$ can be interpreted as $\left\lceil \frac{n}{b} \right\rceil$ or $\left\lfloor \frac{n}{b} \right\rfloor$. Then $T(n)$ must be bounded asymptotically as follows :

case ① if $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ Then

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

case ② if $f(n) = \Theta\left(n^{\log_b a}\right)$, Then

$$T(n) = \Theta\left(n^{\log_b a} \log_2 n\right)$$

case ③ if $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ ~~Then~~ &
$af\left(\frac{n}{b}\right) \leq cf(n)$ for $c < 1$

Then $T(n) = \Theta(f(n))$.

$$T(n) = T\left\lfloor \frac{n}{2} \right\rfloor + T\left\lceil \frac{n}{2} \right\rceil + n^2$$

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

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

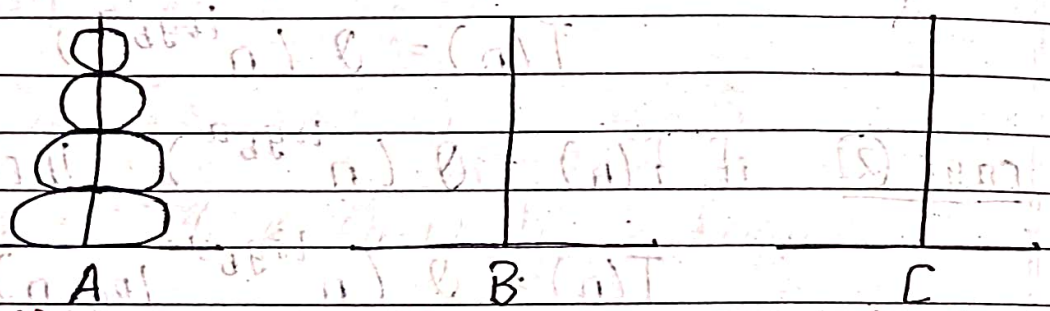
10/03/24

Tower of Hanoi Problem - An Application of Recursion

Problem Def^n :

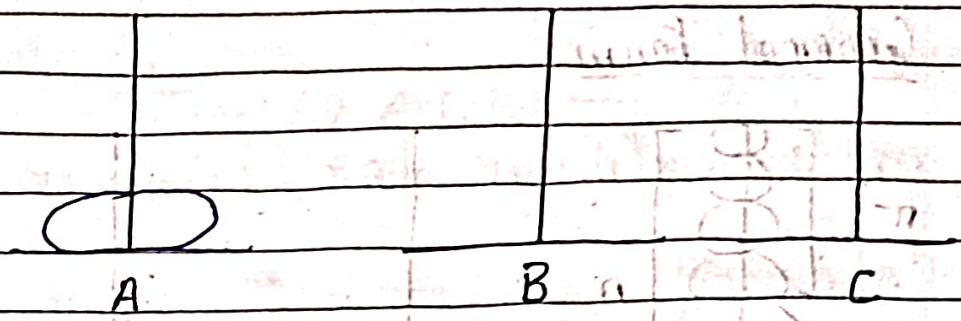
There are 3 pegs (stands) - peg A, peg B, peg C.

peg A has n no. of disks. A smaller disk is on bigger disks. Transfer all the disks from peg A using auxillary peg B such that at no time, a bigger disk is on the smaller disk.

