28.07.2022 Thursday

Date: / / Page no:

PERFORMANCE

- Space
- Time

Light wt. compilers → less memory to store

Available 5.01×10^8 bytes

Required Bytes: $10^6 + (N+E)100$ Time

No. of Edge

Scalability → What will happen if no. of i/p ↑ses.

$$10^2(10^4 + (N+E)) = 5.01 \times 10^8$$

$$\Rightarrow 10^4 + (N+E) = 5.01 \times 10^{6\,4}$$

$$\Rightarrow (N+E) = 500 \times 10^4 = 5 \times 10^6$$

$$\Rightarrow \boxed{(N+E)_{max} = 5 \times 10^6}$$

Parameters of Space Complexity:-

1. Instruction: After compiling
2. Data space: Var. & arrays (auxiliary)
3. Dynamic space: Runtime (Dynamic DS → linked list)
   instance characteristics of program depends on i/p
4. Environmental stack space: Remain for call depends on i/p — n, n-1 stacks

Static part DS do not change much but dynamic DS change a lot.

Q) For finding max. in an array, space complexity is constant.

Sol:

Scanned with CamScanner

Date: / / Page no:

Real Time Response is imp. even though Space ↑ ses

Types

~~Parameters~~ of Time Complexity's - Analysis

① Experimental

↳ Real time checking of program's running time

② Apriori → Asymptotic Analysis

knowing about the time by just the analysis of pseudo-code.

ex.

```
current max = A[0]                          ——→ 1
for i=1 to n-1 do                           ——→ (n)c1
    if A[i] > current max then              ——→ (n-1)c2 constant
        current max ← A[i]                  ——→ (n-1)c3 (other primitive operation in for).
    { increment counter }                   ——→ (n-1)
return current max
```

↳ worst case

Operation that can be done in constant amt. is Primitive Operation.

Total no. of steps = $1 + nc_1 + (n-1)c_2 + (n-1)c_3 + (n-1)$

$\approx 4n + c$

↓ Size of i/p

$\Rightarrow f(n) = 4n + c$ C: Constant

Time in executing f^n changes linearly with n.

Function f(n) & g(n)

f(n) is O g(n) iff there exist some constant c such that $f(n) \leq c \cdot g(n)$ for all $n > n_0$

Scanned with CamScanner

Date: / / Page no:

Let $f(n) = 4n+2$
$g(n) = n$

$f(n) \le c\, g(n)$
$4n+2 \le cn$ $\quad c=5$

$g(n)$ = highest degree term of $f(n)$

ex: $f(n) = \underbrace{10n^2}_{g(n)} + 5n + 6$

Method

n : the integer

$4n+2 \le 4n + 2n$

$4n+2 \le 6n$ $\quad \therefore \boxed{c = 6}$

ex. $f(n) = 10n^2 + 5n + 6$ $\quad \therefore g(n) = 10n^2$

$f(n) \le c\, g(n)$

$\Rightarrow 10n^2+5n+6 \le 10n^2 + 5n^2 + 6n^2$

$\Rightarrow 10n^2+5n+6 \le 21n^2$

$\le (2.1)\, \underbrace{10n^2}_{g(n)}$ $\quad c = 2.1$

Lower order terms have lesser impact ~~on~~ that highest order term for

Q) $f(n) = n^3 + 1999n + 2000$ beyond which value of n, n^3 become greater than $1999n$

Sol: $n^3 + 1999n + 2000$ $\quad n^3 > 1999n$

$n(n^2 - 1999) > 0$

Scanned with CamScanner

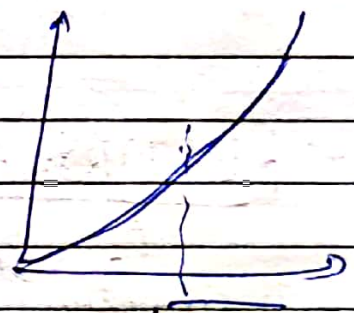
Date: / / Page no:

$n(n-\sqrt{1999})(n+\sqrt{1999}) > 0$

$n \in (-\sqrt{1999}, 0) \cup (\sqrt{1999}, \infty)$

pt. after which n^3 dominates

$\leq \sqrt{1999}$	$\sqrt{1999} <$
n dominates	n^3 dominates



$\therefore O(n^3)$

Q) L → linear — i/p size $n = 10$ — 20 units of time

Q → Quadratic — 5 units of time

Approximate change in running time — Double the i/p ≈ ? ?

Sol:

L → $t \propto n$

$2t \propto 2n$

$\therefore$ 40 units of time

Q → $t \propto n^2$

$t' \propto (2n)^2$

$\Rightarrow t' \propto 4n^2$

$\frac{t'}{t} = \frac{4}{1}$

$t' = 20$ units of time

L → 40 units of time

Q → 20 units of time

(ii) n 10 times

L → 200 units of time

Q → $(10^2)\,5 = 500$ units of time.

Scanned with CamScanner

29.07.2022
Friday

Date: / / Page no:

	I/P size n	2n	
L (Linear)	20 units of time	40	200
Q (Quadratic)	5	20	500

At first L ~~is not~~ takes more time than Q due to more primitive operation

RECURSION :

↳ reduce the lines of code.

~~(i) Recursion call~~

(ii) Base cond^n

(iii) From 1st fn call to another value of n should be updated.

↳ If No Base cond^n then error = " Memory Overflow "

Recursion

Direct

```
xyz() {
  :
  xyz()
  :
}
```

Indirect

```
xyz() {
  :
  abc()
  :
}
```

```
abc() {
  :
  xyz()
  :
}
```

Recursion

Tail

When recursion call is the last line

```
f(n) {
  :
  f(n-1)
```

Non-Tail

else

```
f(n) {
  :
  f(n-1)
  :
}
```

```
f(n) {
  :
  return(n+f(n-1))
}
```

Scanned with CamScanner

Date: / / Page no:

Every recursive problem can be solved through iteration way.

```
fact(n) {
    if (n<=1) return 1;
    else {
        return n* fact(n-1);
    }
}
```

```
fact(n){
    int f, int i;
    for(i=1; i<=n; i++)
        f= f*i;
    print (f);
}
```

recursive way

fact(5)

n: 5

$5 \le 1$

$5 * fact(4)$ → 24

n: 4

$4 \le 1$

$4 * fact(3)$ → 6

n: 3

$3 \le 1$

$3 * fact(2)$ → 2

n: 2

$2 \le 1$

$2 * fact(1)$ → 1

n: 1

$1 \le 1$

return 1;

120

iteration way

n	f	i
5	1 → 2 → 6 → 24 → 120	1 → 2 → 3 → 4 → 5

1 int = 4 bytes

∴ 12 bytes ∴ memory

Recursive fn when called use Stack DS

For Recursive fn

① Time complexity = No. of fn calls * No. of operation in each fn call

② Space complexity = No. of fn calls * Space required in each fn calls.

③ Recursive calls takes more space than iterative way

Date: / / Page no:

$Fact(n) = n * Fact(n-1)$

Recurrence Relation.

~~$f(n) =$~~

Time Complexity

① Big $O(n)$ → worst case TC or Upper Bound TC

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

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

$c > 0,\ n \geq n_0$

$f(n) = n+10$

$g(n) = n$

$n+10 \leq c\, g(n)$

$c = 2$

$n+10 \leq cn$

$c = 2$

$O(2n)$ $\begin{cases} c = 2 \\ n \geq 10 \end{cases}$

False	$n=1$	$11 \leq 2$
	$n=2$	$12 \leq 4$
	$n=3$	$13 \leq 06$
	$n=4$	$14 \leq 8$
	$n=5$	$15 \leq 10$
	$n=6$	$16 \leq 12$
	$n=7$	$17 \leq 14$
	$n=8$	$18 \leq 16$
	$n=9$	$19 \leq 18$
True	$n=10$	$20 \leq 20$
	$n=11$	$21 \leq 22$
	$n=12$	$22 \leq 24$

② $F(n) = \Omega(g(n))$ → Best Case TC or Lower Bound TC

$f(n) \geq c\, g(n)$

Sample

$c = 1/2$

$n \geq 0$

③ ~~$\Theta(n) =$~~ $F(n) = \Theta(g(n))$ → Theta TC

$f(n) \geq c_2\, g(n)$

$f(n) \leq c_1\, g(n)$

$f(n) = n+10$

$g(n) = n$

Scanned with CamScanner

Base-2 & Base-6,
TC also, ~~on~~ Base dependsin

Date: / / Page no:

(i) $f(n) \le c_1 g(n)$

$n+10 = n+10 \Rightarrow n+10 \ge n+10n$

$\Rightarrow n+10 \ge 11n$

$\Rightarrow f(n) \ge c_2 g(n)$

$\boxed{c_1 = 11}$

(ii) $f(n) \ge c_2 g(n)$

$\Rightarrow n+10 \ge n$

$n+10 \ge n$

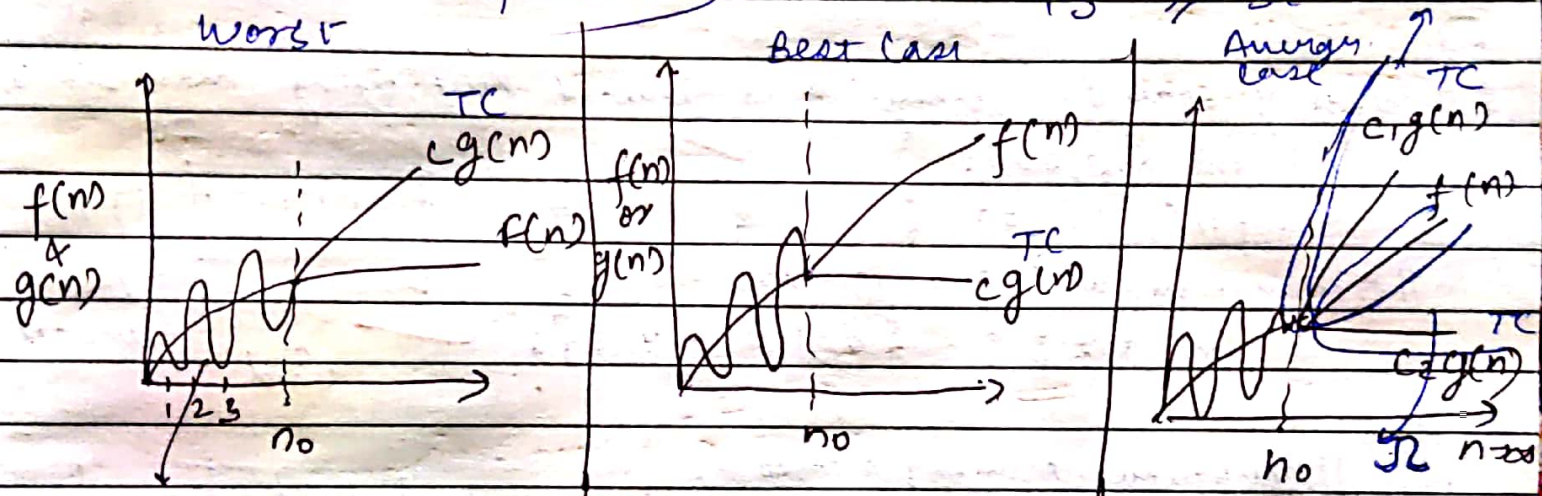
$\boxed{c_2 = 1}$

~~10 ≥ c~~

$11 \ge c$

$12 \ge 2c$

$13 \ge 3c$



any one can be greater	O	Ω	Θ
	$f(n) \le cg(n)$	$f(n) \ge cg(n)$	$f(n) \le c_1 g(n)$
	$c>0$	Greatest lower Bound (at least)	$f(n) \ge c_2 g(n)$
	$n \ge n_0$		Exact time
	$n_0 > 0$	$f(n) = \Omega g(n)$	$c_1 g(n) \le f(n) \le c_2 g(n)$
least upperbound (at most)		$f(n) \ge cg(n)$	$n^2 \le 2n^2+n \le 3n^2$

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

$f(n) \le cg(n^2)$

$2n^2+n \le 3n^2$

$n \le n^2$

$2n^2+n \ge c_1 n^2$

$n^2 \ge n, 1$

$c=2$

$2n^2+n \ge 2n^2$

$n \ge 0$

little O → O removed

little Ω

Scanned with CamScanner

03.08.2022 Wednesday

Date: / / Page no:

ARRAYS

```
#include <stdio.h.>
void main(){
    int arr1[10];
```

① //declarations

40 bytes

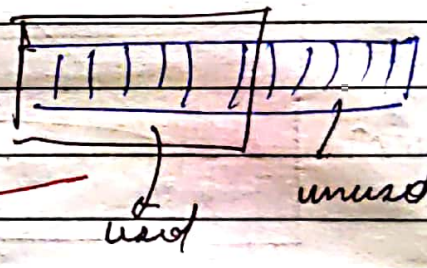
```
int arr2[ ], n; // X Not allowed
```

// Compile time error.

②

// Wastage of space; Only 10 is required but using max. blocks

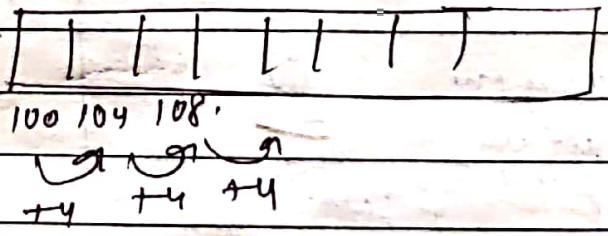
```
int arr3[20];
for(1 to 10) { }; //
```



// Declaration and Initialization

```
int arr[ ] = {0, 1, 2, 3, 4, 5};
```

③ // contiguous memory Allocation.