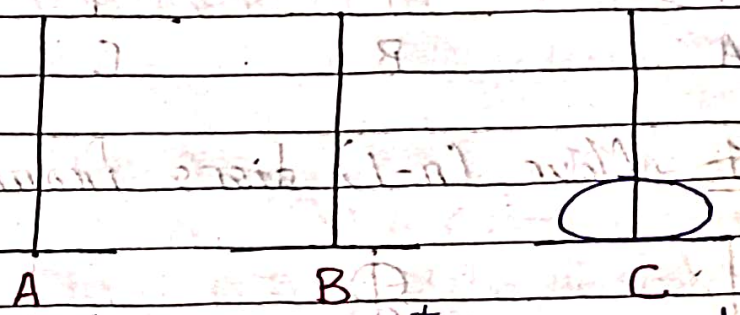


Page No.
Date: / /

n = 1

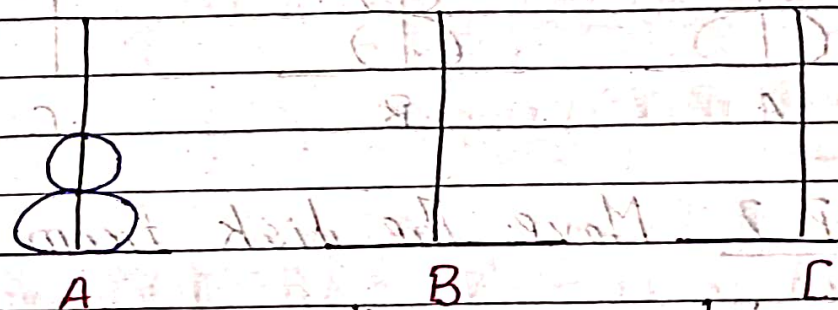


STEP 1

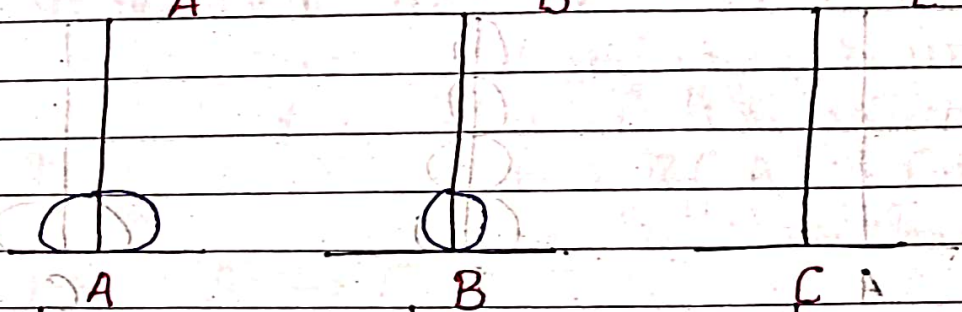


No. of disk movement required = 1

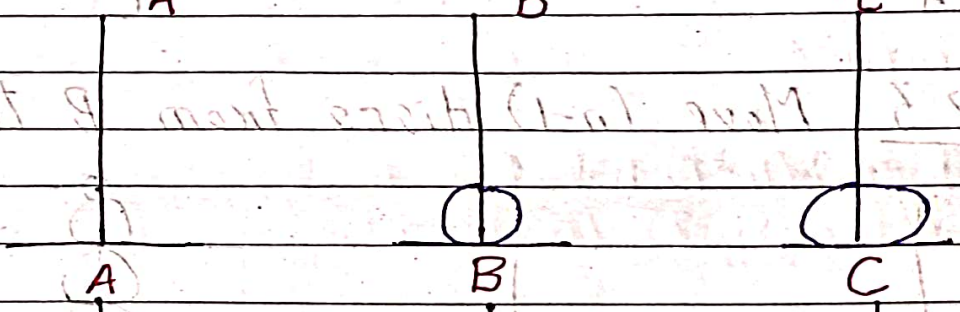
n = 2



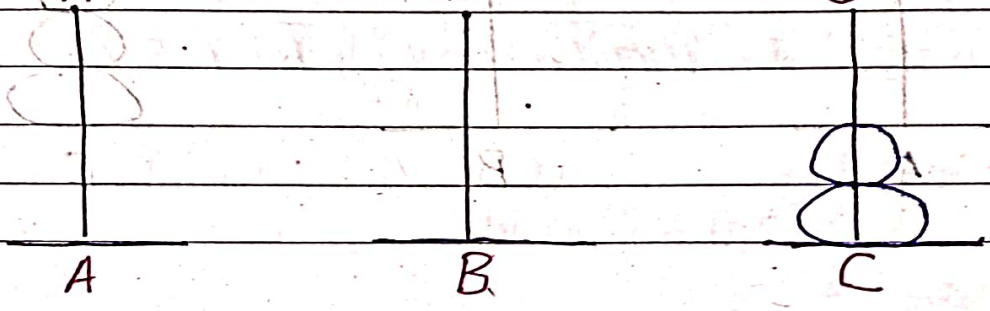
STEP 1



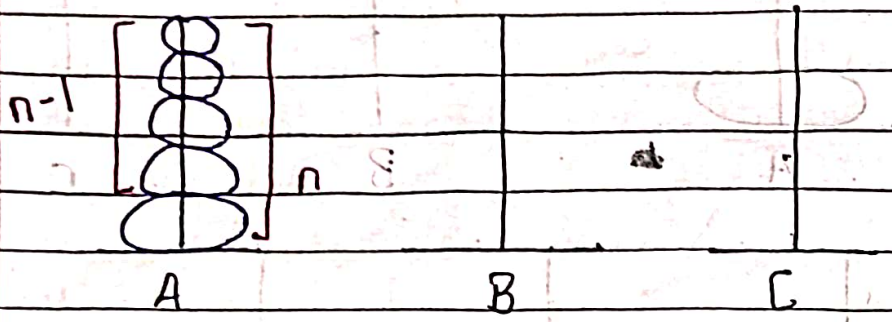
STEP 2



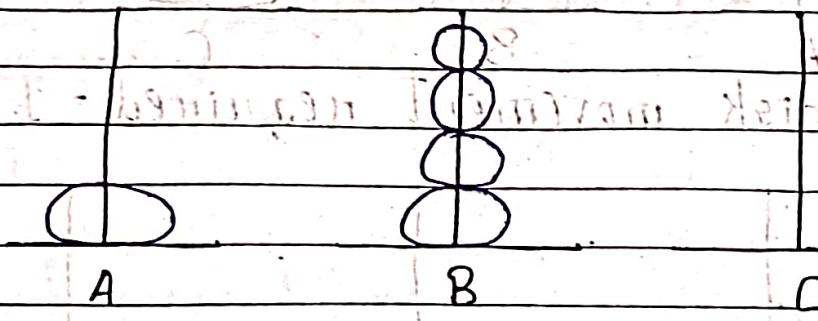
STEP 3



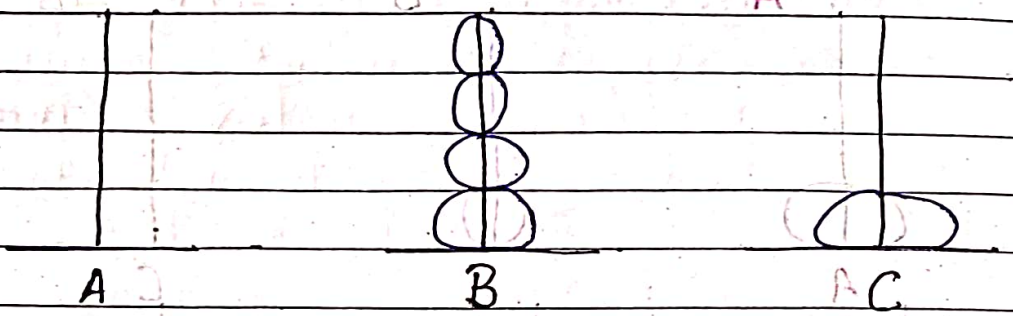
General Form



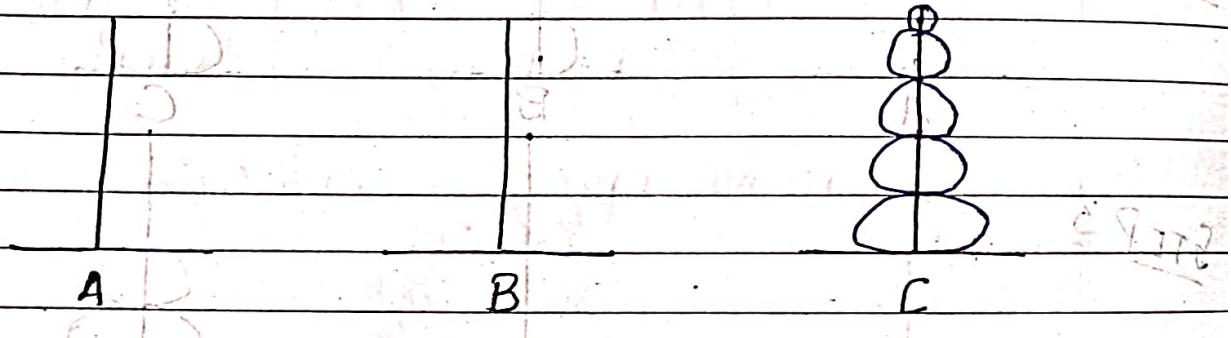
STEP 1 Move (n-1) discs from A to B

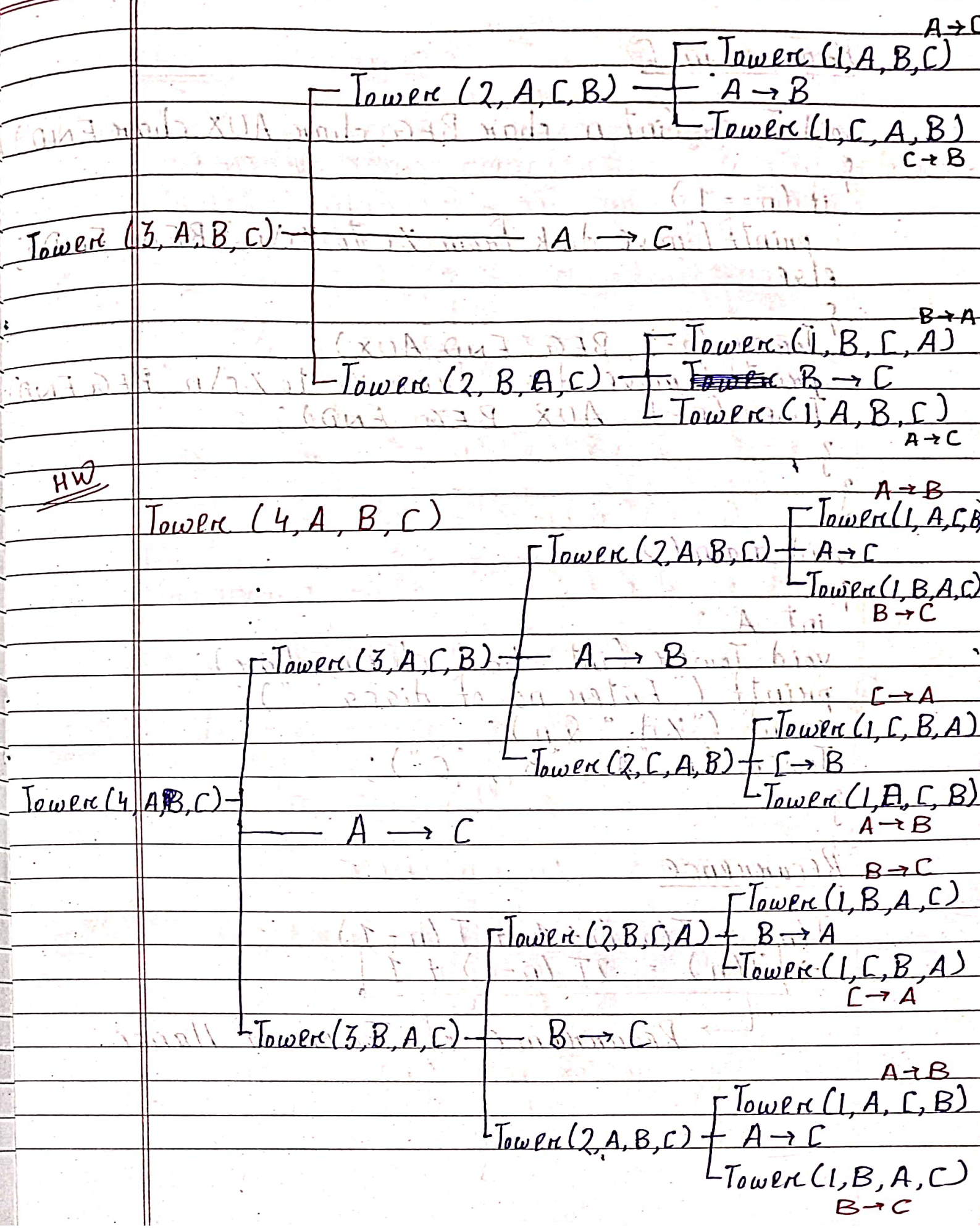


STEP 2 Move the disk from A to C



STEP 3 Move (n-1) discs from B to C




Tower (3, A, B, C)
Tower (2, A, C, B)
Tower (1, A, B, C)
A → B
Tower (1, C, A, B)
A → C
Tower (2, B, A, C)
Tower (1, B, C, A)
B → C
Tower (1, A, B, C)
HW
Tower (4, A, B, C)
Tower (3, A, C, B)
Tower (2, A, B, C)
Tower (1, A, C, B)
A → C
Tower (1, B, A, C)
A → B
Tower (2, C, A, B)
Tower (1, C, B, A)
C → B
Tower (1, A, C, B)
A → B
Tower (4, A, B, C)
A → C
Tower (3, B, A, C)
Tower (2, B, C, A)
Tower (1, B, A, C)
B → A
Tower (1, C, B, A)
C → A
B → C
Tower (2, A, B, C)
Tower (1, A, C, B)
A → C
Tower (1, B, A, C)
B → C

Page No.
Date: / /

Program in C

```
void Tower (int n, char BEG, char AUX, char END
{
  if (n == 1)
    printf ("move disk from %c to %c\n", BEG, END)
  else
  {
    Tower (n-1, BEG, END, AUX);
    printf ("move disk from %c to %c\n", BEG, EN
    Tower (n-1, AUX, BEG, END);
  }
}
void main ()
{
  int A;
  void Tower (int, char, char, char);
  printf ("Enter no. of discs: ");
  scanf ("%d" &n);
  Tower (n, "A", "B", "C");
}
```

Recurrence

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

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

↳ Recurrence of Tower of Hanoi.

11/03/25

Page No.
Date: / /

Master's Theorem

Problem: Write the recurrence of merge sort & derive its time complexity using master's theorem.

A-

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

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

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

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

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

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

Case-I Is $f(n) = O(n^{\log_2 2 - \epsilon})$??

$\Rightarrow n = O(n^{\log_2 2 - \epsilon})$

$\Rightarrow n = O(n^{1-\epsilon})$ FALSE

Case-II Is $f(n) = O(n^{\log_2 2})$??

$\Rightarrow n = O(n^{\log_2 2})$

$\Rightarrow n = O(n)$ TRUE

$\therefore T(n) = \Theta(n^{\log_2 2} \log_2 n)$

$= \Theta(n \log_2 n)$

Problem: Write the binary search recurrence & using the master's theorem, solve the recurrence to find its running time.

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

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

$a = 1, \quad b = 2, \quad f(n) = 1 \quad (n^0)$

Case-I

Is $f(n) = O(n^{\log_b a - \epsilon})$??

$\Rightarrow n^0 = O(n^{\log_2 1 - \epsilon})$

$\Rightarrow n^0 = O(n^{0-\epsilon})$ FALSE

Case-II

Is $f(n) = \Theta(n^{\log_b a})$??

$\Rightarrow n^0 = \Theta(n^{\log_2 1})$

$\Rightarrow n^0 = \Theta(n^0)$ TRUE

$\therefore T(n) = \Theta(n^{\log_2 1} \log_2 n)$

$= \Theta(n^0 \log_2 n)$

$= \Theta(\log_2 n)$

1) $T(n) = 9T\left(\frac{n}{3}\right) + n$ HW

Page No.
Date: / /

Problem:- $T(n) = 3T\left(\frac{n}{4}\right) + n\log_2 n \quad (\log_4 3 = 0.793)$

A-

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log_2 n$$

$a = 3, \quad b = 4, \quad f(n) = n\log_2 n$

Case-I

Is $f(n) = O(n^{\log_b a - \epsilon})$??

$\Rightarrow n\log_2 n = O(n^{\log_4 3 - \epsilon})$

$\Rightarrow n\log_2 n = O(n^{0.793 - \epsilon})$ FALSE

Case-II

Is $f(n) = \Theta(n^{\log_a b})$??

$\Rightarrow n\log_2 n = \Theta(n^{\log_4 3})$

$\Rightarrow n\log_2 n = \Theta(n^{0.793})$ FALSE

Case-III

Is $f(n) = \Omega(n^{\log_b a + \epsilon})$??

$\Rightarrow n\log_2 n = \Omega(n^{\log_4 3 + \epsilon})$

$\Rightarrow n\log_2 n = \Omega(n^{0.793 + \epsilon})$ TRUE

Also, Is $af\left(\frac{n}{b}\right) \leq cf(n)$ for $c < 1$ & $n \to \infty$??

$\Rightarrow 3\left(\frac{n}{4}\log_2 \frac{n}{4}\right) \leq cn\log_2 n$ for $c < 1$ & $n \to \infty$

$\Rightarrow \frac{3}{4}\log_2 \frac{n}{4} \leq c\log_2 n$ for $c < 1$ & $n \to \infty$

$\Rightarrow \frac{3}{4}(\log_2 n - 2) \leq c\log_2 n$ for $c < 1$ & $n \to \infty$

$$\lim_{n\to\infty} \frac{3}{4} \frac{\log_2 n - 2}{\log_2 n} \leq c$$

$$\Rightarrow \frac{3}{4} \lim_{n\to\infty} \left(1 - \frac{2}{\log_2 n}\right) \leq c$$

$$\Rightarrow \frac{3}{4} \left(1 - \frac{2}{\infty}\right) \leq c$$

$$\Rightarrow \frac{3}{4} \leq c \quad \Rightarrow c \in \left[\frac{3}{4}, 1\right) \text{ TRUE}$$

$\therefore T(n) = \Theta(f(n))$

$= \Theta(n \log_2 n)$

Tower of Hanoi Problem

Problem: Write the recurrence for Tower of Hanoi problem & solve it to find its running time or the no. of disk movement.

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

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

Solving by forward Substitution

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

$= 2 + 1$

$= 3 \geq (2^2 - 1)$

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

$= 6 + 1$

$= 7 \geq (2^3 - 1)$

Page No.
Date: / /

Applying induction to find T(n)

If T(k) is true
$\Rightarrow$ T(k+1) is true

Let $T(k) = 2^k - 1$

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

$T(k+1) = 2\,T(k+1-1) + 1$

$= 2\,T(k) + 1$

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

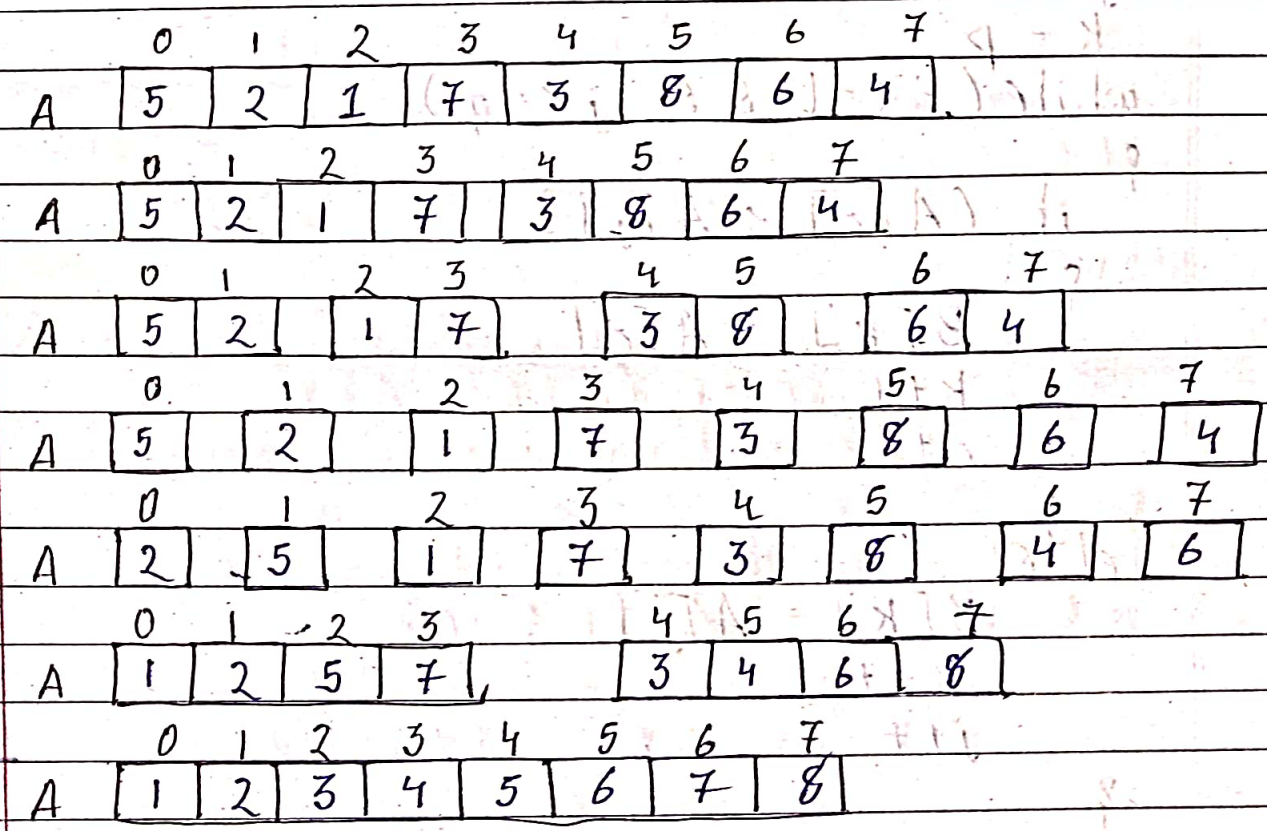
$= 2 \cdot 2^k - 2 + 1$

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

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

17/07/25

Merge Sort



Page No.
Date: / /

```
Algo MergeSort (A, p, q)
{
  P is the start index
  Q is the end index of the array

  if (p < q)
  {
     l = (p+q)/2
     mergeSort (A, p, l)
     MergeSort (A, l+1, q)
     MergeSort (A, p, l, q)
  }
}

Algo Merge (A, p, l, q)
{
   i = p
   j = l+1
   k = p
while (i <= l && j <= q)
{
   if (A[i] < A[j])
   {
      B[k] = A[i]
      k++
      i++
   }
else
      B[k] = A[j]
      k++
      j++
}
```

2	5	9	12

3	6	15	17	18

Page No.
Date: / /

```
while (i<=l)
{
   B[k] = A[i]
   k++
   i++
}
while (j<=q)
{
   B[k] = A[j]
   K++
   j++
}
for (i=p; i<=q; i++)
   {
      A[i] = B[i]
   }
}
```

18/03/25

Quick Sort

(p) 0	1	2	3	4	5	6 (q)
9	2	8	7	12	11	(10)

→ pivot element

Algo partition (A, p, q)

```
{
// Purpose : This algorithm selects x = A[q]
             as the pivot element. Arrange
             x = A[q] in the array such
             that LHS(x) ≤ x & RHS(x) > x.
```

Page No.
Date: / /

```
i = p-1 , x = A[q]
for (j = p to q-1)
{
  if (A[j] ≤ x)
  {
    i = i+1
    swap (A[i], A[j])
  }
}
swap (A[i+1], A[q])
return (i+1)
```

Initialization p = 0 , q = 6 , i = p-1 = 0-1 = -1
x = A[6] = 10

Step 1 j = 0
if (A[0] ≤ 10) TRUE
i = i+1 = -1+1 = 0
swap (A[0], A[0])

STEP 2 j = 1
if (A[i] ≤ 10) TRUE
{
i = i+1 = 0+1 = 1
swap (A[1], A[1])
}

STEP 3 j = 2
if (A[i] ≤ 10) TRUE
{
i = i+1 = 1+1 = 2
swap (A[2], A[2])

Page No.
Date: / /

STEP 4 $j = 3$
if $(A[i] \leq 10)$
{
$i = i + 1 = 2 + 1 = 3$; TRUE
swap $(A[3], A[3])$
}

STEP 5 $j = 4$
if $(A[4] \leq 10)$ FALSE
No operation

STEP 6 $j = 5$
if $(A[5] \leq 10)$ FALSE
No operation

Page No.
Date: / /

```
Algo QuickSort (A, p, q)
// Purpose : This program sorts the
   array A indexed p to q. It calls
   the algo partition.
{
  if (p < q)
  {
    l = partition (A, p, q);
    QuickSort (A, p, l-1)
    QuickSort (A, l+1, q)
  }
}
```

Running Time Analysis

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

Solving, $T(n) = n \log_2 n$

Worst Case Running Time

If the array is already sorted, QuickSort takes max^{m} time to run algorithm

0	1	2	3	4	5
1	2	5	7	9	10

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

After kTh iteration

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

Putting $n-k = 1 \Rightarrow k = n-1$

$\Rightarrow T(n-n+1) + (n-n+1+1) + \ldots + n$
$= T(1) + 2 + 3 + \ldots + n$
$= 1 + 2 + \ldots + n$
$= \frac{n(n+1)}{2}$
$= \frac{n^2}{2} + \frac{1}{2} \simeq n^2$

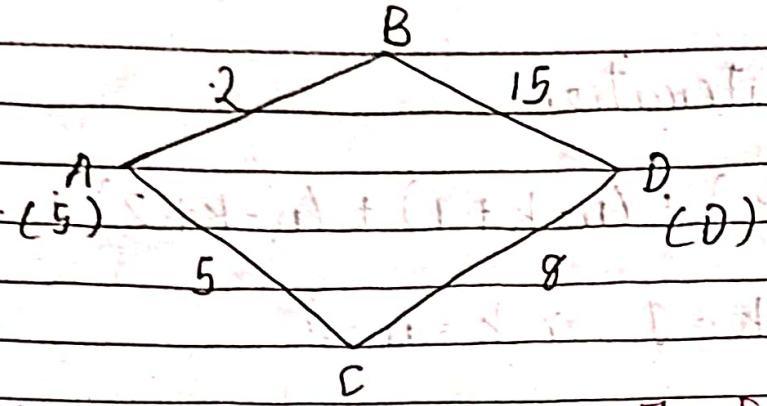
23/03/25

Greedy Technique

It is used for optimization problems. Optimization means either we have to maximise or minimise the value.

How the Greedy Technique Works

You are given with a candidate set. At each iteration, Greedy technique selects the best technique out of the available possibilities. If it's feasible to add that selected element to the sol^n set, then it will be included in the sol^n set.



The best possible path from A to D is
$A \rightarrow C \rightarrow D$ $(5+8 = 13)$
But, Greedy technique chooses the path $A \rightarrow B \rightarrow D$ $(2+15 = 17)$
Here greedy technique fails.

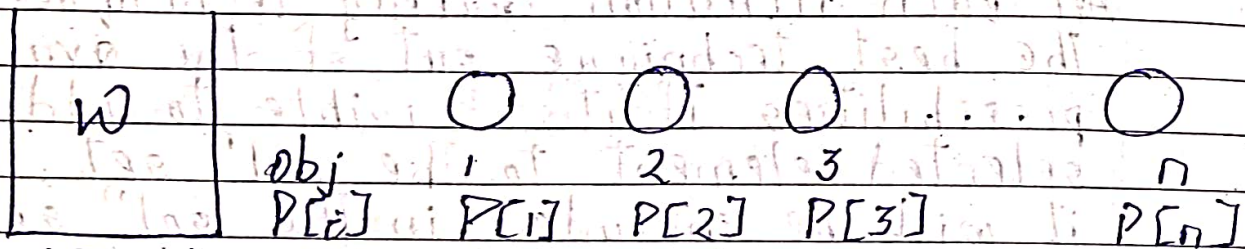
Drawbacks of Greedy Technique

⇒ It sometimes traps at local optima.

Well known Problems using Greedy Technique

1) knapsack Problem
2) Minimum Spinning Tree
3) Huffman Coding

1) knapsack Problem



Page No.
Date: / /

Let a thief entered into a house with a knapsack having weight = W. There are n no. of objects in the house. Each object has a profit & a weight. The thief wants to put the objects into the knapsack so as the earn max^m profit. An object may be taken into fraction

Mathematically,

$$\text{maximise} \sum_{i=1}^{n} P[i]\, x[i]$$

$$\text{constraints} \sum_{i=1}^{n} w[i]\, x[i] \leq W$$

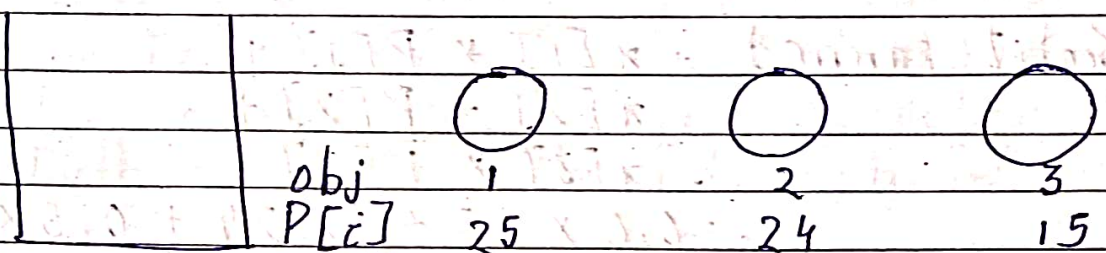
$$\text{and } 0 \leq x[i] \leq 1 \ \forall \ i$$

Problem: W (knapsack weight) = 20

n = 3

$(P_1, P_2, P_3) = (25, 24, 15)$

$(W_1, W_2, W_3) = (18, 15, 10)$



	obj	1	2	3
20	P[i]	25	24	15
	W[i]	18	15	10
	P[i]/W[i] (Relative Profit)	1.38	1.6	1.5

Page No.
Date: / /

STEP 1 i = select the best remaining object
= 2
if Feasible (0 + w[i])
{
x[2] = 1.0
weight = 0.0 + 15
= 15
}

∵ 0 + 15 = 15 < 20
So, feasible ()
return true

STEP 2 i = select the best remaining object
= 3
if Feasible (15 + w[3])
{
x[i] ≠ 1.0
x[3] = $\frac{W - weight}{W[3]}$ $\frac{20 - 15}{10} = \frac{5}{10} = \frac{1}{2}$
}
Weight = 15 + 10 × 1/2
= 20

Since (weight == w)
stop

Profit earned = x[1] * P[1] +
x[2] * P[2] +
x[3] * P[3]
= 0.0 × 25 + 1 × 24 + 0.5 × 15
= 31.5

25/03/25

Page No.
Date: / /

* * *

Huffman Coding

Characters	a	b	c	d	e	f
Frequency (in Thousands)	45	13	12	16	9	5
Variable length encoding	0	101	100	111	1100	110

Huffman code is used for encoding the contents present in a file. It is an encoding technique.

The above example shows the occurance of the diff. characters present in a text file 'a' present 45,000 times, 'b' represent 13000 times & so on. To store the file we need

1,00,000 × 8 bits = 8,00,000 bits

Instead of string each character with 8 bits, if we apply bit-field & store the characters using variable length Huffman coding, the file requires

45,000 × 1 + 13,000 × 3 + 12,000 × 3 + 16,000 × 3 + 9000 × 4 + 5000 × 4

= 45000 + 39000 + 36000 + 48000 + 36000 + 20000

= 2,24,000 bits

Page No.
Date: / /

Huffman coding gives lesser no. of bits whose frequency is more.

For instance, a's frequency is 45,000, gives a single bit 0.
f's frequency is 5000 gives a 4 bit sequence of 1100.

How the algo works?

Eg.

a : 45	b : 13	c : 12	d : 16

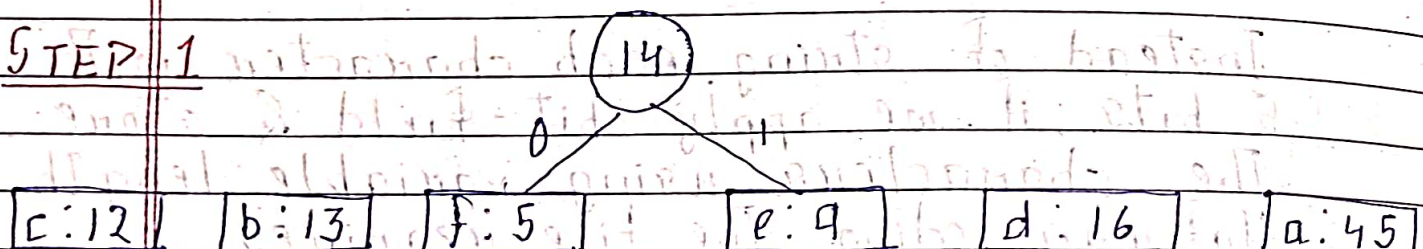
e : 9	f : 5

Initialization: Arrange the characters in ascending order of their frequencies.