}.

// Sequential Search. - O(n)

Q) search an element in an array

Sol:

```
#
{

for(int i=0; i<10; i++){
    if(arr[i]== target)
        { cout << target <<"at location" << i << endl;
        break;}
}
```

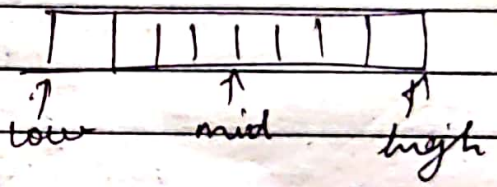
Scanned with CamScanner

Q) int i=2
if (i != 5){
break;
}

Sol: Error.
break can be used only in "Loops" and "Switch"

Binary Search. - O(log n) Sorted array is required

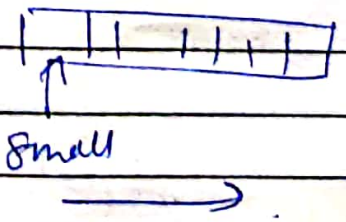
```
while (low < high) {
    mid = (high+low)/2;
    if (arr[i] == item)
        pos = mid;
    else {
        if (arr[mid] < item)
            low = mid+1;
        else
            high = mid-1;
    }
}
```



2^1
2^2

Q) Write the smallest element in an array.

Sol:
```
for (int i=0; i<n; i++)
{   if(arr[i] <= small)
        small = arr[i];
}
printf("smallest element is %d", small);
}
```



Scanned with CamScanner

$O(n^2)$

Date: / / Page no:

// Selection Sort

Search for smallest element and then swap it with the first element

Then repeat process in remaining array.

```
for (i=0; i<10; i++)
{   flag = 0;
    small = arr[i];
    for (j=i+1; j<10; j++)
    {  if (arr[j] < small)
          { flag = 1;
            small = arr[j];
            pos = j
          }
    }
    if (flag != 0) {
        temp = arr[i];
        arr[i] = small;
        arr[pos] = temp;
    } }.
```

// indentation // use

Worst Case

arr is in descending order

// Bubble Sort

| 8 | 6 | 5 | 1 | 3 | 7 | 10 | 2 |

6 8 5 1 3 7 10 2

6 5 8 1 3 7 10 2

6 5 1 8 3 7 10 2

6 5 1 3 8 7 10 2

6 5 1 3 7 8 10 2

repeat

6 5 1 3 7 8 10 2

| 6 | 5 | 1 | 3 | 7 | 8 | 2 | 10 |

Scanned with CamScanner

Date: / / Page no:

```
for(int i=0; i<n; i++) {
    for (int j= 0; j<n-1; j++) {
        if (arr[j] > arr[j+1]) {
            temp = arr[j];
            arr[j] = arr[j+1]
            arr[j+1] = temp;
        }
    }
}
```

// Insertion Sort.

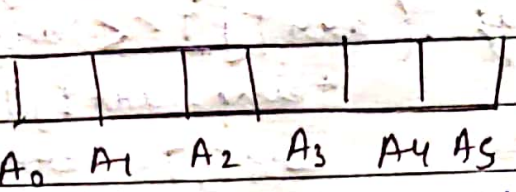
Scanned with CamScanner

04.08.2022
Thursday

Date: / / Page no:

Arrays: set of Homogenious stored in sequence memory location.
Each element is indexed

A:



for each index there is a value associated with.

1. Domain
2. Function → Operation
3. Axioms

Structure Array (value, index)

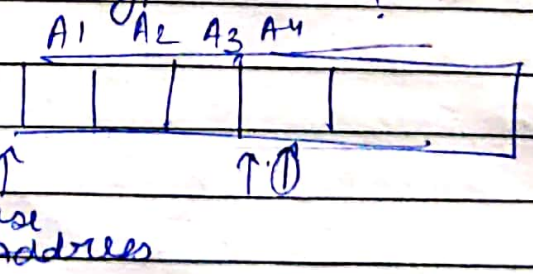
functions
- declare create() → array
- ~~Retrieve same data~~ Retrieve (Array, index) → value
- Store (Array, index, value) → (new) Array

Axioms rules
- for all A ∈ Arrays i, j ∈ index, x ∈ value let
  - Retrieve (~~Array~~ Create, i) := Error
  - Retrieve (Store (Array, index, value), j) := if Equal (i, j) then x else Retrieve (A, j)

end

Q) Why array has same data type values?

Sol: Random Access of value, Direct indexing mapping



$A_4 = A + (3 * \text{size of (data type)})$

(n-1)

Scanned with CamScanner

Date: / / Page no:

Minimum Addressable Unit

Word — 8 bits, 16 bits

Mapping is done through this — Physical Space.

Indexing formula — logical formula relates to Memory formula

: direct access.

$$\boxed{\text{Indexing formula} = B + (i - L) * W}$$

B → Base address

i → Required index

L → Lowest index of language

W → word size

$A_L \quad A_{L+1} \quad \ldots \quad A_U$

W	W	W	W	W	.

↑ B.

arr = [A_L, ... A_U]

A_L → Lower index

A_U → upper index

Q) Starting address = 459 Array [5 ... 85]

word size = 2 (2 bytes)

Sol: (i) No. of elements in array = 85 − 5 + 1 = 81

U − L + 1

(ii) memory of array = 2 * 81 = 162 bytes

(iii) A_{70} = 459 + (70 − 5) * 2 → B + (i − L) * W

130 $\quad \times \frac{65}{2}$ $\quad$ 130

⇒ A_{70} = 589

Multidimensional array

Matrix → continuously stored

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ & \vdots & & \\ A_{31} & A_{32} & A_{33} & A_{34} \end{bmatrix}$$

Scanned with CamScanner

Date: / / Page no:

Stored as such.

Row-major order :- | First Row | 2nd row | ...

Column major order :

| Colum-1 | Colum-2 | .... |

$\longleftarrow n \longrightarrow$

Matrix A_{mn} $\uparrow m \downarrow$

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ \vdots & & & \\ A_{31} & A_{32} & A_{33} & A_{34} \end{bmatrix}_{m \times n}$$

Indexing Formula. (IF)

① $(IF)_{R\text{-}M\ order} = M + [(i-1)*n + j - 1]*W$

② $(IF)_{C\text{-}M\ order} = M + [(j-1)*n + i - 1]*W$

3-D Array

A_{mnk}

slowest + row fastest

little rowers

A_{xyz}

$$(IF)_{RM} = M + \underbrace{(z-1)*m*n}_{\text{extra term}} + \underbrace{(x-1)*n + (y-1)}_{\text{2D array}}$$

for lower index = L] $z - L$ $x - L$

Scanned with CamScanner

05.08.2022
Friday

Date: / / Page no:

Indexing formula : Mapping of logical formula to physical address.

Q) Storing polynomial in an array.

Sol: M.1:

3	2	5	1	1	0

base power b p b p

$= 3x^2 + 5x + 1x^0$

Extra storage

M.2: (Method : Use of index)

a [degree + 1]

2	1	3	0	1	1

→ elements as value.

$2x^5 + x^4 + 3x^3 + x + 1$

5 4 3 2 1 0 → power as index

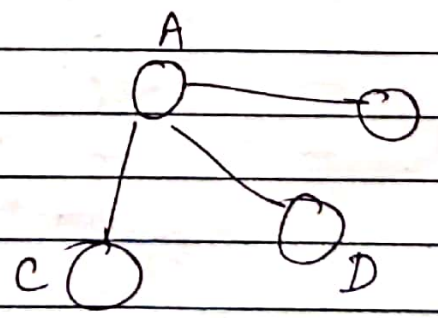
mapped

Sparse Matrix

Majority of the entries of the Matrix = 0

'dense'

	A	B	C	D
A	1	1	1	1
B	1	1	0	0
C	1	0	1	0
D	1	0	0	1



Graph.

↳ various calculation can be performed.

less than 1/3 values = Non zero

Sparse matrix but not clear definition

Scanned with CamScanner

Date: / / Page no:

* Advantage of matrix representation
→ Getting / Accessing element directly.

Sparse Matrix

Symetric / Regular ← → Non-Symetric / Irregular SM

(i) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

diagonal element = 1

(ii) lower triangular matrix

$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 5 & 2 \end{bmatrix}$

(iii) upper Δ Matrix

Q) Indexing formula for the Lower, upper Δ matrix

Scanned with CamScanner

Sparse Matrix

Logical view

$$\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 15 & 0 & 0 & 22 & 0 & -15 \\ 1 & 0 & 11 & 3 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & -6 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 91 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 28 & 0 & 0 & 0 \end{array}$$

Storage → Array of triplets. (row no, col no, value) of Non-zero elements.

Transposing

If $m \times n$ TC = $O(mn)$

	Row	colum	value
A	i	j	
A[0]	6	6	8
A[1]	5	2	28
A[2]			

Size = No. of non-zero values + 1

→ do not place randomly other TC increases

Ordering

	Row	Col	Value
A[0]	6	6	8
A[1]	0	3	22
A[2]	0	5	-15
A[3]	1	1	11
A[4]	1	5	3
A[5]	2	1	18
A[6]	3	4	-1
A[7]	3	5	-63

→ No. of Non-zero elements

Row Major order for both matrices.

Not to follow but can be made.

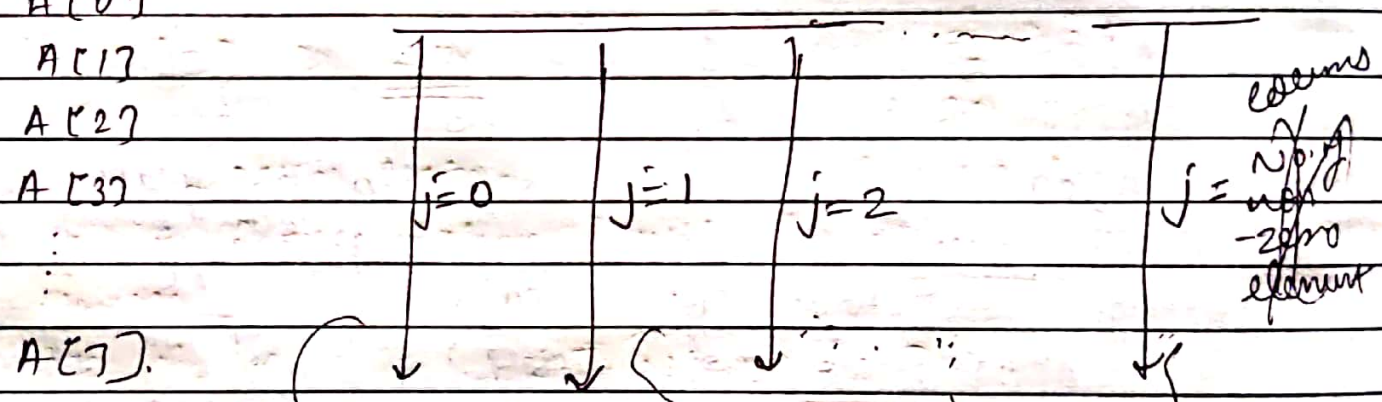
Physical View

Scanned with CamScanner

Transpose

swap(i,j) → we get ans.
but our data structure is disturbed
ordering is disturbed

Method
(i) Scan j for 0 put in ~~Matrix~~ res array
(ii) Scan j = 1 put in res-array
⋮
(iii) Scan j = col-No. put in res-array
{a[0]j}



$\boxed{TC = \text{No. of col} \times \text{No. of Non-zero elements}}$

	Row	Col	value
B[0]	0	1	15
B[1]	0	4	91
B[2]	1	1	11
B[3]	1	3	6
B[4]	1	4	7
B[5]	2	1	16
B[6]	3	2	-8
B[7]	5	0	-15

Scanned with CamScanner

Arbit Sahni

Date: / / Page no:

Same ~~struct~~ ordering even after.

$(TC)_{Sparse}$ is efficient

else it would be even worse

$(TC)_{Normal\ Transpose}$ = No. of ~~elements~~ columns × No. of elements

Fast Transpose

	0	1	2	3	4	5
Row terms =	2	1	2	2	0	1
Start pos =	1	3	4	6	8	8

Size [No. of col]

check for no. of ~~row~~ col no = j' say = 0

→ next starting pos for R-1, R-2

while filling increment & put corresponding swapped value in res ~~array~~ matrix till next start pos.

TC = (No. of columns + No. of elements)

while we do for SC two arrays

Scanned with CamScanner

Date: / / Page no:

Representing Sparse Matrix.

Row	Col	Value
n_r	n_c	n_{nz} v

} Non zero elements

① Triplet Representation

② CSR or Yale (Compressed Sparse Row,

$\frac{2}{3}$ (Size of matrix) = 0

Q) $m \times n = 5 \times 6$ matrix

N (Non zero) = 6 Is is a Sparse Matrix/not?

If yes, space ~~sto~~ occupied by Sparse Matrix.

Sol: $m \times n = 30$

$N(NZ) = 6$

$N(Z) = 30 - 6 = 24$

$\therefore \boxed{N(Z) > 20}$

For sparse Matrix

$N(Z) = \frac{2}{3} \times N$ (elements)

$= \frac{2}{3} \times 30$

$= 20$

$\therefore$ Sparse Matrix.

$\Rightarrow$ Triplet Representation will be efficient in terms of storage.