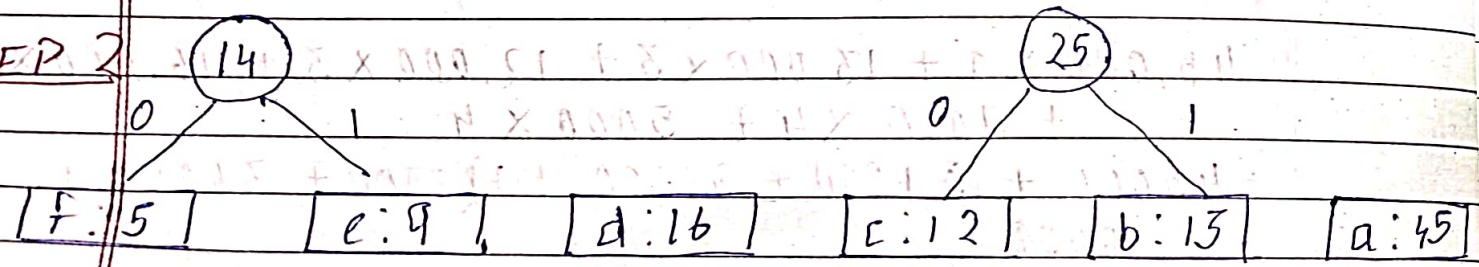
f : 5	e : 9	c : 12	b : 13	d : 16

a : 45

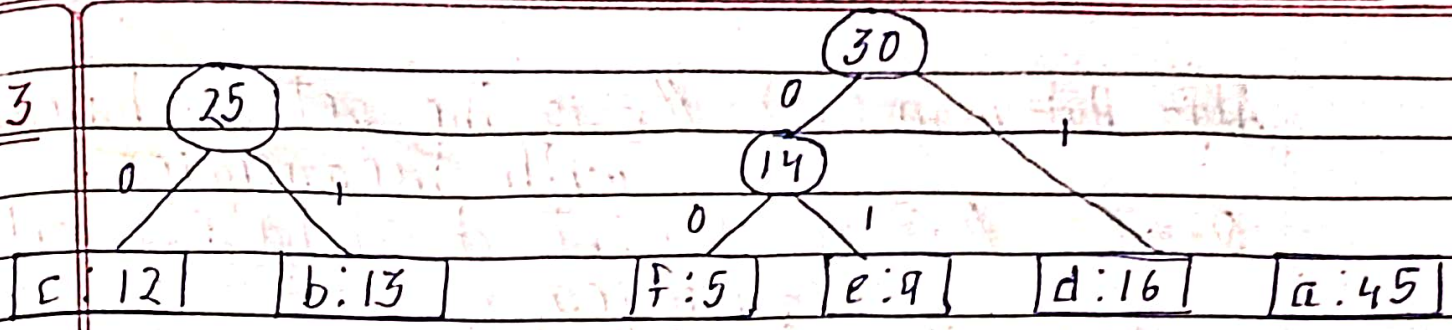
STEP 1



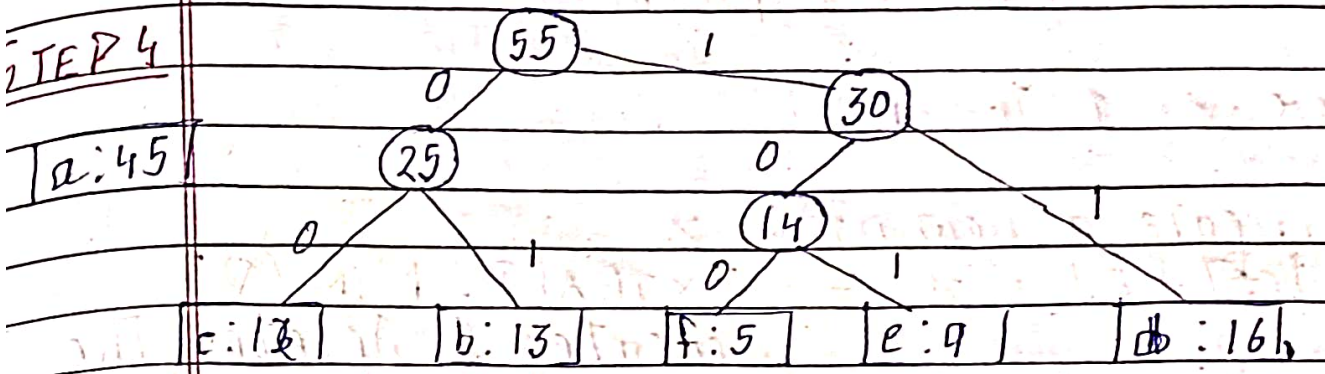
STEP 2



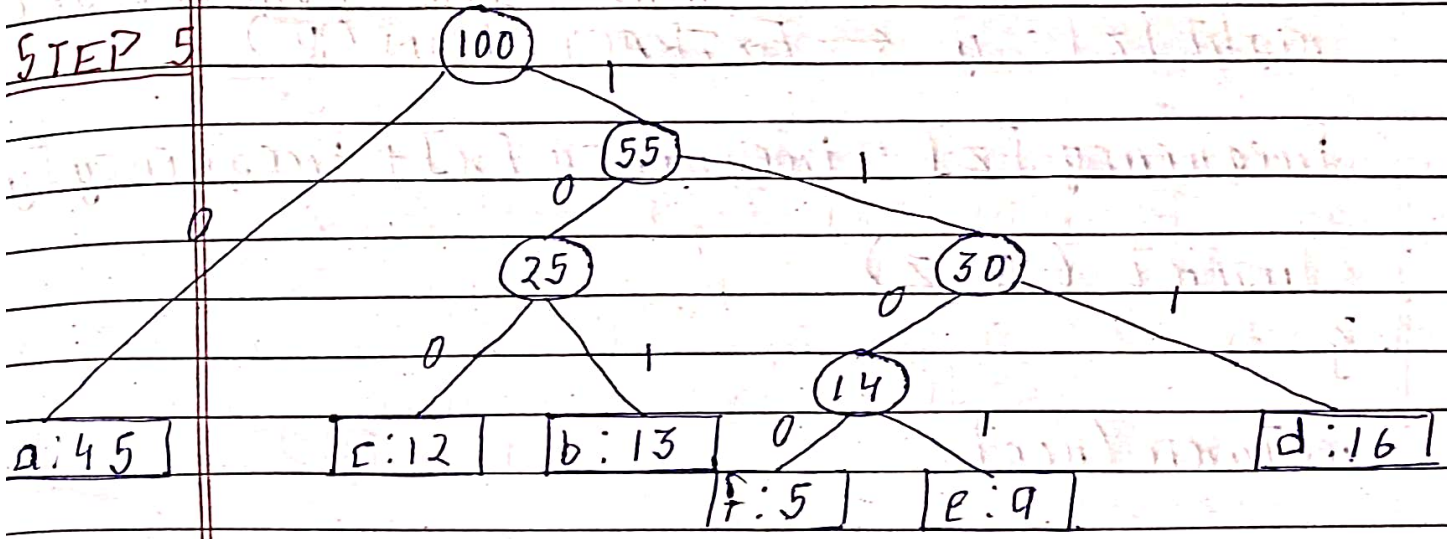
STEP 3



STEP 4



STEP 5



a = 0
b = 101
c = 100
d = 111
e = 1101
f = 1100

Encoding value

Page No.
Date: / /

```
Algo Huffman (C)  // C is the set of characters
                     with frequencies
Q = C             // Q is the set of alphabets with
                     frequencies
n = |C|           // no of alphabets

for i = 1 to (n-1)
{
  create a new node z
  left [z] = x <-- EXTRACT_MIN (Q)
                   creates & deletes the
                   min data from the set
  right [z] = y <-- EXTRACT_MIN (Q)

  frequency [z] = frequency [x] + frequency [y]

  Insert (Q, z)
}
return (root)
```

28/03/25

Page No.
Date: / /

Algorithm for knapsack problem

```
Algo greedy-knapsack
(P[] array of profits
W[] array of weight
x[] is the output array
n no. of objects)

for (i = 1 to n)
{
  x[i] = 0.0 //initialize the x[] array
  weight = 0.0
  while (weight < w)
  {
    i = select the best remaining object
    if (weight + w[i] <= w)
    {
      x[i] = 1.0
      weight = weight + w[i]
    }
    else
    {
      x[i] = (w - weight) / w[i]
      weight = w
    }
  }
}
```

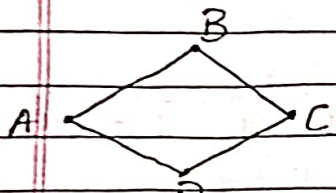
Page No.
Date: / /

Minimum Spinning Tree

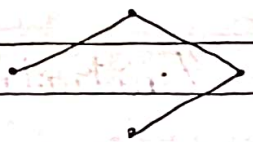
Spanning Tree

It is a tree generated from a graph having all the vertices of the graph.

NOTE Every tree is a graph.

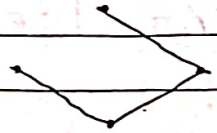


G



T_1

T_2



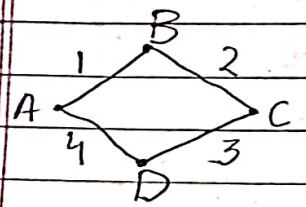
T_3

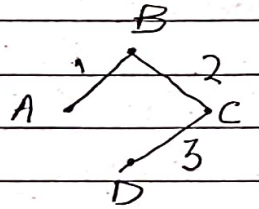
T_4

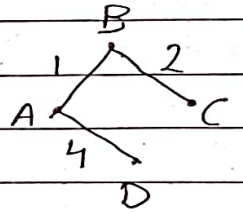
T_1, T_2, T_3, T_4 are the spanning tree of given graph G.

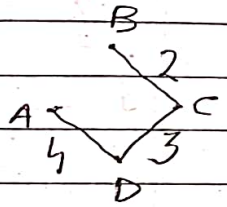
MST

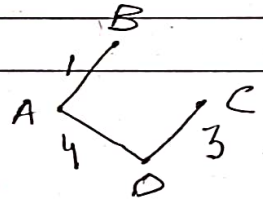
MST is a spanning tree with minimum cost.











Page No.
Date: / /

Cost $T_1 \rightarrow 6$
$T_2 \rightarrow 7$
$T_3 \rightarrow 9$
$T_4 \rightarrow 8$

Here T_1, T_2, T_3 & T_4 are the spanning tree of the graph G with costs 6, 7, 9, 8 respectively.

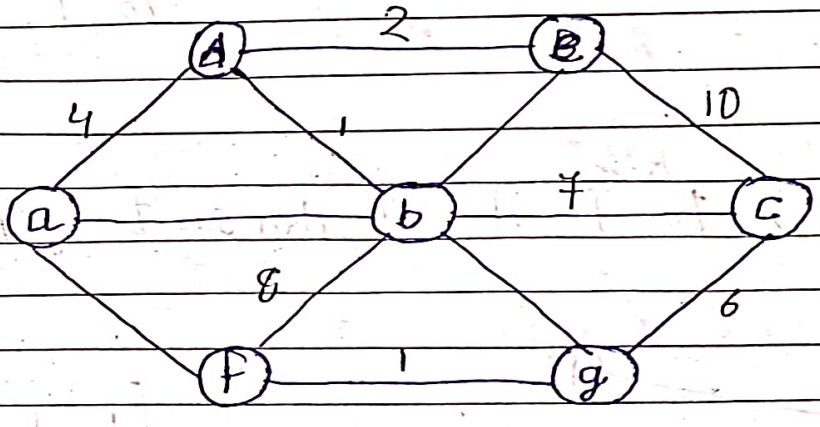
$\therefore$ T_1 is the MST of graph G as it is with min. cost of 6

Algorithm for MST

1) Kruskal's Algorithm
2) Prism's Algorithm

04/04/25

Kruskal's Algorithm



Page No.
Date: / /

STEP 1

Arrange the edges in the following ascending order according to their weight:

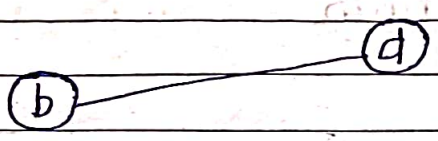
bd (1), gf (1), df (2), ab (2), be (3), bg (4), ad (4), af (5), cg (6), bc (7), bf (8), ec (10)

STEP 2

Create an empty spanning tree, T = ϕ

STEP 3

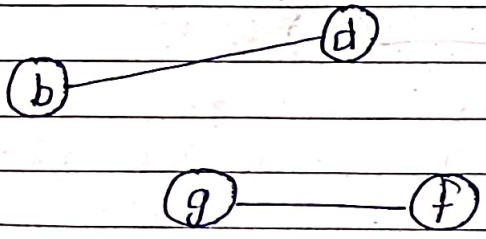
~~Total~~ Add bd (1) to the empty spanning tree T to modify T



b, d are added to T.