② Size of TR $= 3 * (N(NZ) + 1)$

$= 3 * (6+1) = 21$

Scanned with CamScanner

18.08.20
Thursday

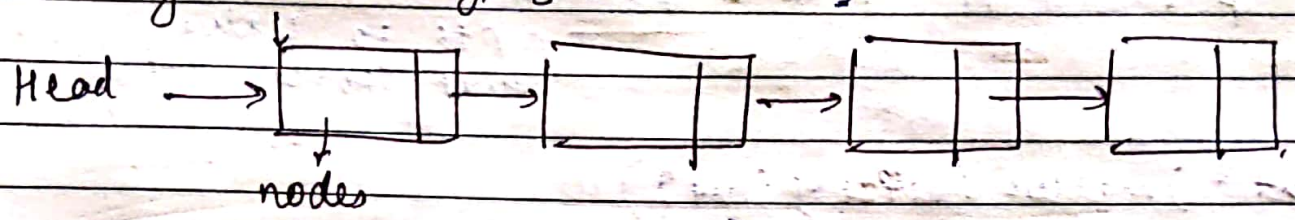
Date: / / Page no:

Linked List

Dynamic memory (size not known ~~early~~ before)

(In array, we need to give before hand)

Homogenous data types



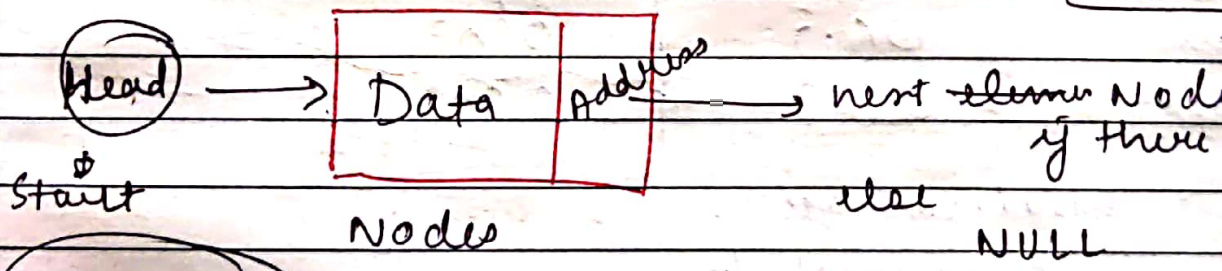
Not stored continuous

Head → Address pointing to first ~~elem~~ Node

(only this access pt)

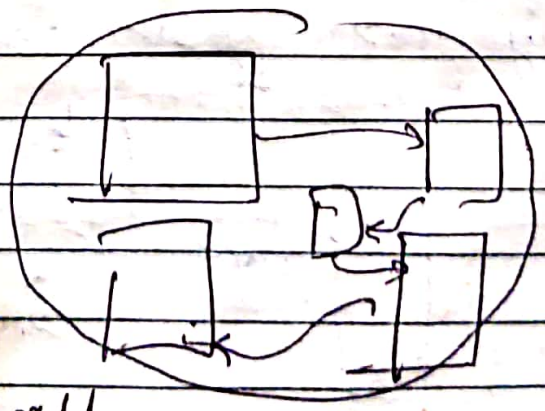
Advantages of array:
direct access
size defined in prog time or ~~at~~ ~~ee~~ ~~ly~~

Disadvantages of array
Insertion & deletion



Runtime Heap of memory

Memory manager allocates memory dynamically



Scanned with CamScanner

Date: / / Page no:

There are no names for dynamic variables.
~~Ptr~~ with the ptr. we do that.

~~p = (int *~~

~~p~~ int * p ; // p is a pointer
↳ pts to value that stores "int"

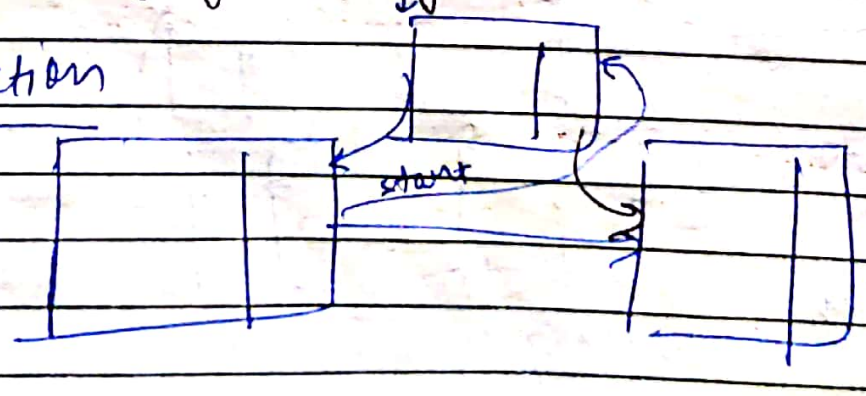
p = (int *) malloc (sizeof (int))

// memory allocates of size int
and points that ~~to~~ memory chunks
~~to~~ start address to p

	linked list	Arrays
Traverse	O(n)	O(1)
Deletion	O(1)	O(n)
Insertion	O(1)	O(n)

→ Disadvantage
Traversal time
Only good if are on that element

Insertion

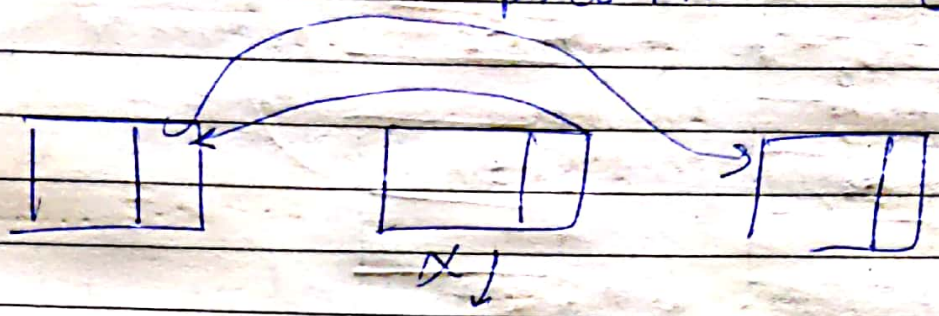


Scanned with CamScanner

Date: / / Page no:

Deletion

two ptrs require
prev ptr curr ptr

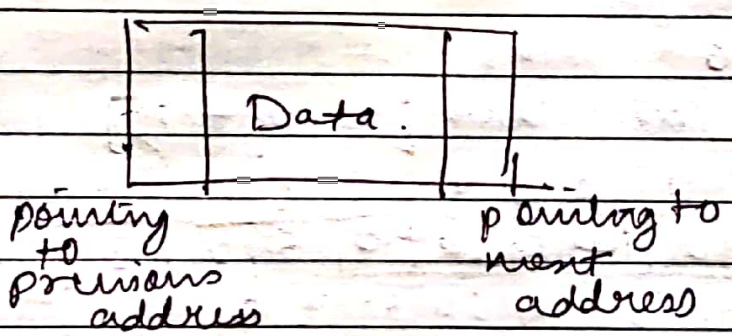


free this space

We can't go back in linked list (since contains address of next element)

∴ Singly linked lists

Doubly linked list



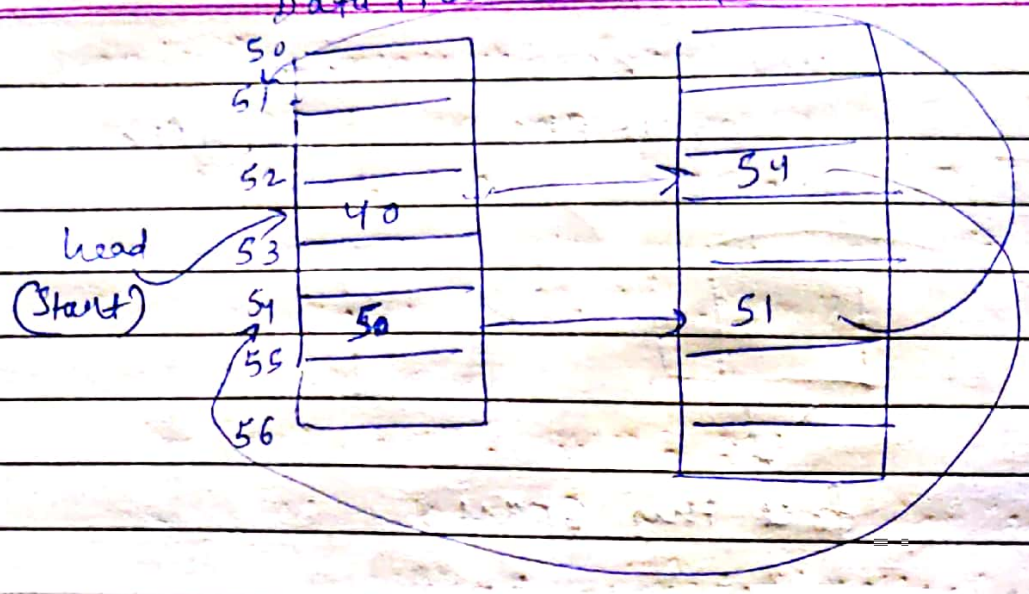
Circular linked list

SLL → ∅ last Address pts to head.

In old books Static Representation of linked list (Info based)

In old languages "FORTRAN"

Scanned with CamScanner



Application of LL:

① Size of data is not known before hand.

② Polynomial (in array).

① $10x^{10} + 4x^8 + 6x^7 + 5x^4 + 3x^2 + 2x + 1$

degree = 10

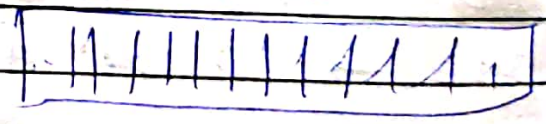
int arr [degree + 1]

0	1	2	3	4	5	6	7	8	9	10
10	0	4	6	0	0	5	0	3	2	1
10	9	8	7	6	5	4	3	2	1	0

arr(size − 1).

② $3x^{1000} + 4x^{50} + 6$

(Sparse polynomial)



wastage of space

1000	3
50	4
0	6

int arr [terms * 2]

$3x^{1000} + 4x^{50} + 6$

$4x^{10} + 5x^{2} + 3x + 2$

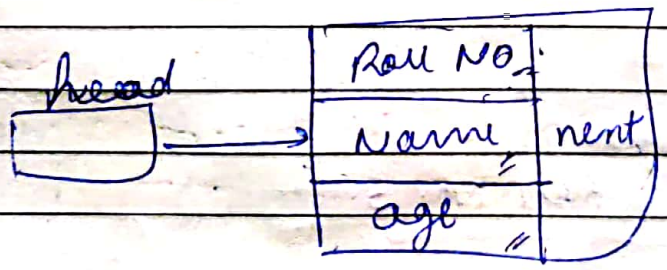
Multiplication of these by arrays is difficult task.

∴ LL is benefiicial

Q) Create a LL student data

Sol:

```
struct Stud {                    // defines structure so that memory manager allocates memory chunk.
    int Roll_No ;
    char name [20];
    int age ;
    struct stud * next ;
}

typedef struct stud node ;       // node = struct stud
node * head ;                    // head ptr pointing of type to Node
```




```
head = (node *) malloc ( sizeof (node));
```

// ~~adding new node~~ first node

Scanned with CamScanner

Date: / / Page no:

```
node * CreateList() {
    int k, n;
    node *p, *head;   // p = curr ptr "  this changes
    printf("How many elements to be inserted");
    scanf("%d", &n);

    for (int k=0; k<n; k++)
    {   if (k==0) {   // edge case for head
            head = (node*)malloc(sizeof(node));
            p = head
        }
        else {
        p->next = (node*)malloc(sizeof(node));
        p = p->next;
        }

    Copy the contents to the newly inserted node

    p->next = NULL;
    return head;
}
```

Insert a Node at Roll No. 5

```
~~for(int i=0; i~~ int ct
while(p ~~->next~~ != NULL) {
    ct++;
    p = p->next;
}
```

Scanned with CamScanner

Date: / / Page no:

```
node * target = head;
while ( p -> next != NULL && p -> RollNo != 5 ) {
        p = p -> next;
        target = p;
        p = p -> next;
}

// New Node
node * newN = (node*) malloc(sizeof(node));

newN -> next = target -> next;
newN -> data = Data.
target -> next = newN;
```

Pseudo Code

```
new = Getnode (Node)
if new = NULL then
print "Memory not sufficient"
exit
else
    ptr = head
    while (ptr -> Roll != key) and (ptr -> next != NULL)
    do
        p = p -> link
                next
    endwhile
```

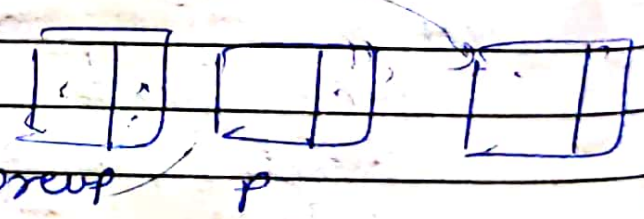
I

Scanned with CamScanner

```
if p -> ~~link~~ next = NULL then
    print (key now available)
    exit
else
    new -> link = p -> link
    new -> data = data
    p -> link = new
endif
```

Deleting a Node with a given Roll No

```
(I)                node * prevp
while (p->link != key && p->link != NULL)
    do prevp = p
       p = p -> link;
end while

if p -> link = NULL then
    print (key not available);
    exit
else
    prevp -> next = p -> next
endif
```



Creating a Doubly linked list

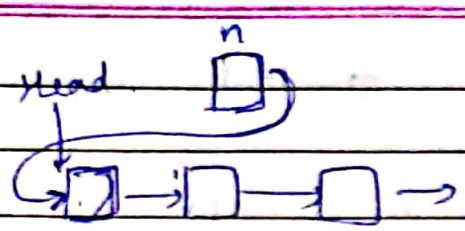
Scanned with CamScanner

Date: / / Page no.

C++

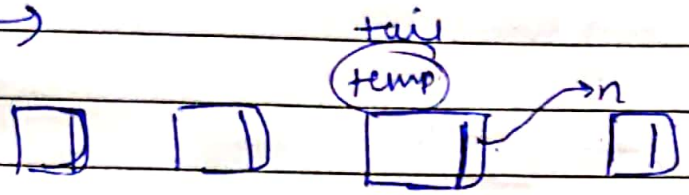
① Insert at Tail

② Insert at Head



n->next = head

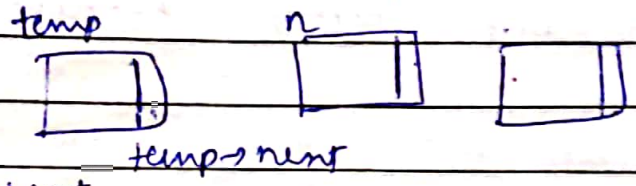
head = n



temp->next = NULL

tail->next =

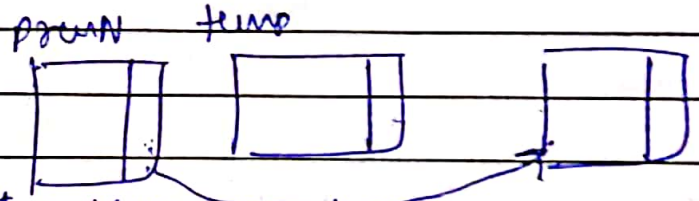
③ Insert a node at specific pos.



n->next = temp->next

temp->next = n

④ Delete a node

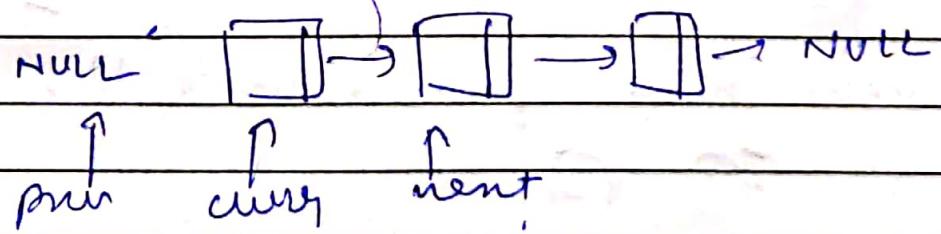


prevN->next = temp->next

temp->next = NULL.

⑤ Reverse the LL; ~~3 pt~~

// Iterative-Way 3 ptr



⑥ merge 2 sorted linked list

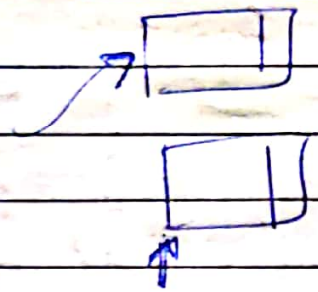
~~External~~ Extra space

No space

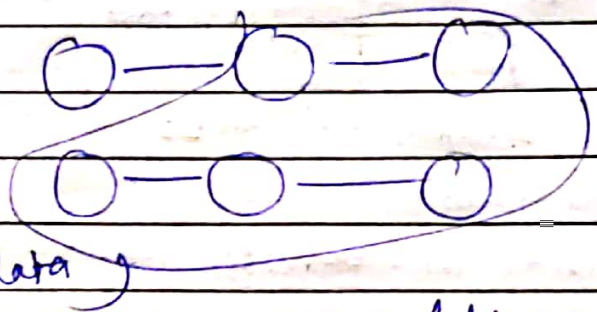
(structure should not be disturbed)

Scanned with CamScanner

Date: / / Page no:

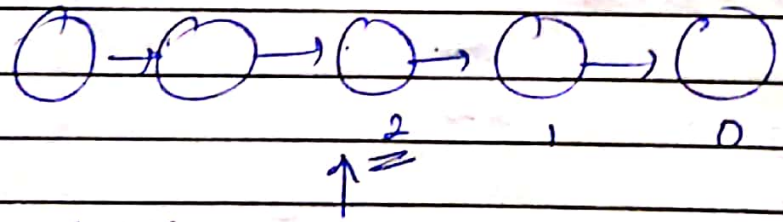


node*
merge (l1, l2)



if (l1→data <= l2→data)
{ l1→next = merge (l1→next, l2) return l1
else
l2→next = merge (l2→next, l1) return l2

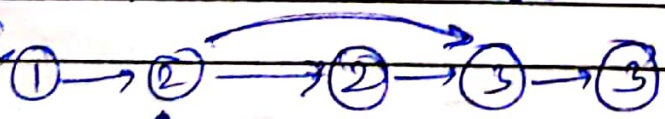
// Get ith Node from tail



n = 5
5 - 2 = 3

// Delete duplicate value nodes from a sorted ll;

Wanted



prevP=

if (prevP → value == curr P → value)
curr P = curr P → next
else
{ prevP = curr P
→ next
}

if (sing

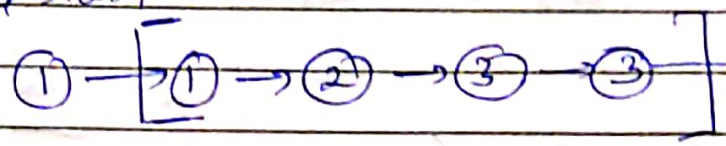
Scanned with CamScanner

~~temp~~

temp->data == next->data.

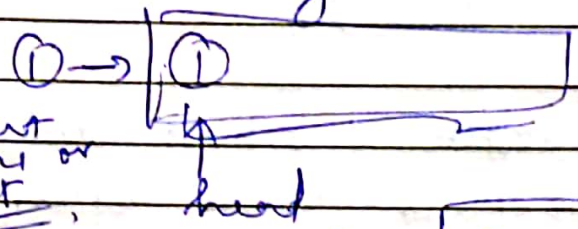
temp->next = next->next

Way.2 Recursion



losing it through rec.

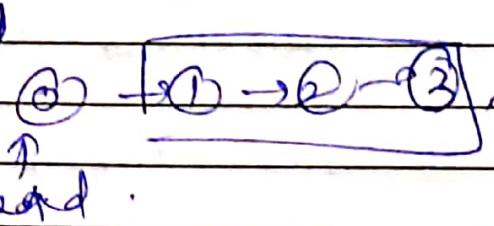
Chek.



head

if element same or not.

else.



head.

if (temp->data = rec

// Cycle detection in linked list

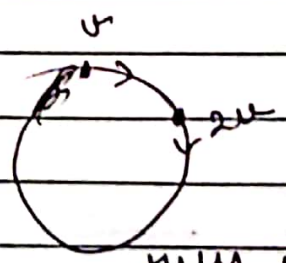
if (!cycle) then we will get NULL ptr.

Way.1 Hashset

Extra space: O(n)

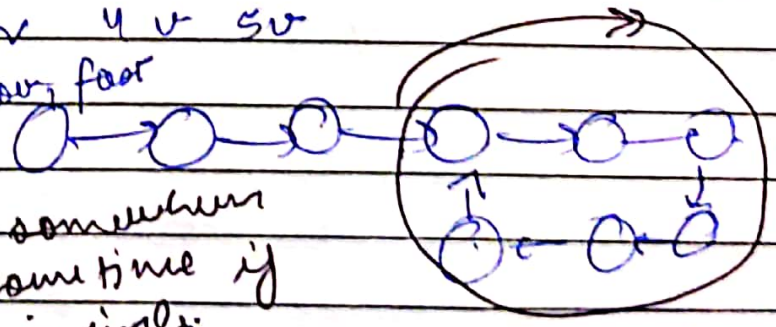
Way.2 Fast ptr Slow ptr.

2v 2v



3v 4v 5v

slow fast



they will somewhere at some time if in circle.

Scanned with CamScanner

Date: / / Page no:

// Intersection of Linked List

Way-1 $O(m \times n)$

Way-2 unordered_set <~~int~~ node*> u;

$O(m+n)$ → put

$O(n)$

while (b != NULL){

$O(1)$

~~if~~ (~~b->data~~ == u.~~search~~(b->data)){

Way-3

① find length of both strings

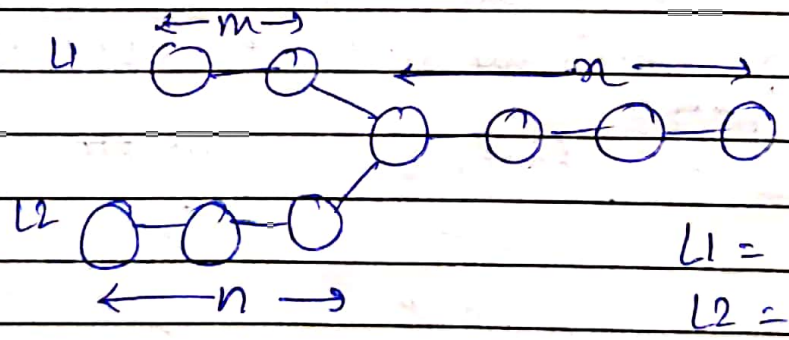
② make their length equal by updating bigger length //

③ Traverse both simultaneously

if (a == b) return data.

else Not

Way-4



$L1 = (m+x) + (n+x)$

$L2 = (n+x) + (m+x)$

Scanned with CamScanner

Date: / / Page no:

Doubly linked list

// deletion easy but size is more { than singly //

NULL ← (1) ⇄ (2) ⇄ (3) → NULL

```
class Node {
    int data;
    Node* next;
    Node* prev;
}
```

STL

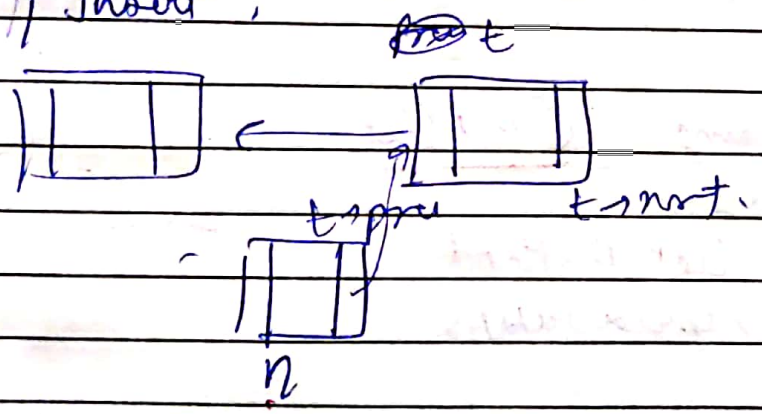
forward_list <int> fl // singly linked list

→ erase_after (prev element)

list <int> l // Doubly linked list

→ erase ( )

// Insert,



n → next = t

more use

Doubly LL > singly LL

Uses of linked list :-

1. Implimentation of stacks & queues
2. Image Gallery } Doubly linked list — Can go to next img & prev img
3. Music Gallery
4. Browsers — prev page & next page
5. Hashmaps & Hashsets (Collision time chaing technique)
6. LRU (Least Recently used) Cache : DLL

Scanned with CamScanner

25-08-2020
Thursday

Date: / / Page no:

```
struct node *p, *q, *r;
p = q = r = head;
p = p -> next -> next;
q = q -> next
r -> next = NULL
q -> next = r


while (p ≠ null) {
        r = q;    q = p;
        p = p -> next;
        q -> next = r
    }
    *head = q
}
```

Sparse data is stored using linked list.

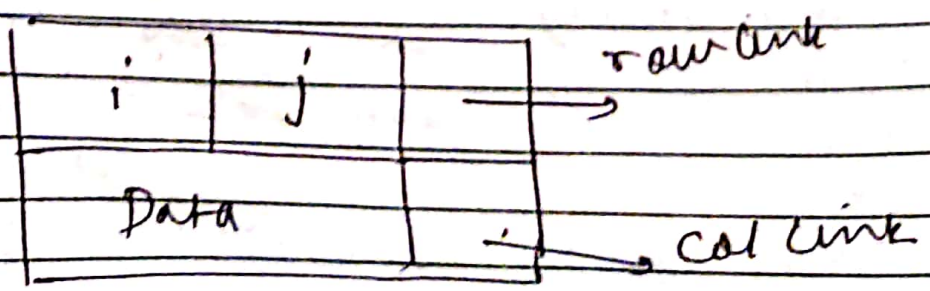
Sparse Matrix : Linked list Method

over array representation

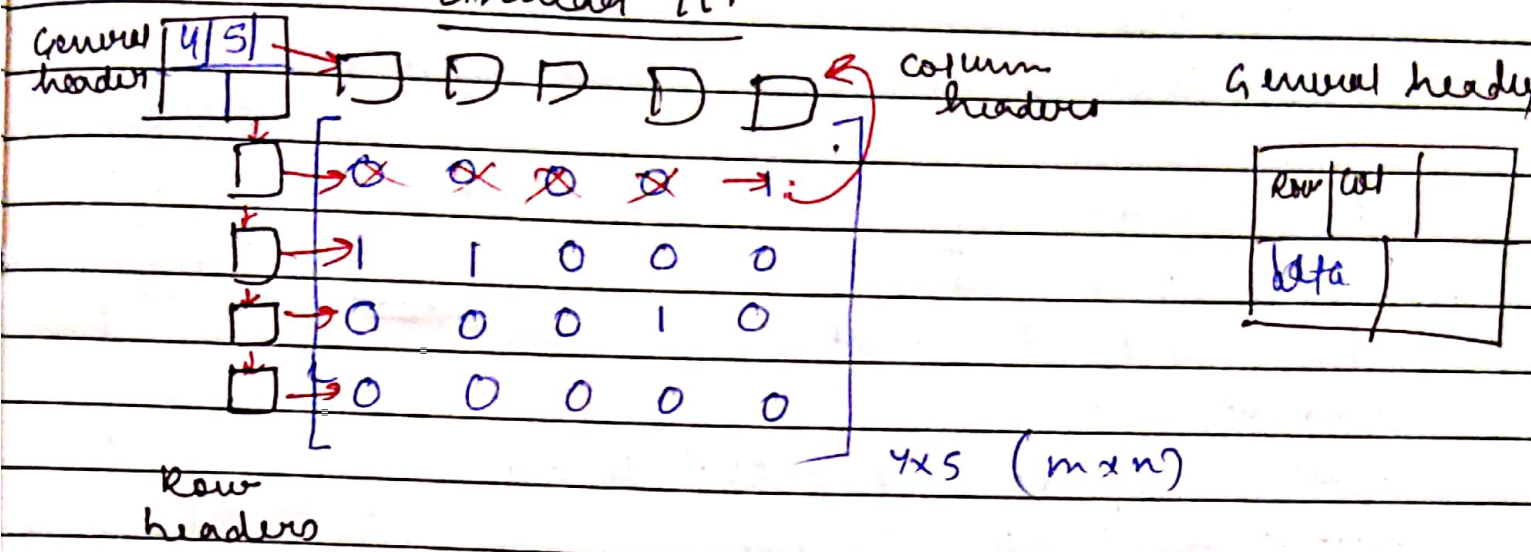
Q) Why preferred?