STEP 4

Add gf (1) to T if it doesn't generate a cycle

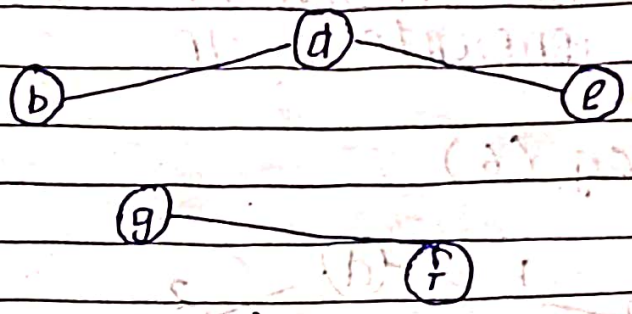


b, d, g, f

Cycle - A path whose source & dest° are same

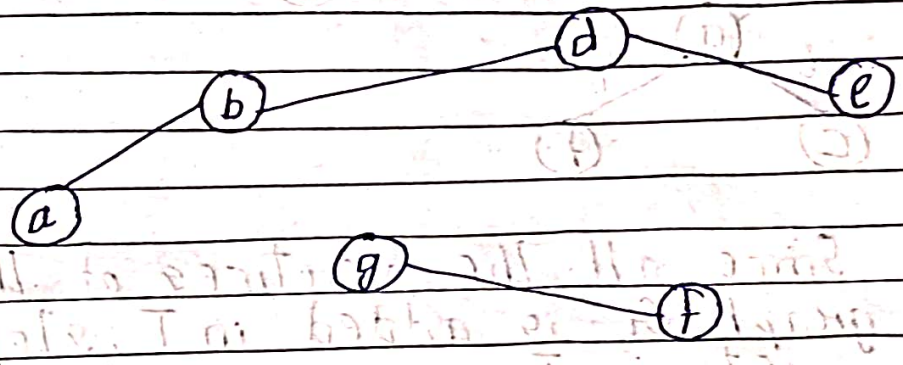
Page No.
Date: / /

STEP 5 de (2)



b, d, g, f, e

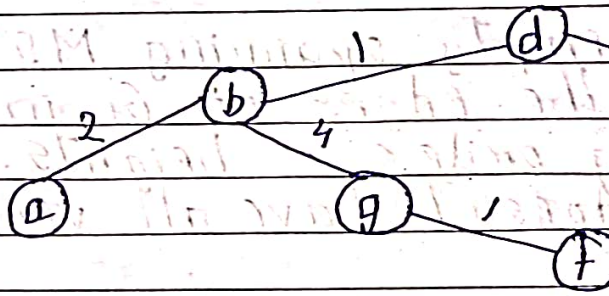
STEP 6 ab (2)



a, b, d, e, f, g

STEP 7 be (3)
is not added in T, as it generate cycle.

STEP 8 bg (4)



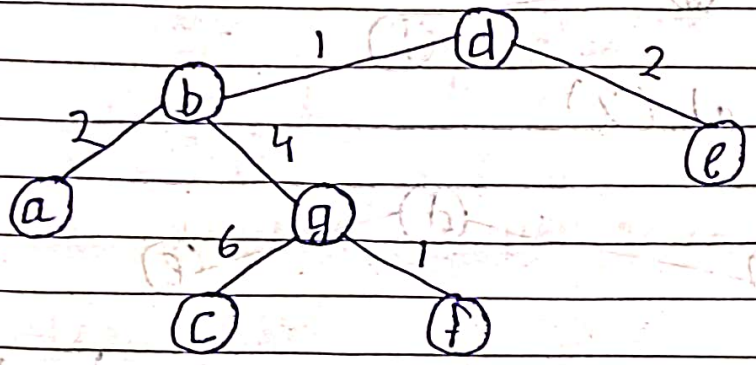
a, b, d, e, f, g

STEP 9 ad (4)
is not added in T as it generate cycle

Page No.
Date: / /

STEP 10 af (5) is not added in T as it generate cycle.

STEP 11 cg (6)



STEP 12 Since all the vertices of the graph G is added in T, stop & return T

∴ cost (T) = 2+1+4+6+1+2
= 16

Algo kruskal's - ~~Algo~~ MST (G)

{

1. Create an empty spanning MST T = Φ
2. Arrange the edges of G in their ascending order of heights.
3. while (T doesn't have all vertices of G)

{

4 Add the min. weight edge into T if it doesn't generate cycle

5 update T

}

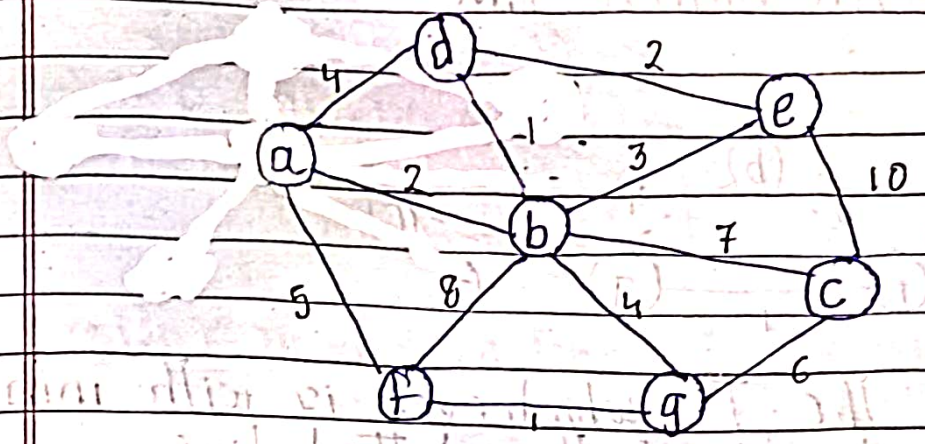
6 return T

7 stop

}

Page No.
Date: / /

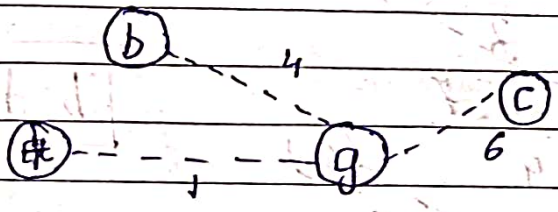
Prism's Algorithm



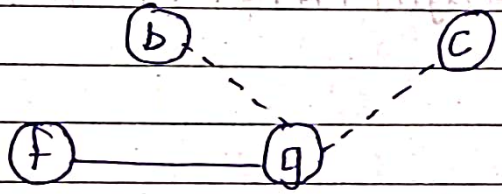
STEP 1 $T = \phi$

select an arbitrary vertex. Let it be 'g'.

STEP 2 Draw all the adjacents of the vertex 'g' in dotted lines.



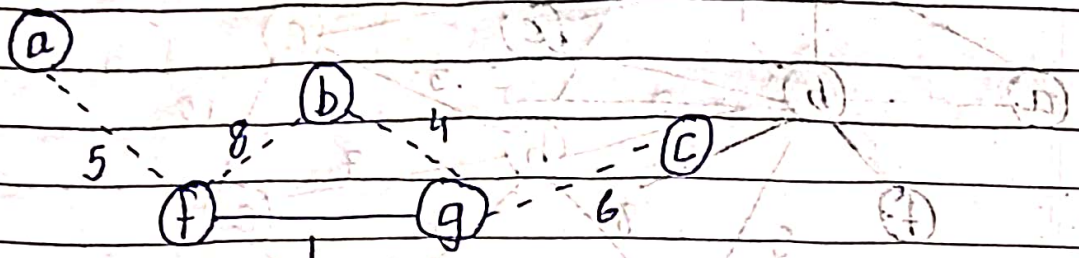
Add the edge into tree T, which has the min. cost.



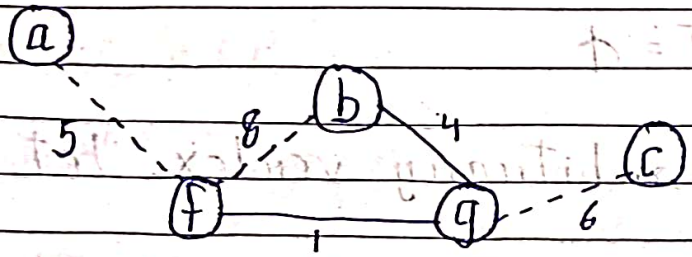
g, f

Page No.
Date: / /

STEP 3 Draw all the adjacents of 'f' with dotted line

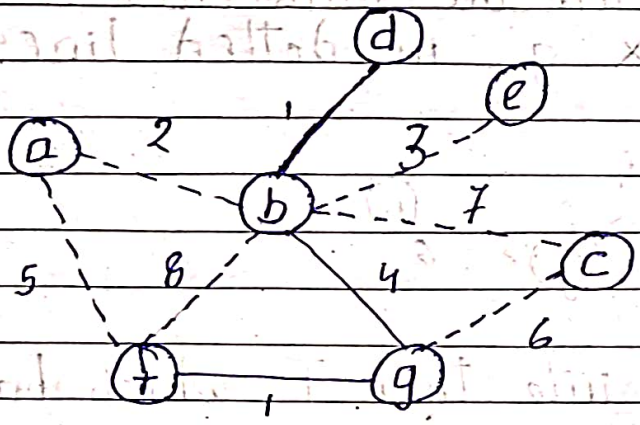


Add into the T, whichever is with min weight out of all the dotted line.



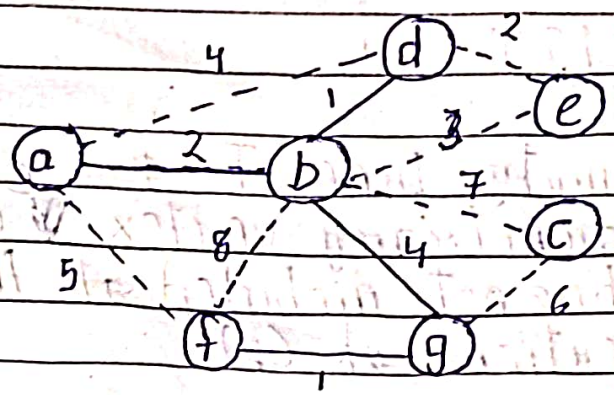
b, g, f

STEP 4 Draw all the adjacents of 'b' with dotted line.



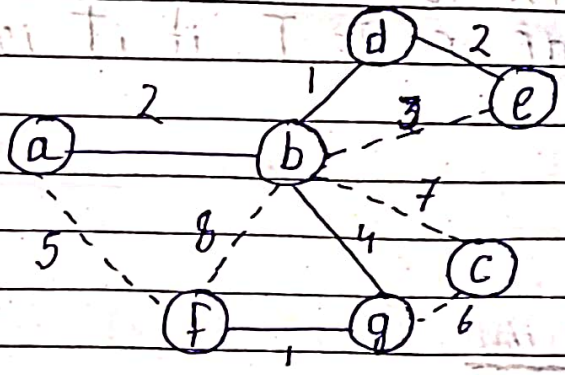
b, d, g, f

STEP 5 Add adjacents of 'd'.