Sol: Deletion & updation of taken O(1) time + Traversing time.
(if Matrix changes)

Linked list Node Structure →



These are circular ll.



Program

```
// ① Get the size of sparse matrix
// ② Get the node of header.

Read m, n
Header = Get node (NODE)          // Structure same
// ③ Error checking in some languages
  if (Header == NULL) then
    Print "Memory not Available";
    exit if
else                              // ④ Initialise header node
    Header -> i = m
    Header -> j = n
// Structure should follow rules when its empty.
  // Circular ll
    Header -> rowlink = Header
    Header -> Collink = Header
    Header -> data = NULL
```

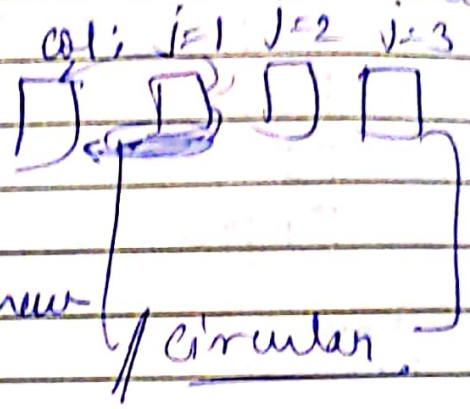
Scanned with CamScanner

Date: / / Page no: No. of col.

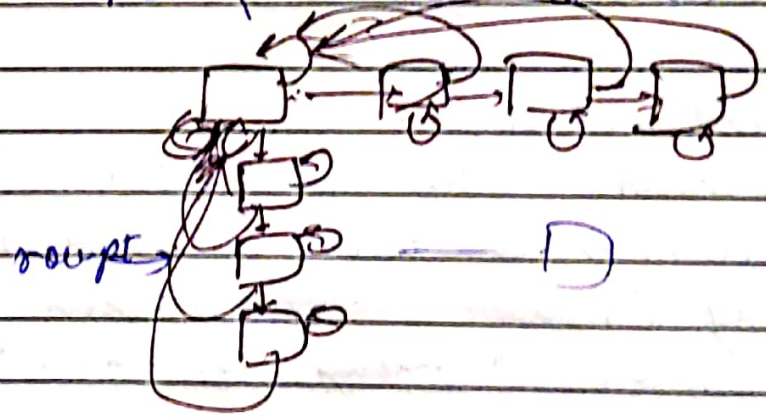
// (5) Get column headers & initialise them (n col headers)
// (6) Get Row headers & initialise them (m Row headers)
No. of Rows

Ptr = Header

for col=1 to n // (5)
  new = Get node (NODE)
  new → i = 0;
  new → j = col; , ptr → rowlink = new
  new → ~~row col~~ link = header
  ~~new → col link =~~
  new → collink = new ;



// (6) Same code replace {row → col & col → row}



// (7) Adding data (nodes) Insertion
Read(data, row, col)
row Header = Header → col link
col Header = Header → row Link // learn

// (8) Traversing to required field

Scanned with CamScanner

// Reaching row

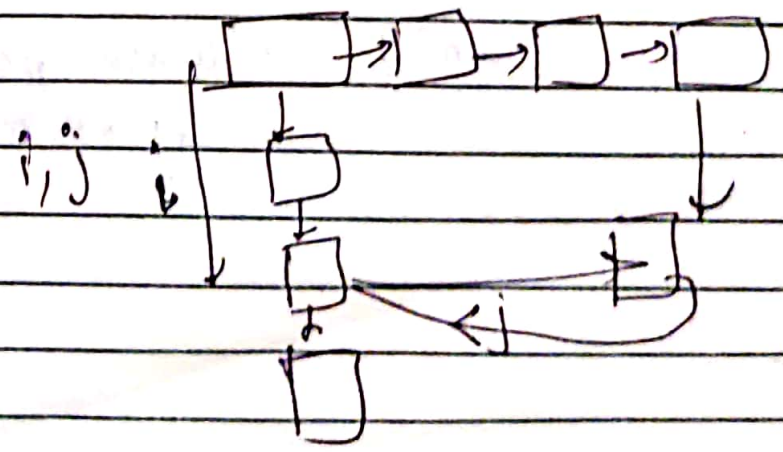
Date: / / Page no:

```
while (row header -> i < row)
    rowheader = rowheader -> colLink
end while
// Reaching column.
while (colheader -> i < ~~column~~ col)
    colheader = colheader -> rowlink
end while

rowpt = rowheader

while (rowpt -> j < col) do
    ptr1 = rowpt
    rowptr = rowptr -> Rowlink
    if (rowptr == rowheader) then
        break;
```

Search for col.

```
    Ptr1 -> rowlink = new
    ptr2 -> col link = new
    new -> Rowlink = rowptr
    ~~ptr2~~ new -> col link = colptr
    new -> Data = Key
end if
```

#
Sparse matrix is used extensively



Scanned with CamScanner

26.08.2022 Friday

Date: / / Page no:

Polynomials

Abstract Data Structure (ADT)

① Dense Polynomial

$4x^4 + 6x^3 + 10x^2 + 5x + 1$

4	6	10	5	1
4	3	2	1	0

→ size = 4+1

② Sparse Polynomial

1000	100	99	0
5	5	4	3

$5x^5 + 3x^4 + 2x^3 + 10x^2$

$10x^5 + 3x^2$

Two pointer approach

3 case

power = 0

$p1 > p2$

$p1 < p2$

$p1 = p2$

~~$10x^{1000} +$~~

$10x^{1000} + 5x^{100} + 4x^{99} + 3$

$10x^5 + 4x^2 + 3x + 1$

$100x^{1005} + 40x^{1002} + 30x^{1001} + 10x^{1000} + 50x^{105} + 20x^{102} + 15x^{101} + 5x^{100} + 40x^{104} + 16x^{101} + 12x^{100} + 4x^{99} + 30x^5 + 12x^2 + 9x + 3$

$\boxed{\text{Time Complexity} \rightarrow O(m+n)}$ when there is different powers

Exclusive set

$O(\max(m,n))$ → When powers of one is greater of other

Scanned with CamScanner

Memory Management

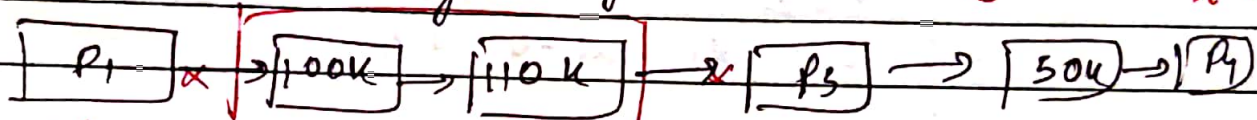
Task is going on

P1	100K	P3	40K	P4	10K	P6

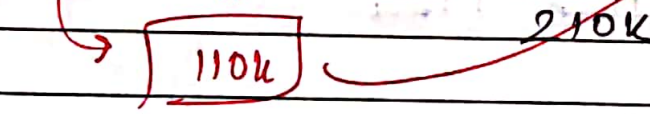
← free memory →

Memory Manager work :-

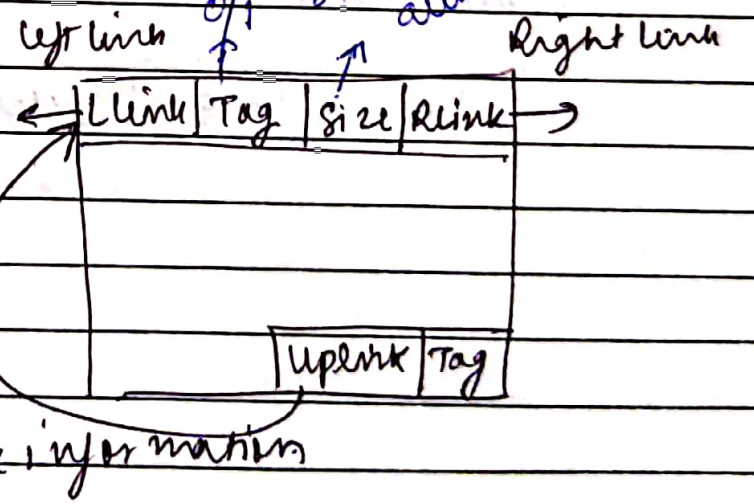
1. Give the free memory to new work
2. Keep the track of all free memory blocks. doubly
   ↳ in the form of linked list (circular ll)



3. can also merge different free memory fields
   like 100K + 110K
   210K.



Boundary Tag System

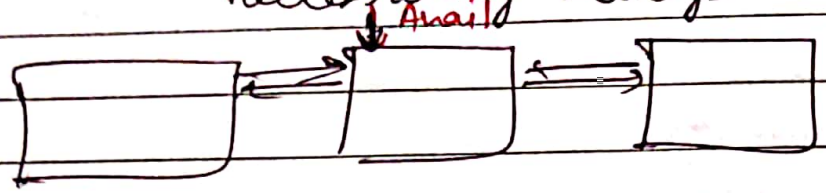


Tag
0 → free
1 → engaged / ~~no~~ allocated

Since it works on boundaries of 0 & 1
∴ Boundary Tag System.

Scanned with CamScanner

Avail pointer to point towards that node which is free. Not necessarily always on header




```
ptr = Avail
flag = 0
while( flag == 0 ) do
    if ( ptr -> size >= n ) then
        // ① allocate whole block
        or
        // ② fragment that block and make cond^n
        or
        // ③ merge two block
    if (ptr -> size < n) then
        // merge two blocks of memory
```

02.09.2022
Friday

Date: / / Page no:

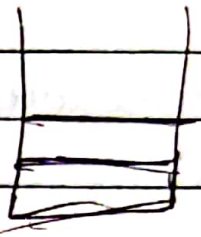
STACKS

Linear data structure — Add & Remove from one end

Storing linear list

(monitored) LIFO → last in first out — Pile of coins

FILO → first in last out.

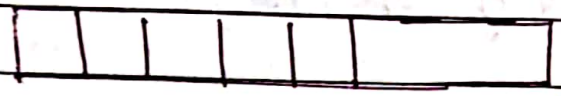


Restricting ~~entry~~ entry & removing of elements

Most important

Application :- Recursion (Error = Memory Overflow)

ARRAY



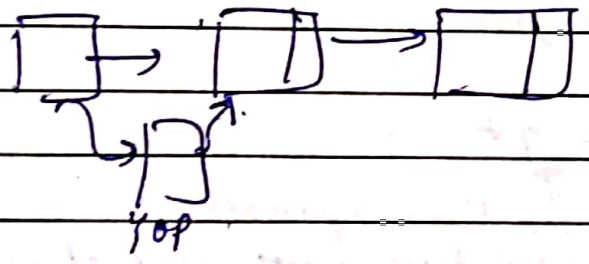
Top element → O(1) Time In end

O(N) Time In front

LINKED LIST

Benefit of linked list, No memory overflow.

Head ptr is used as Top



PUSH (item) → Inserting an element

POP (item) → Removing an element

In array, if stack is empty() → 0 index array −1

or 1 Starting from 1 ~~False~~ 0

POP

if (Top = −1)

else Top = Top − 1

bool isempty( )

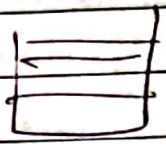
[check top]

push( ) pop( )

top( )

Scanned with CamScanner

Applications of Stacks :-



① Recursion

② Evaluation of Expression

A + B * C **D → Infix Notation
(Scan again & again to know priority of operators)

——— → Postfix Notation
(In a single go we can evaluate)

— Use of "Stack" —

A + B * C ABC* +
Infix Postfix

A $\boxed{B\ C\ *}$ +

Scan till first operator
evaluate prev. two with use of next operator

A T_1 + (B*C = T_1)
A + T_1 ⇒ A + B*C

Priorities = Unary, ( ), exponentiation, Divide = Multiply, [L to R]
parenthesis
Add, Subtract [L to R] left to Right.

Q) A + B * C/D + E**F
Convert this infix form to postfix form.

Scanned with CamScanner

Sol: ① Left operant, Right operand followed by its operator

A <op> B → A B <op>

Infix Postfix

NOTE:

Prefix == Polish Notation

Postfix == Reverse Polish → most compilers use this

Stacks are used -

(i) To change Infix → Postfix

(ii) To evaluate Postfix.

In Prefix / Postfix do not change the order of variables.

A + B * C

Scanning →

	Stack	O/P Postfix
①	—	A
②	+	A
③	+	AB
④	+ *	AB
⑤	+ *	ABC

Comparing with prev. operator

next op > prev op

end of i/p token

pop ( ) from stack and append it to end of o/p Postfix

Postfix → ABC+*

Scanned with CamScanner

Date: / / Page no:

A + B - C

	Stack	O/p Postfix
①	-	A
②	+	A
③	+	AB
④	-	AB +
⑤	-	AB+C
		end

Compare + & -

+ > - pop then

~~pop~~ append + and add -

postfix → AB+C-

Q) A + B / C * C / D + E * * F

Sol:

	Stack	O/p Postfix
	-	A
	+	A
	+	AB
* > +	+ *	AB
	+ *	ABC
/ > *	+ * /	ABC
	+ * /	ABCD
/ > +	~~+ * +~~	~~ABCD * /~~
append pop	~~+ * +~~	~~ABCD * /E~~
* * > +	~~+ * *~~	~~ABCD + * /E~~
	~~+ * *~~	~~ABCD + * /EF~~
		end

Postfix →

∴ ABCD+*/EF**

+ +	ABCD*/
+ +	ABCD*/E
+ + * *	ABCD*/EF
	end

ABCD/*EF**++

Comparing

(i) P(next) > P(prev) append in stack

(ii) P(next) ≤ P(top) Remove top from stack & append that top in postfix string add next on top

Evaluation of expression

Date: / / Page no:

Create an empty stack

for each token in the string {

if token is an operand.

Stack.push (token);

else if token is an operator {

let op denote the operation that ~~token~~ offers provide

Stack-pop (Right operand);

Stack_pop (left operand);

Result = left operand op Right operand

Stack-push (Result)

A + B * C / D + E ** F

		Stack	o/p Postfix
A		–	A
A B		+	A
A B C		+	A B
A B C *	(* > +)	+*	A B
A B C * D / + E ** +		+*	A B C
	ϕ	+*/	A B C
	(/ > *)	+*/	A B C D
	(+ < /)	+*	A B C D /
	(+ < *)	++	A B C D / *
		++	A B C D / * E
	(** > +)	++**	A B C D / * E
		++**	A B C D / * E F
			end

A B C D / * E F ** ++

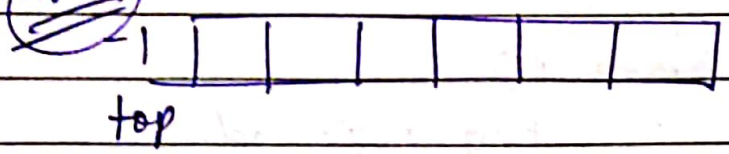
Stack operations

1. Push(x) → insert element
2. pop() → remove top element
3. top() → return top element (pointer)
4. empty() → whether stack is empty? True if empty.

TC of all → O(1)

5. full() → whether memory ends or not

arr

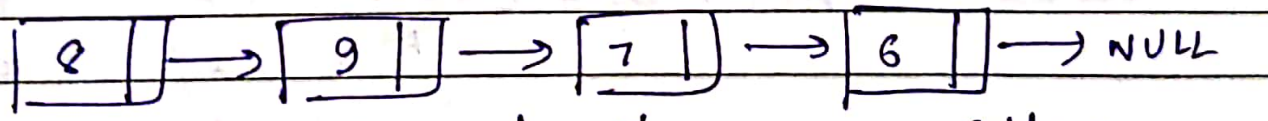


1. push → top++

Cond^n: If array ends then join one more array

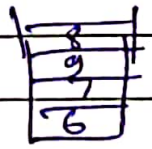
2. pop() top-- value remains but can't get
3. top() cout
4. empty() top == -1 True
   else false.

ll



Side 1

O(1) insertion
deletion



Side 2

O(n)

ptr = NULL

I. is used 1. Push()
2. Pop()

II. Stack is never be full → Only fails when Heap memory is exhausted. (dynamic memory)

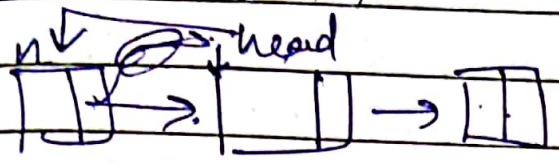
III. Head is now → top

IV. Stack empty Cond^n (top == NULL)

V. custom size of ll can be set (keep check of no. of elements)

Scanned with CamScanner

① Push ( )



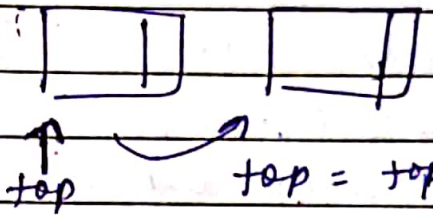
n→next = head→next

head = n

② ~~pop~~ empty( )

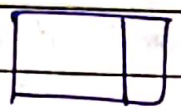
top == Null true

③ pop ( )



top = top→ next

④



top → data

Postfix Evaluation

① operands in stack until we get operator

② top → right operand

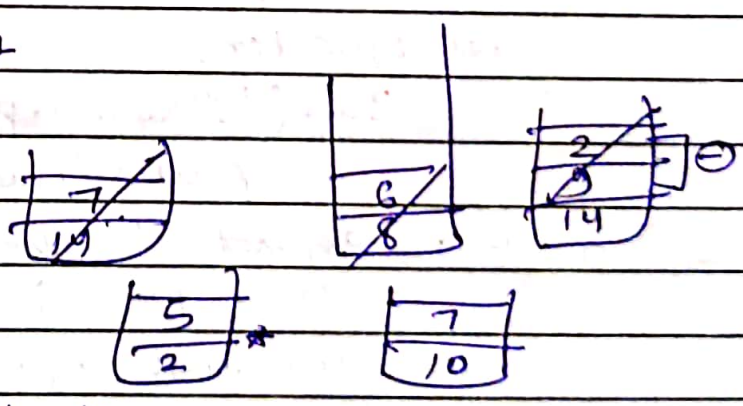
~~left~~ → top-- → left operand

pop & evaluate

& push ans

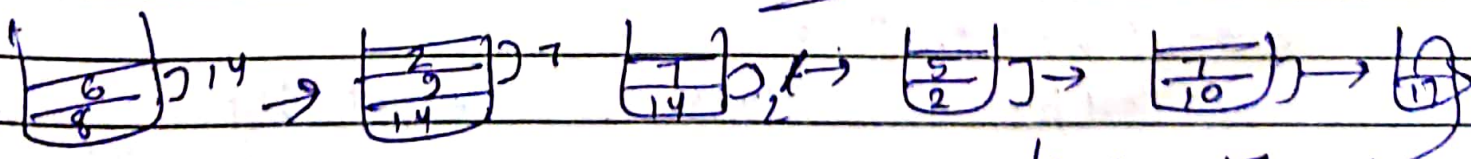
Q) 8 6 + 9 2 − / 5 * 7 +

Sol: 17



14 9 2 − 2 5 * 7 +

7 / 10 17



o/p = 17

03.09.2022
Saturday

Date: / / Page no:

o/p Postfix Pseudocode

(b + (c * d)/e

Stack	o/p post
-	b
+	b
+	c
+*	d

bcd * +

until

```
create an empty stack to hold
operands
for each token tok in i/p infix string{
    switch (tok) {
        case operand :
            append tok to right end of postfix;
            break;
        case '(' :
            Stack push (tok);
            break;
        case ')' ;
            while (stack.top () ! = '(') {
                append stack.top () to right end of postfix;
                Stack.pop ();  }
            break;
        case operator ;
            while (! Stack.empty () && stack.top() != '(' &&
                precendence (tok) <= precendence (stack.top())
                append stack.top () to right end of postfix;
                Stack. pop () ;  }
            Stack. push (operator);
            break;  } }
```

~~Q) a - (b + c~~

Q) a - (b + c * d) / e

Sol:

o/p Postfix	Stack	Input infix
a	-	a - (b + (c * d)/e
a	-	(b + c * d)/e
a	-(	(b + (c * d)/e

Scanned with CamScanner

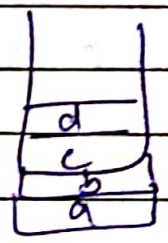
o/p Postfix	Stack	Input infix.	
ab	-(	b+c*d)/e	
ab	-(+	+c*d)/e	
abc	-(+	c*d)/e	
abc	-(+*	*d)/e	
abcd	-(+*	d)/e	
abcd*+	-	)/e	[Getting)]
abcd*+	-/	/e	
abcd*+e	-/	e	
		-end-	

∴ abcd*+e/-

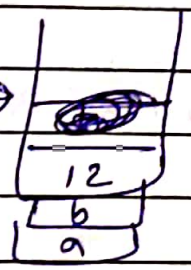
Q) Take a=2, b=5, c=3, d=4, e=2

Sol:

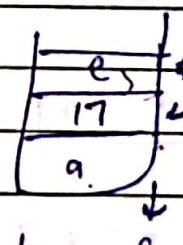
M-1 Stacks



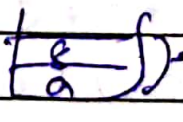
c*d
3*4
=12



12+b
=12+5
=17



17/2 = 8



a-8
2-8
=-6

M-2: (directly) in expression

2 - (5 + 3 * 4)/2

= 2 - (17)/2 = 2 - 8 = -6

```
while (!stack.empty())
{
    append stack.top() to right end of postfix;
    stack.pop();
}
```

Q) a*(b+c*d)/e abcd*+*e/

Scanned with CamScanner

Infix ~~No~~ Expression

<operand> <operator> <operand>

Precedence (BODMAS)		Associativity
1	( )	
2	^	RL
3	*, /	LR
4	+, -	LR

Associativity → Same priority then order

LR: Left to Right

Prefix expression — Polish Notation

(i) put bracket as per priority
(ii) bring that sign before that bracket

$4 * 2 + 3$

$((4*2)+3)$

Reverse str → Postfix → Reverse o/p postfix

$+ * 4\ 2\ 3$ ←

(Same as postfix.)

Q) $5 - 6/3$ write prefix

Sol: $(5 - (6/3))$

$- 5 / 6\ 3$ ←

$- 5 (6/3)$ $5 - (6/3)$

Postfix expression — Reverse Polish Notation

(i) ——
(ii) —— n after ——

$4 * 2 + 3$

$((4*2)+3)$

$4\ 2 * 3 +$ →

$(4*2)\ 3 +$

$4*2+3$

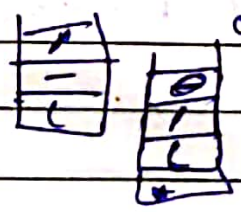
Q) $5 - 6/3$

Sol: $(5 - (6/3)) = 5\ 6\ 3 / -$ →

$5\ (6/3) -$ $5 - (6/3)$

Q) $(a-b/c) * (a/k-l)$

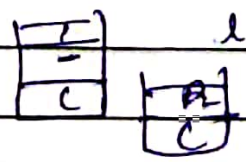
Sol: $a b c / - a k / l - *$



Q) $(a-b/c) * (a/k-l)$

Sol: $(l-k/a) * (c/b-a)$.

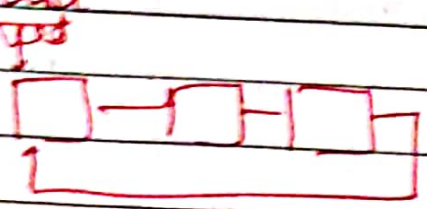
$l k a / - c b / -$



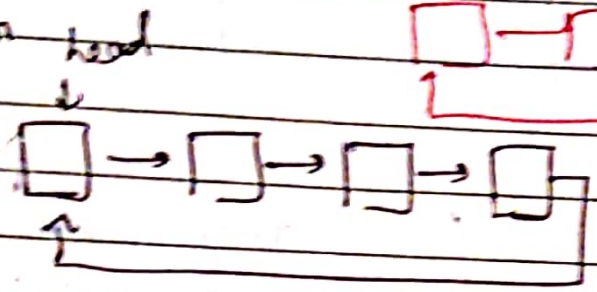
Scanned with CamScanner

Date: / / Page no:

Circular linked list

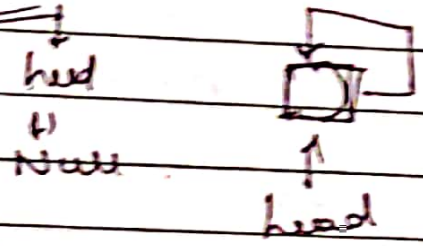


Circular linked list



Traversing as necessary even if Insert At Head

head = NULL



n → next = ~~NULL~~ n

head = n

// Display ckl

do {

temp != head → 1st for cond" false

∴ 1 time run no do while()

} while ()