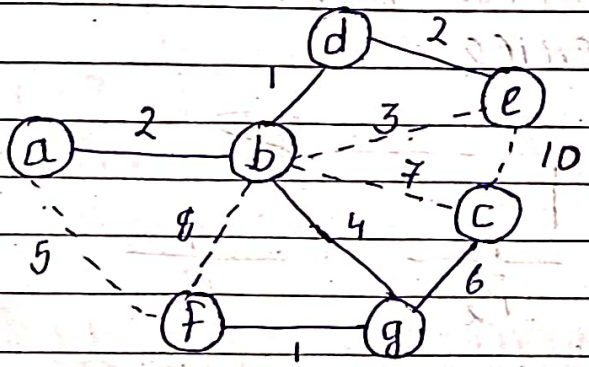
a, b, d, g, f

STEP 6 Add adjacents of 'a'

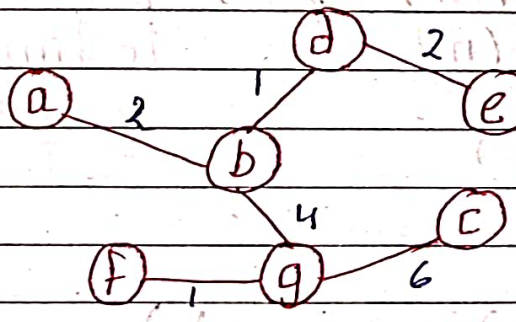


a, b, d, e, f, g

STEP 7 Draw the adjacents of 'e'



a, b, c, d, e, f, g



Since all the vertices of the graph G is in T, stop & return T.

Cost = 2+1+4+1+2+6 = 16

Page No.
Date: / /

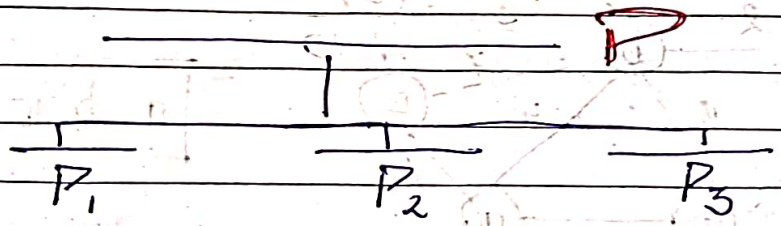
Algo Prism's - MST (G)
{
1. Create an empty tree, T = Φ
2. Select an arbitrary vertex, v ∈ G, T = v
3. while (T has not included all the vertices of G)
{
4. Draw all the adjacent recently added vertex V
5. Add the edge e to T, if it is cyclic.
}
6. return T
7. stop

07/04/25

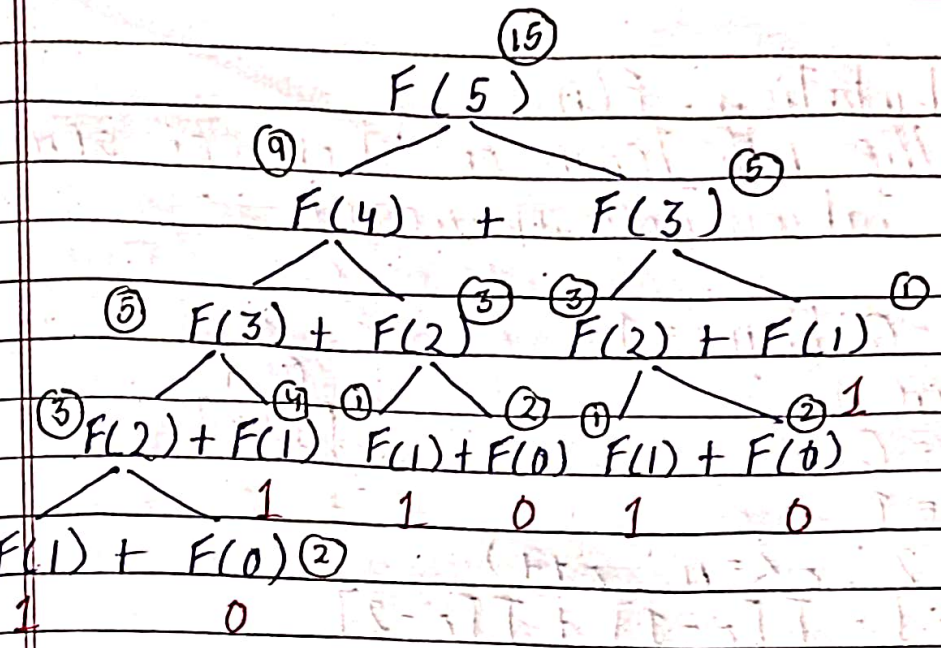
Dynamic Programming

It is also used for optimization problems & used when there is no overlapping in the sub queries

D & C



Algo Traditional - F(n) 0, 1, 1, 2, 3, 5, 8 → to find this.
{
if (n == 0 || n == 1)
return n
else
return (F(n-1) + F(n-2))
}



Problem: WAP to find Factorial of the no.s from 0 To n.

A:

```
Algo Tabulation - Factorial (n)
{
  if (n == 0 || n == 1)
    return 1
  m[0] = 1
  m[1] = 1
  for (i = 2; i <= n; i++)
      m[i] = i * m[i - 1]
}
```

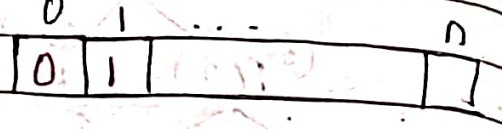
Dynamic Programming Approach

1) Memorization (Top-down approach)
2) Tabulation (Bottom-up approach)

Page No.
Date: / /

Algo tabulation - F(n)
// Find the nth term of fibonacci series using tabulation approach.

```
{
  if (n==0 || n==1)
    return n;
  T[0] = 0
  T[1] = 1
  for(i=2; i<=n; i++)
    T[i] = T[i-1] + T[i-2]
  return T[n]
}
```



Algo memorization - F(n)
// nth term of the Fibonacci series using memorization approach

```
{
  if (m[n] != -1)   //already ~~accepted~~ computed
    return m[n]
  else
    if (n<=1)   //computation needed
      M[0] = n
  else
    M[n] = memorization - F(n-1) +
           memorization - F(n-2)
  return M[n]
}
```

NOTE Before ~~creating~~ calling the func^n create an M[] array & initialize it to -1.

Page No.
Date: / /

Well known DP Problems

1) LCS : (Longest common sequence)
2) 0/1 knapsack Problem
3) Matrix chain multiplication

1) Longest Common Sequence (LCS)

Let $X = \langle A, B, C, B, D, A, B \rangle$
$Y = \langle B, D, C, A, B, A \rangle$

$\langle B, C, A \rangle$ => common subsequent of length = 3.
$\langle B, C, B, A \rangle$ => common subsequent of length = 4.
$\langle B, D, B, A \rangle$ => common subsequent of length = 4.
There is no common subsequent in X & Y of length = 5.

So, $\langle B, C, B, A \rangle$ & $\langle B, D, B, A \rangle$ are said to be LCS (X, Y) of length = 4.

Eg.
$X = A\ B\ A$
$Y = B\ A\ A$

LCS (X, Y) = "AA" or "BA"

$X = A\ B\ A$
$Y = B\ A\ A$

LCS (AB, BA) + 'A' = 'A' or 'B' + 'A'
= 'AA' or 'BA'

A B
B A

max (LCS (A, BA), LCS (AB, B))
= max ('A', 'B')
= 'A' or 'B'

B A
Φ + 'A' = 'A'

A B
B
Φ + 'B' = 'B'

Problem Defⁿ :

Let X = < X₁, X₂, ... Xₙ >
Y = < Y₁, Y₂, ... Yₙ >

Find Z = < Z₁, Z₂, ... Zₙ >
such that Zₖ = LCS (Xₘ, Yₘ)

Problem: X = < A, B, C, B, D, A, B >
Y = < B, D, C, A, B, A >
Find LCS (X, Y)

		0	1	2	3	4	5	6
		y_i	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0 ↑	0 ↑	0 ↑	↖ 1	← 1	↖ 1
2	(B)	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	(C)	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	(B)	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	(A)	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

C [7, 6]

⇓

LCS (X_7, Y_6)

C [0, 0]

⇓

LCS (X_0, Y_0)

C [2, 0]

⇓

LCS (X_2, Y_0)

Let X = ⟨a b c b c f⟩

Y = ⟨a b c d a f⟩

Let Z = ⟨a c⟩ is a common subsequence of both X & Y & is of length = 2.

Another subsequence Z = ⟨a c f⟩ is in both X & Y of length = 3.

Z = ⟨a b c f⟩ is another common subsequence of length = 4

∴ So, ⟨a b c f⟩ is said to be the LCS of X & Y.

$X_3 = A\ B\ B$
$Y_3 = B\ A\ B$

$Z = LCS(AB, BA) + 'B' = 'B'$ or $'A' + 'B'$
$= BB'$ or AB'

$A\ B$ X
$B\ A$

$\max(LCS(AB, B), LCS(A, BA))$
$= 'B'$ or $'A'$

Problem: Given a 2 strings $\langle a\ c\ b\ c\ f \rangle$ & $\langle a\ b\ c\ d\ a\ f \rangle$. Find the LCS of the given strings.

$X_m = \langle a\ c\ b\ c\ f \rangle$, $m = 5$
$Y_n = \langle a\ b\ c\ d\ a\ f \rangle$, $n = 6$

Draw a matrix of size 6×7

		0 y_i	1 a	2 b	3 c	4 d	5 a	6 f
0	x_i	0	0	0	0	0	0	0
1	a	0	↖ 1	← 1	← 1	← 1	↖ 1	← 1
2	c	0	↑ 1	↑ 1	↖ 2	← 2	← 2	← 2
3	b	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 2	↑ 2
4	c	0	↑ 1	↑ 2	↖ 3	← 3	← 3	← 3
5	f	0	↑ 1	↑ 2	↑ 3	↑ 3	↑ 3	↖ 4

15/04/25

Page No.
Date: / /

```
Algo LCS (X, Y)
{
// X & Y are the input strings
   m = length (X)
   n = length (Y)
   create a matrix of size C (m+1, n+1),
                           B (m+1, n+1)
   for (i = 0; i < m ; i++) // 1st row filled with 0's
      c [0, i] = 0
   For (j = 1 , j <= n ; j++) // 1st col filled with 0's
      c [j, 0] = 0
   For (i = 1 ; i <= m ; i++)
     for (j = 1 ; j <= n , j++)
       if ( X [i] == Y [j])
       {
          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-1, j]
          b [i, j] = ←
   return (c, b)
}
```

Page No.
Date: / /

Algo Print Path (b, X, i, j)
{
// Input : b contains the arrow move matrix (↖ ↑ ←)
X is the input string
i & j are the length of X & Y
{
if (i == 0 || j = 0) // if the length of X & Y is 0 then return from the func
return
if (b[i, j] == '↑')
Print Path (b, X, i-1, j)
else if (b[i, j] == '←')
Print Path (b, X, i, j-1)
else
{
Print Path (b, X, i-1, j-1)
display (X[i])
}
}

3) Matrix Chain Multiplication

$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}_{2\times 2} \qquad \begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}_{3\times 3}$$

Multiplication NOT POSSIBLE

no. of cols of 1st matrix = no. of rows of 2nd matrix
then matrix Multiplication POSSIBLE

Page No.
Date: / /

$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}_{2\times 2} \quad \begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & 1 \end{pmatrix}_{2\times 3}$$

$$= \begin{pmatrix} 1.0+2.1 & 1.1+2.0 & 1.3+2.1 \\ 2.0+3.1 & 2.1+3.0 & 2.3+3.1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 1 & 5 \\ 3 & 2 & 9 \end{pmatrix}_{2\times 3}$$

No. of scalar multiplication
$= 2 \times 2 \times 3$
$= 12$

NOTE $(A)_{m\times n} \times (B)_{n\times p} = (C)_{n\times p}$

No. of scalar multiplication $= m \times n \times p$

(A_1, A_2, A_3)

1) $A_1 (A_2 A_3)$
2) $((A_1 A_2) A_3)$

$A_1 A_2 A_3 A_4$

1) $((A_1 (A_2 A_3))(A_4))$
2) $((A_1 A_2)(A_3 A_4))$
3) $(A_1((A_2 A_3) A_4))$
4) $(A_1 (A_2 (A_3 A_4)))$

Problem Def^n :

Let we want to multiply the matrices from A_1 to A_n where a matrix A_i is denoted by the dimension $P_{i-1} \times P_i$. Find the optimal way of getting $A_{1\ldots n}$ such that the no. of scalar multiplications should be minimum.

21/04/25

Page No.
Date: / /

2) 0/1 knapsack Problem

Maximize $\sum_{i=1}^{n} P_i x_i$ → object func^n

subject to
1) $\sum_{i=1}^{n} w_i x_i \leq W$
2) $x_i \in \{0, 1\}$

Problem: Maximize the profit where $W = 12$ (knapsack weight)
And the given objects are -

P_i	1	6	22	18	43	28
W_i	1	2	6	5	10	7

A- $\frac{P_i}{W_i}$ = 1.00 3.00 3.67 3.60 4.30 4.00

K [6] [13]
↓ ↓
no. of objects w+1

K[i][j] = The value earned if the weight of the knapsack is j & i no. of objects are taken into consideration.