① Delete

i<1 i<pos

~~pos-1~~ pos

temp

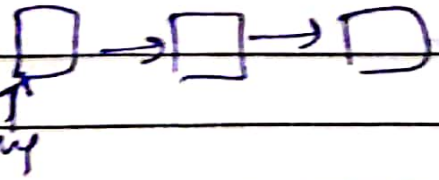
temp → next = temp → next → next

even if pos = last

delete at head

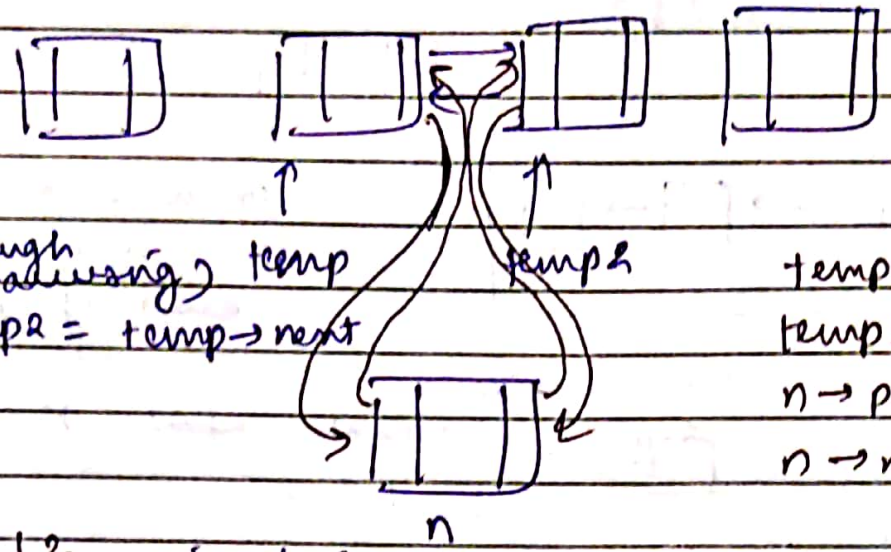
(i) traverse till last

② insertion



Scanned with CamScanner

Insertion in between DLL



(Through ~~traversing~~) temp2 = temp → next

temp → next = n
temp2 → prev = n
n → prev = temp
n → next = temp2

Q) L1 − L2 list of elements of present in L1 & not in L2

(1) Final list

L1: 1 → 4 → 5 → 8 (h1, h1, h1)

L2: 2 → 4 → 7 (h2, h2, h2)

1 < 2

(1) ~~temp~~ h1 → data < h2 → data
h3 → next = h1
temp1 = temp1 → next
h3 = h3 → next

(2) 4 > 2

(3) h1 → data == h2 → data

h1 → data > h2 → data
h2 = h2 → next

1, 5, 8

(i) h1 → data < h2 → data

1 → 4 → 5 → 8 (h1, h1, n)

(ii) h2 → data < h1 → data
h2 = h2 → next

2 → 4 → 7 (h2, h1, n)

(iii) h1 → data ==
h2 → data
h1 = h → next
h2 = h2 → next

5 2 3 4 5

Scanned with CamScanner

06.09.2022 ~~Sat~~ Tuesday.

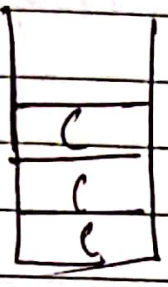
Date: / / Page no:

Stacks

Q) Balanced Brackets — Easy

(((()))()) ✓ (()()) ✓ (((() ✗) ✗

```
for (i=0; i<n; i++)
    if (arr[i] = ')' && st.top() = '(') {
        continue;
        st.pop();
    }
    else if (arr[i] = '(') {
        st.push('(');
    }
if st.empty() && arr[i] = ')'
```



pro n

① '('
push.

② ')'
if st.empty() → wrong.
~~push~~ pop()

③ at last st.empty() → wrong.

level-2

Q) {[( )]} ✓ {([]})

if ① if any opening bracket then push

② else check whether closing check top for open corresp-ending brackets.

③ pop empty() → NO / yes

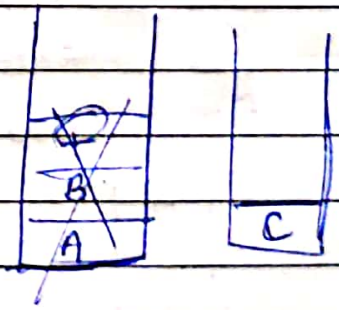
yes → yes
NO → NO

While overcomes pop check for if it is empty.

Q) Postfix evaluation — Easy

AB+C- → ~~vector~~ vector.

```
for (int i=0; i<s.length(); i++)
    if (isAlpha(s[i])) {
        st.push()
    }
```



res = A+B.-C.

Tough 1000 200 3 + 200 /53 * 47 +

842

Scanned with CamScanner

08, 09, 2022
Thursday

Date: / / Page no:

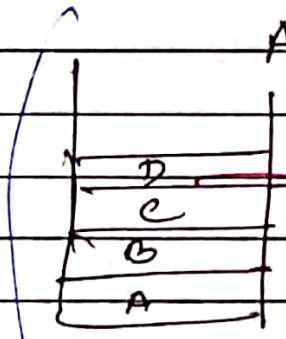
Q) Implement 2 stacks in an array.

A	C	B		B					
0	1	2	3	4	5	6	7	8	9

In ~~copy~~ VS Code assignment-04

Activation Record Management (ARM)

↳ how memory is allocated dynamic & static
↳ which program is activated / not



A → B → C → D

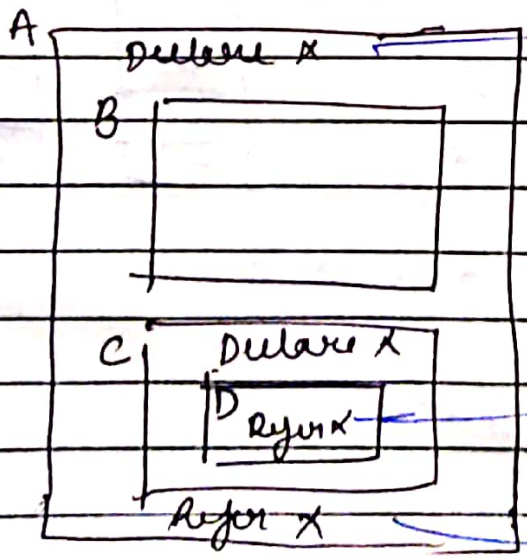
returns when D ends & go to C

⇒ C is activated (pop when it ~~is~~ ~~used~~ ends)

activated

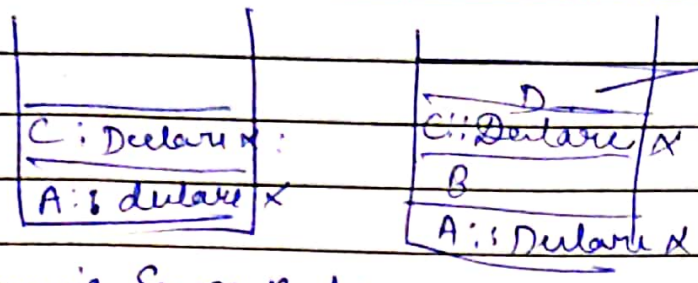
governing (local & global) scope of any dynamic & static variables.

Use Activation Record Management.



→ Here, C declared x will be used

→ Here A Global declared x will be used



Dynamic Scope Rule

Declaration of the variable is done least recently used declaration of that variable.

(Active)

Q)

```
MAIN
    Boolean b = true;
    Procedure P:
        Begin
            Output (b)
        end
    Procedure Q
        Boolean b = false
        Begin
            output (b)
            call P
        End

Begin
    Call P
    Call Q
End
```

Solve normally as we do.

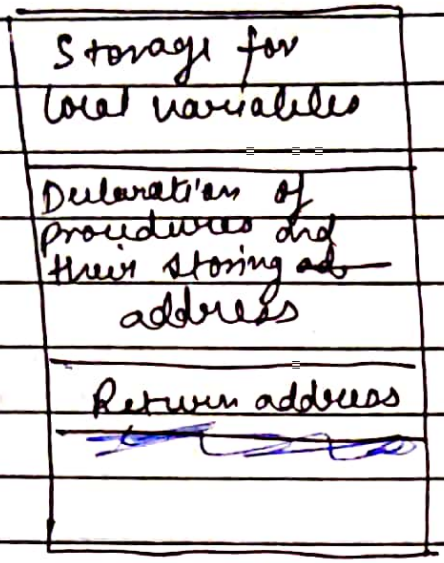
write o/p when there is (i) Static Scope Rule (ii) Dynamic Scope Rule

for dynamic when P is called from Q then Q's b is activated. b = null take Q's b value & Not Global variable

Q) SOL: Static → b = true b = false b = true

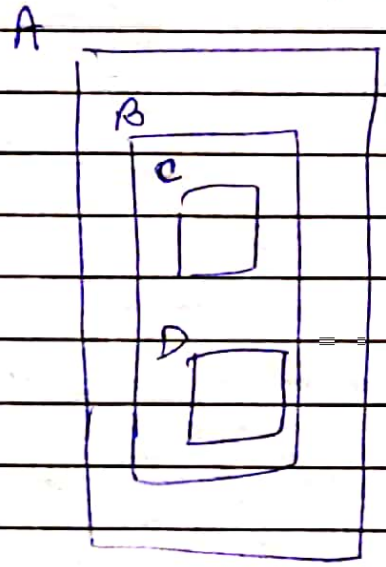
Dynamic → b = true b = false b = false.

Scanned with CamScanner



Pointer to the location of calling function

// This ptr signifies scope rule



Stack

↓Top

A	Activation Record of A is pushed
AB	A invokes B
ABC	B invokes C
AB	C completes
ABD	B invokes D
AB	D completes
A	B complete

```
MAIN
    ABC  Fn begins
        Procedure Q
        Begin
            A = A+2
            C = C+2
```

Scanned with CamScanner

```
7        end
8        Procedure R
9            C integer
10           Begin
11           C = 2
12           Call Q
13           B = A + B
14           end
15       Procedure S
16           B, C integer
17       Procedure Q
18       Begin
19       A = A + 1
20       C = C + 1
```

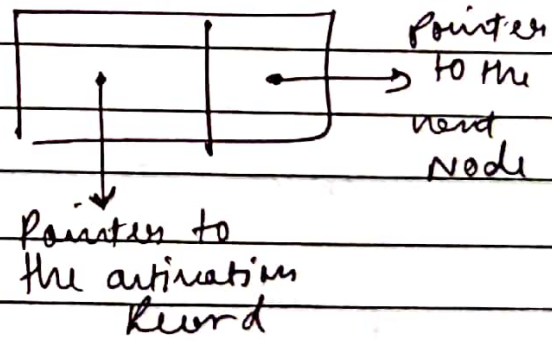
```
21       end
22       Begin
23       B = 3
24       C = 1
25       call Q
26       Call P
27       end
28       Begin
29       A = 1
30       B = 2
31       C = 3
32       call R
33       Call S
34       end
```

Main

Main

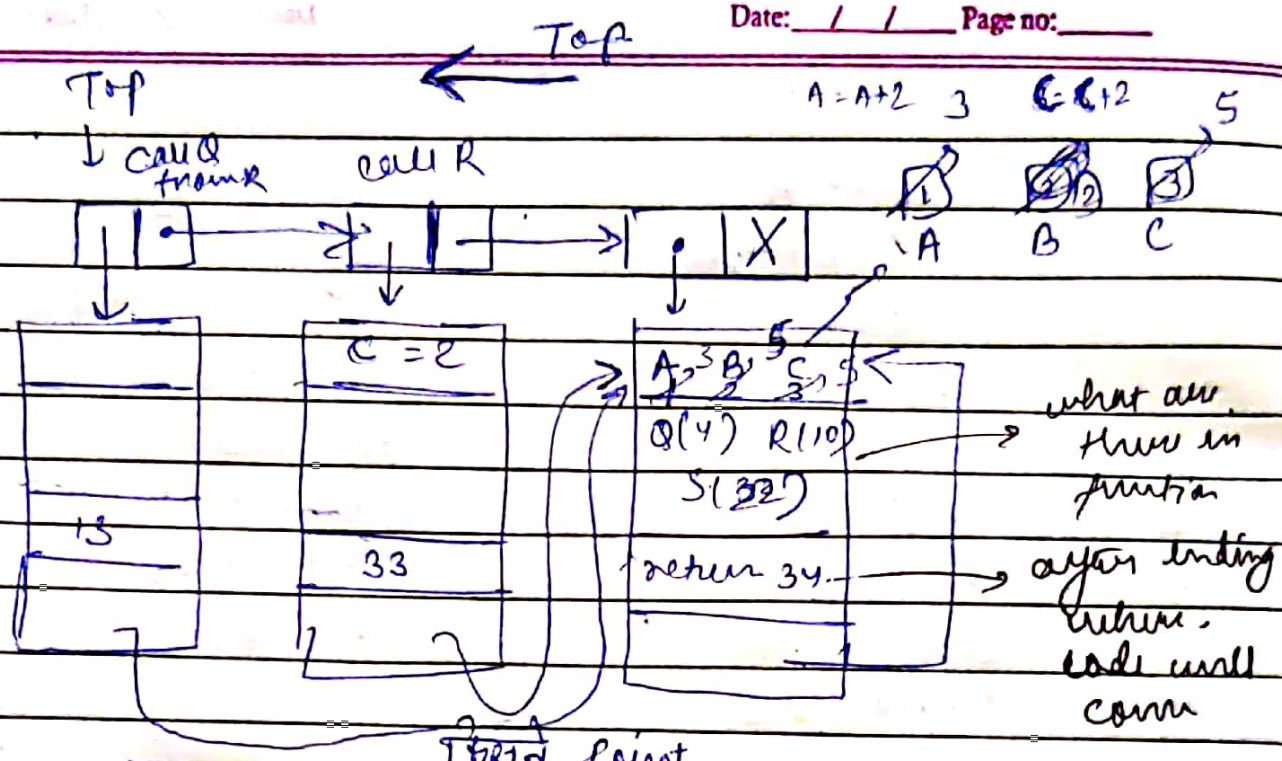
A B C integer
Q(4), R(10) S(22)
Return address → 34
Calling program