Eg. K[3, 5] = ?
The profit formed if
i) weight of knapsack = 5
ii) No. of weight existences = 3.

Page No.
Date: / /

Objects \ Weights	0	1	2	3	4	5	6	7	8	9	10	11	12
$W_1=1, P_1=1$ $\frac{P_1}{W_1}=1.00$	0	1	1	1	1	1	1	1	1	1	1	1	1
$W_2=2, P_2=6$ $\frac{P_2}{W_2}=3.00$	0	1	6	7	7	7	7	7	7	7	7	7	7
$W_3=5, P_3=18$ $\frac{P_3}{W_3}=3.60$	0	1	6	7	7	18	19	24	25	25	25	25	25
$W_4=6, P_4=22$ $\frac{P_4}{W_4}=3.67$	0	1	6	7	7	18	22	23	28	29	29	40	41
$W_5=7, P_5=28$ $\frac{P_5}{W_5}=4.00$	0	1	6	7	7	18	22	28	29	34	35	35	46
$W_6=10, P_6=43$ $\frac{P_6}{W_6}=4.30$	0	1	6	7	7	18	22	28	29	34	43	44	49

Formula :-

$$K[i,j] = \max\{K[i-1,j],\ K[i,j-w[i]] + P[i]\}$$

- $K[i-1,j]$ → The ith object is not considered
- $K[i,j-w[i]] + P[i]$ → if the ith object is considered.

22/04/25

Page No.
Date: / /

Problem:

	value	weight	$
Microscope	$ 300	2 kg	150
Globe	$ 200	1 kg	200
Cup	$ 400	5 kg	80
Crown	$ 500	3 kg	166.67

$W = 10$ kg

Implement 0/1 knapsack problem to find the max^m value control.

Objects	Weights 0	1	2	3	4	5	6	7	8	9	10
$P_1 = 5$ $P_1 = 400$ $P_1 = 80$ P_1	0	0	0	0	0	400	400	400	400	400	400
$P_2 = 3$ $P_2 = 300$ $P_2 = 150$ P_2	0	0	300	300	300	400	400	700	700	700	700
$P_3 = 3$ $P_3 = 500$ $= 166.67$	0	0	300	500	500	800	800	800	900	900	1200
$P_4 = 1$ $P_4 = 200$ $= 200$	0	200	300	500	700	800	1000	1000	1000	1100	1200

Page No.
Date: / /

Heap Sort

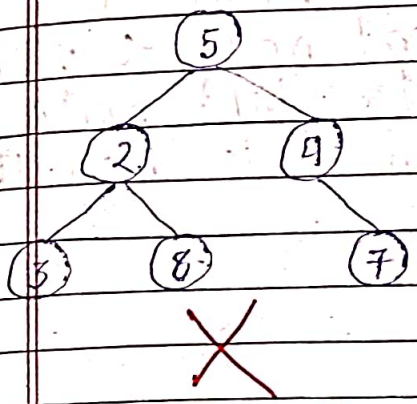
Heap Data Structure

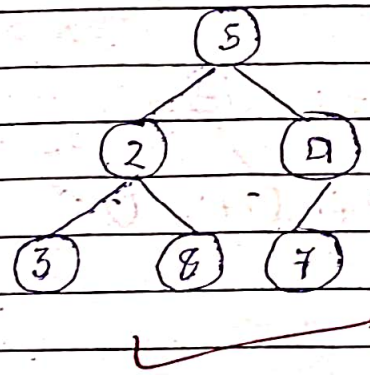
Heap is a DS which follows 2 properties:
1) Shape Property
2) Order Property

28/04/25

Shape Property

A heap must be a complete binary tree





Complete binary tree but not a heap.

Complete Binary Tree

For a binary tree with height h, upto height(h-1) it must be full with nodes & the last level must have the nodes from left to right

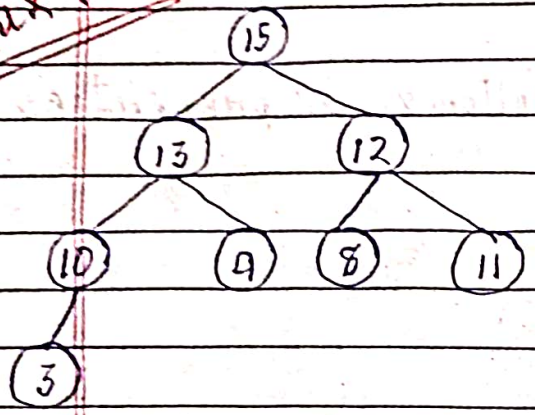
Order Property

Following the order property, heap DS is divided into 2 categories:-

Page No.
Date: / /

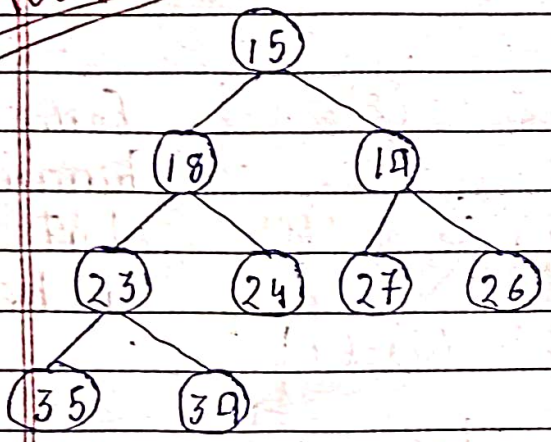
1) Max Heap
2) Min Heap

Max Heap



If the value of parent node > value of its child node then it is Max Heap.

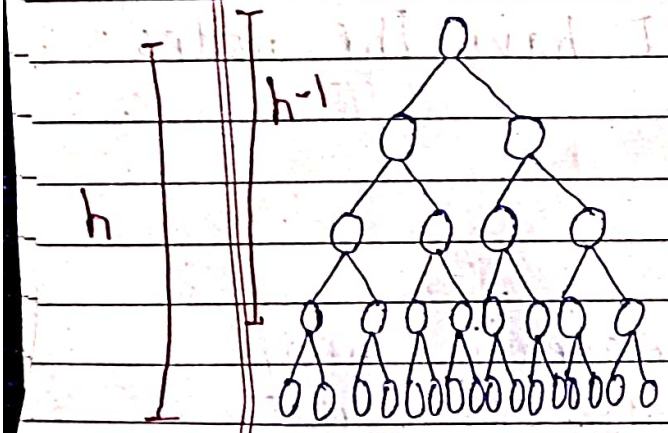
Min Heap



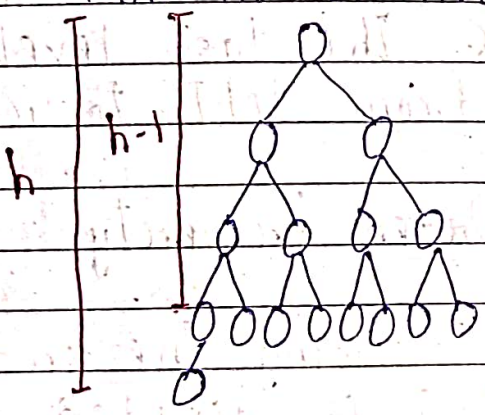
If the value of parent node < value of its child node then it is Min Heap.

Problem: Find the max^m & min. no. of nodes present in a heap of height h?

Max^m no. of nodes



Min no. of nodes



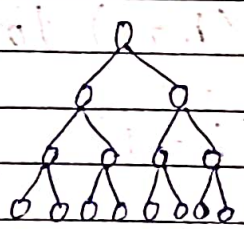
G.P $\frac{a[r^{n+1}-1]}{r-1}$

Page
Date: /

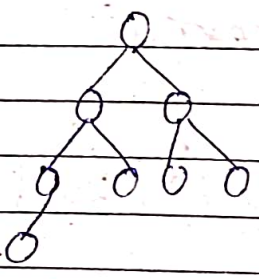
Max^m no. of nodes
$= 2^0 + 2^1 + 2^2 + \dots + 2^h$ (h+1) Terms
$= \frac{2^0(2^{h+1}-1)}{2-1}$
$= 2^{h+1} - 1$ //

Min. no. of nodes
$= \underbrace{2^0 + 2^1 + \dots + 2^{h-1}}_{h \text{ terms}} + 1$
$= \frac{2^0(2^h - 1)}{2-1} + 1$
$= 2^h - 1 + 1 = 2^h$ //

Eg. $h = 3$



$2^{3+1} - 1$
$= 2^4 - 1$
$= 15$ //
} Max^m



2^3
$= 8$
} Min.

Problem: Prove that an n-element heap has height $= \lceil \log_2 n \rceil$

$5 \leq 5.1 < 6$

$\Rightarrow 5 = \lfloor 5.1 \rfloor$

Page No.
Date: / /

We know

The max^m no. of nodes in a heap $= 2^{h+1} - 1$

The min. no. of nodes in a heap $= 2^h$.

Let n be the no. of nodes in a heap.

$$2^h \leq n \leq 2^{h+1} - 1$$

$$\Rightarrow 2^h \leq n < 2^{h+1} \quad [\because 5 \leq 6-1 \Rightarrow 5 < 6]$$

Taking $\log_2$ in the inequality

$$\log_2 2^h \leq \log_2 n < \log_2 2^{h+1}$$

$$\Rightarrow h \log_2 2 \leq \log_2 n < (h+1) \log_2 2$$

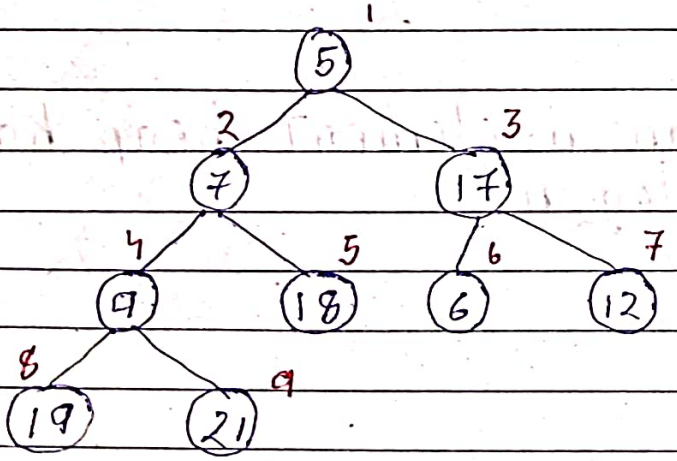
$$\Rightarrow h \leq \log_2 n < (h+1)$$

$$\therefore h = \lfloor \log_2 n \rfloor$$

(Proved)

29/04/25

1	2	3	4	5	6	7	8	9
5	7	17	9	18	6	12	19	21



Page No.
Date: / /

Algo Max-Heapify (A, i)
// Purpose :- It creates a maxheap at idx i.

l = LCHILD (i) // returns the idx of left child
r = RCHILD (i) // returns the idx of right child

max = i

if (l ≤ heapsize (A) && A[l] > A[max])
    max = l

if (r ≤ heapsize (A) && A[r] > A[max])
    max = r

if (i ≠ max)
[swap (A[i], A[max])
[Max-heapify (A, max)

Algo Build-Heap (A)
// Purpose: It creates a max heap for the array A by calling the max-heapify algorithm for all non-leaf nodes

for i = $\left\lfloor \frac{\text{heapsize}(A)}{2} \right\rfloor$ down to 1 step-1
Max-heapify (A, i)

Page No.
Date : / /

Execute Max-heapify algorithm for i = 2

l = LCHILD(i) = 4
r = RCHILD(i) = 5
max = 2

Algo LCHILD(i)
return (2i+2)
Algo RCHILD(i)
return (2i+3)

if (4 ≤ 9 (T) && A[4] > A[2] (T))
max = 4

if (5 ≤ 9 (T) && A[5] > A[4] (T))
max = 5

if (2 ≠ 5 (T))
swap (A[2], A[5])
Max-heapify (A, 5)

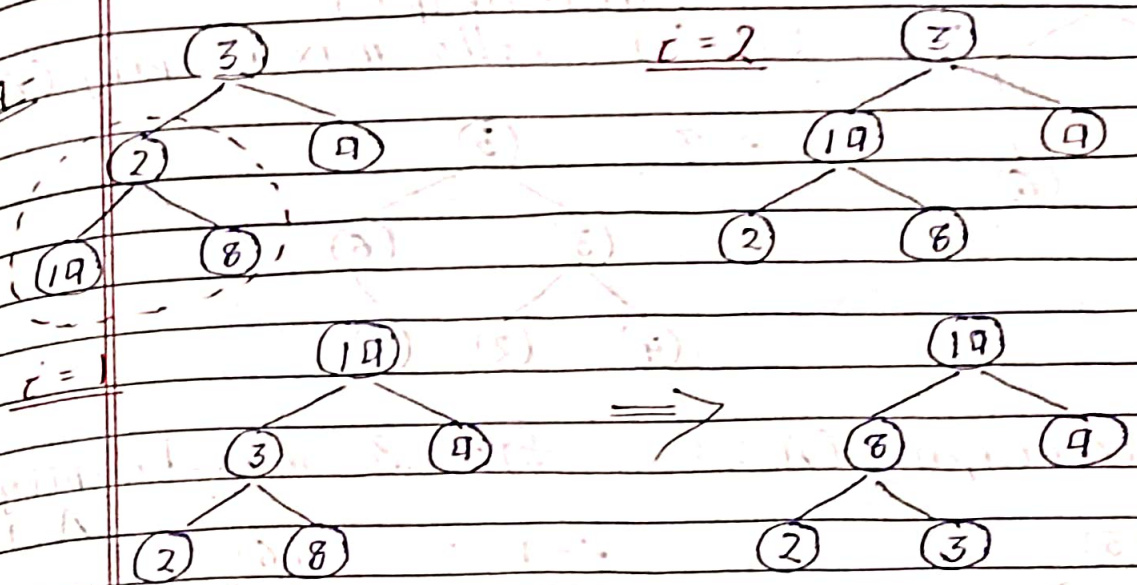
Max-heapify (A, 5)
{
l = LCHILD(i) = 2×5+2 = 12
r = RCHILD(i) = 2×5+3 = 13
max = 5

if (12 ≤ 9 && ...) ✗
if (13 ≤ 9 && ...) ✗
if (5 ≠ 5) ✗
{ ✗
}
}

Problem: Execute Build Heap algorithm for the following array

1	2	3	4	5
3	2	9	19	8

A:



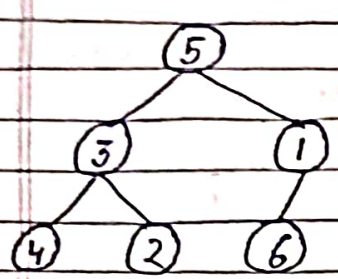
02/05/25

Algo Heapsort (A)
// This algorithm sorts the array element in ascending order. This algorithm calls BuildHeap() algorithm once & max-heapify() algorithm iteratively.

Build-Heap ()
for $i = \left\lfloor \frac{heapsize(A)}{2} \right\rfloor$ down to 2 step -1
{
  swap (A[1], A[i])
  heapsize (A)--
  Max-heapify (A, 1)
}

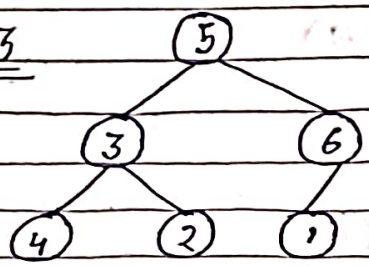
Page No
Date : / /

Problem: A = {5, 3, 1, 4, 2, 6}
Execute heapsort() algorithm to sort A in ascending order.



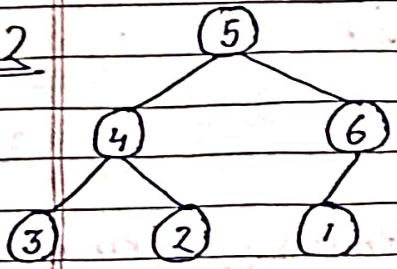
Build-Heap()
STEP-1 calls max-heapify (A, 3)

i = 3



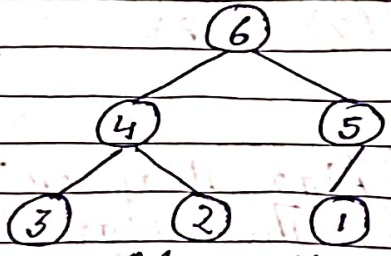
STEP 2 max-heapify (A, 2)

i = 2



STEP 3 max-heapify (A, 1)

i = 1

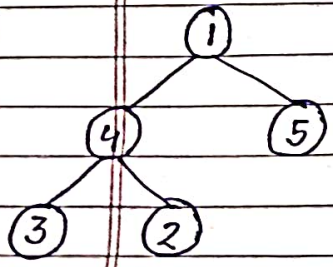


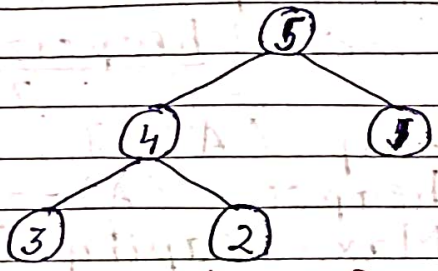
Max Heap

Loop will execute

STEP 4 i = heapsize(A) = 6
swap (A[1], A[6])

1	4	5	3	2	6





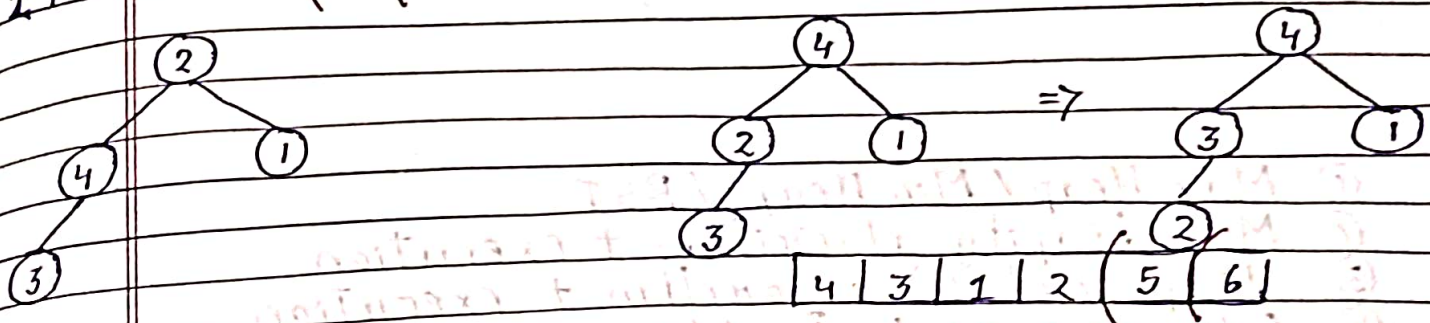
max-heapify (A, 1)

Page No.
Date : / /

STEP 5 i = heapsize (A) = 5
swap (A[1], A[5])

| 2 | 4 | 1 | 3 | 5 | 6 |

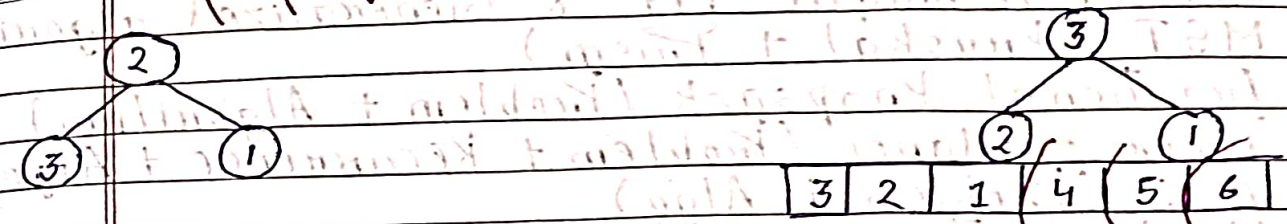
max-heapify (A, 1)



STEP 6 i = heapsize (A) = 4
swap (A[1], A[4])

| 2 | 3 | 1 | 4 | 5 | 6 |

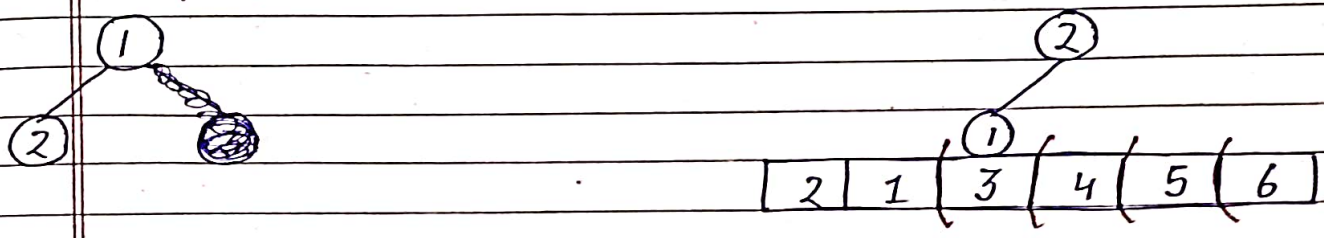
max-heapify (A, 1)



STEP 7 i = heapsize (A) = 3
swap (A[1], A[3])

| 1 | 2 | 3 | 4 | 5 | 6 |

max-heapify (A, 1)



Page No.
Date: / /

STEP 8 c = heapsize (A) = 2
swap (A[1], A[2])

1	2	3	4	5	6

1. Max Heap / Min Heap / BST
2. Max-heapify algorithm + execution
3. Heap sort () algorithm + execution
4. 0/1 knapsack Problem
5. Huffmann Coding
6. DP (Defⁿ, Tabulation + memorization method for a term of Fibbonacci sequence)
7. Matrix Chain Multiplication (Problem)
8. Greedy Technique Defⁿ & Generalized algorithm
9. MST (kruskal + Prism)
10. Fractional knapsack (Problem + Algorithm)
11. Tower of Hanoi (Problem + Recurrence + Algo)
12. LCS (Problem + Algo)
13. Master's Theorem Defⁿ + Problem
14. Recurrence Solving
15. Quicksort (Algo + Tracing + Partition Algo + Running Time)
16. Mergesort (Algo + Running Time)

Page No.
Date: / /

Problem

A_5	10×2	$P_4 \cdot P_5$
A_6	2×5	$P_5 \cdot P_6$
A_7	5×10	$P_6 \cdot P_7$
A_8	10×6	$P_7 \cdot P_8$

Find the optimal way of $A_{5 \ldots 8}$.

A:-

$$m[5,8] = \min \begin{cases} m[5,5] + m[6,8] + P_4 P_5 P_8 \\ m[5,6] + m[7,8] + P_4 P_6 P_8 \\ m[5,7] + m[8,8] + P_4 P_7 P_8 \end{cases}$$

$$= \begin{cases} 0 + m[6,8] + 10 \times 2 \times 6 = m[6,8] + 120 \\ 10 \times 2 \times 5 + 5 \times 10 \times 6 + 10 \times 5 \times 6 = 700 \\ m[5,7] + 0 + 10 \times 10 \times 6 = m[5,7] + 600 \end{cases} \quad \text{---} (1)$$

$$m[6,8] = \begin{cases} m[6,6] + m[7,8] + P_5 P_6 P_8 \\ m[6,7] + m[8,8] + P_5 P_7 P_8 \end{cases}$$

$$= \begin{cases} 0 + 5 \times 10 \times 6 + 2 \times 5 \times 6 = 360 \\ 2 \times 5 \times 10 + 0 + 2 \times 10 \times 6 = 220 \end{cases} \quad \text{---} (2)$$

$$m[5,7] = \begin{cases} m[5,5] + m[6,7] + P_4 P_5 P_7 \\ m[5,6] + m[7,7] + P_4 P_6 P_7 \end{cases}$$

$$= \begin{cases} 0 + 2 \times 5 \times 10 + 10 \times 2 \times 10 = 300 \\ 10 \times 2 \times 5 + 0 + 10 \times 5 \times 10 = 600 \end{cases} \quad \text{---} (3)$$

Putting (2) & (3) in (1)

$$m[5,8] = \begin{cases} 340 \\ 700 \\ 900 \end{cases} \min.$$

(Ans.)