Implementation using linked list



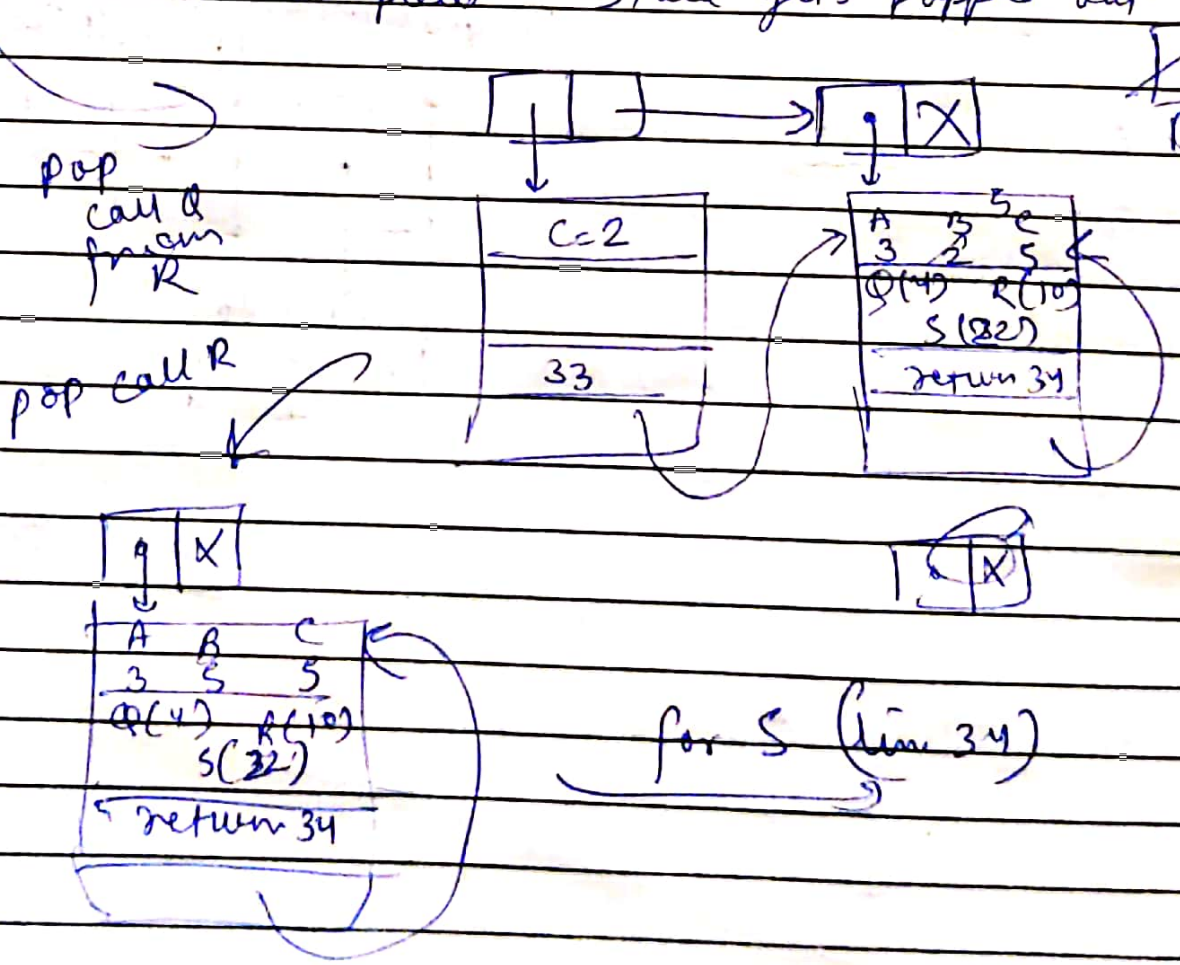
Scanned with CamScanner

Date: / / Page no:



As it completes Stack gets popped out

B = A+B
= 3+2
= 5

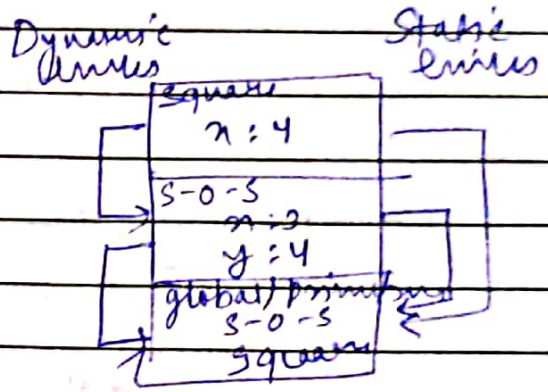


Date: / / Page no.

Dynamic Links:- Points to where it was called from
Known ~~at~~ during run time

Static Links:- Points to the activation record of the procedure where this one was defined
Points to enclosing scope, does not change during run time.

```
(define (square x))
  (* x x)

(define (s-o-s x y)
  (+(square x)(square y)))

(s-o-s 3 4)
```



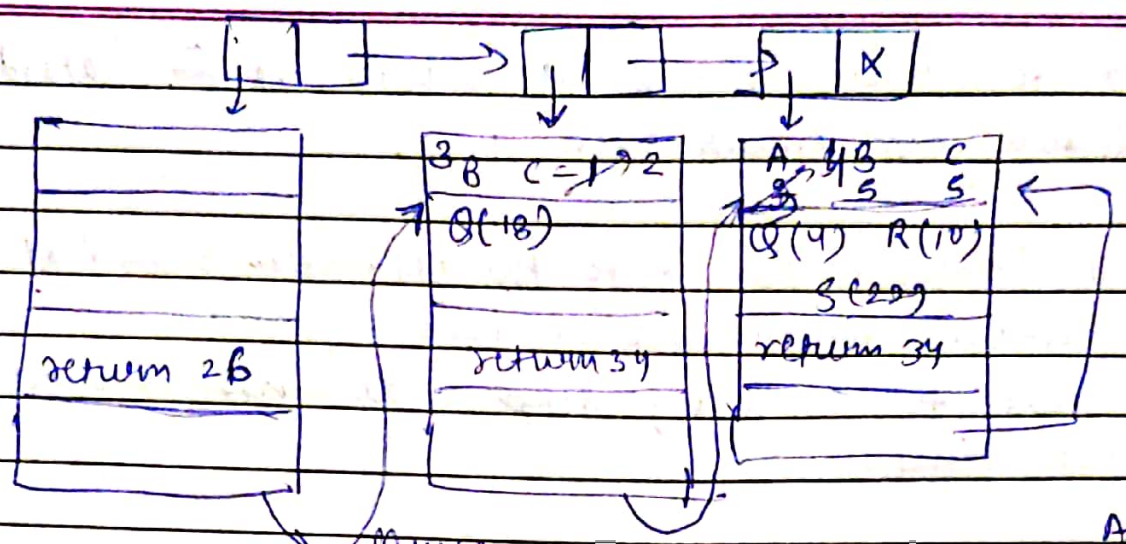
In Javascript (Closures) & Python (Continuation) is not supported by LIFO ∴ Scheme for allocation & deallocation activating records is different prev. activation Records

Scanned with CamScanner

at line 25 call Q from S

call S (33)

Date: / / Page no:



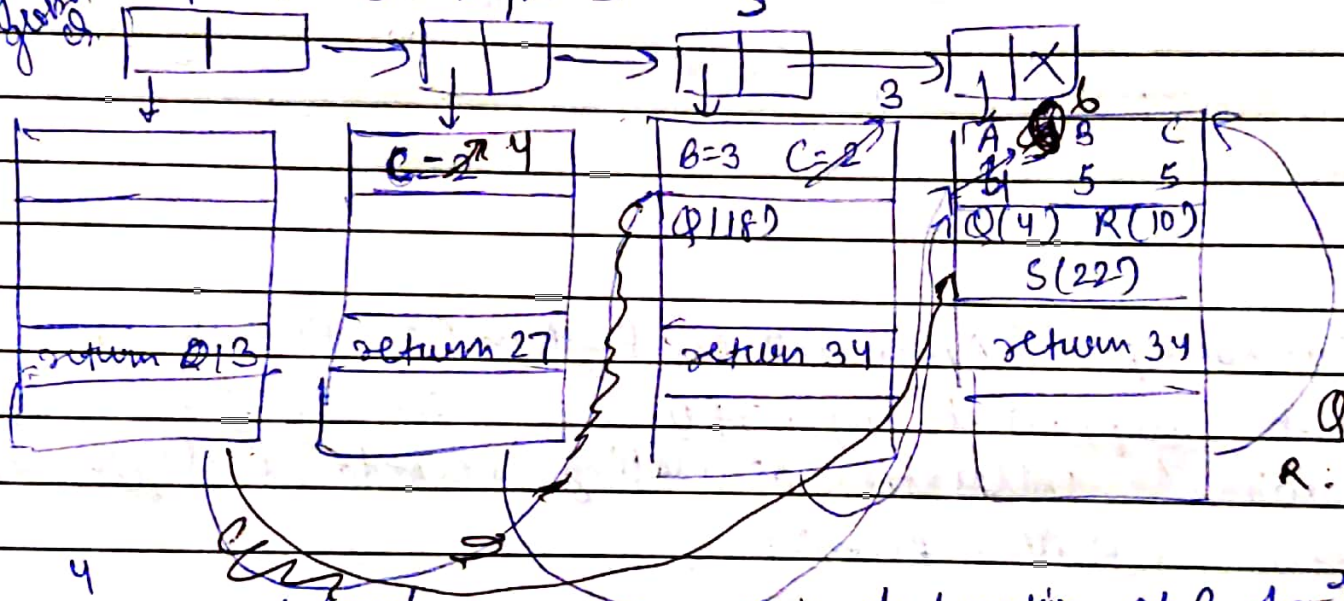
Q was declared in S

Q from S

pop ()

C = C+1 But here is no A

A = A+1



A B C

Q: 6 5 7

R: 6 11 7

declared in S

No declaration of R in S

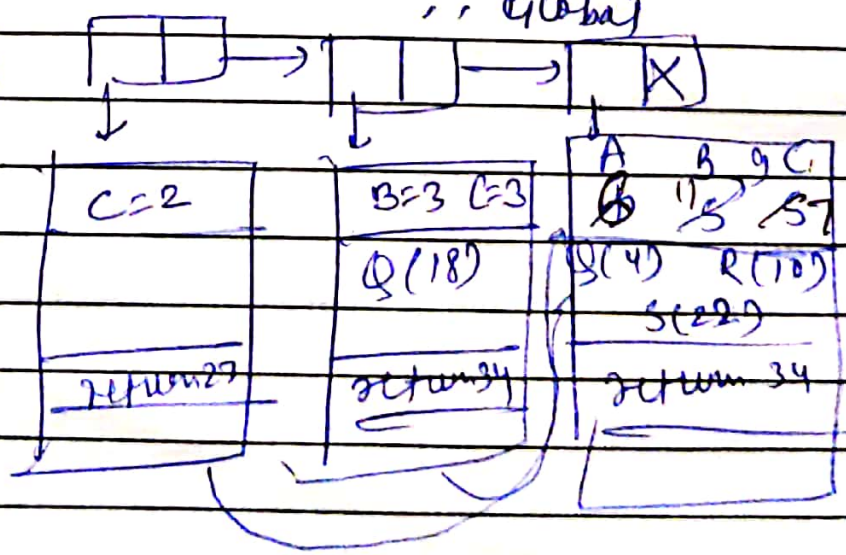
∴ Global

A = A+1

Global

2+1=3

pop Q from



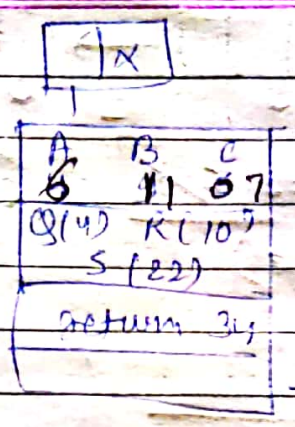
6 11 7

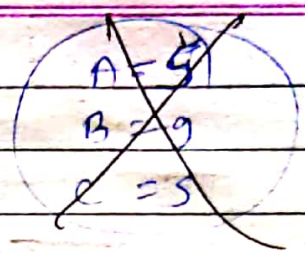
Global

B = A + B = 4 + 5

Scanned with CamScanner

Both pops.





A = 6
B = 11
C = 7

pop out.

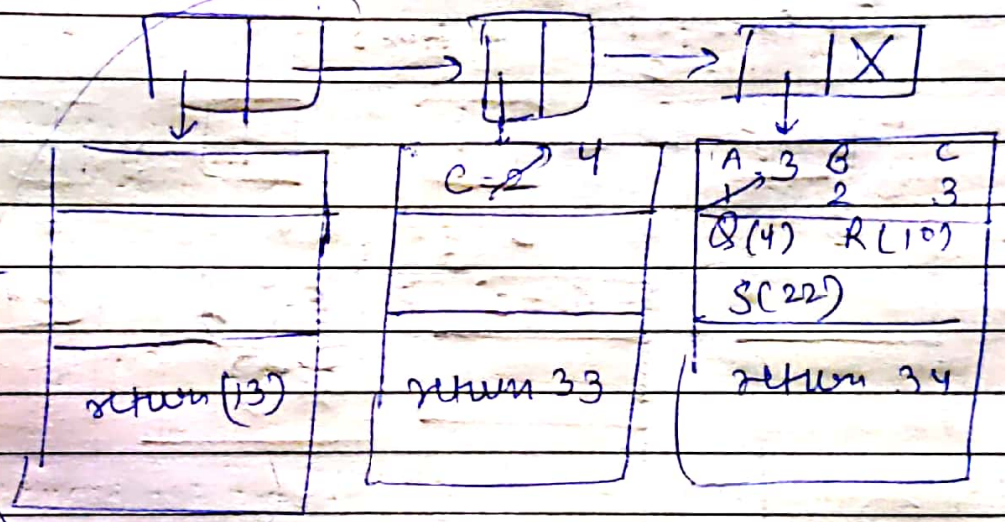
end

Dynamic Scope

No dynamic pointers

Set last entered stack

Only 3 columns

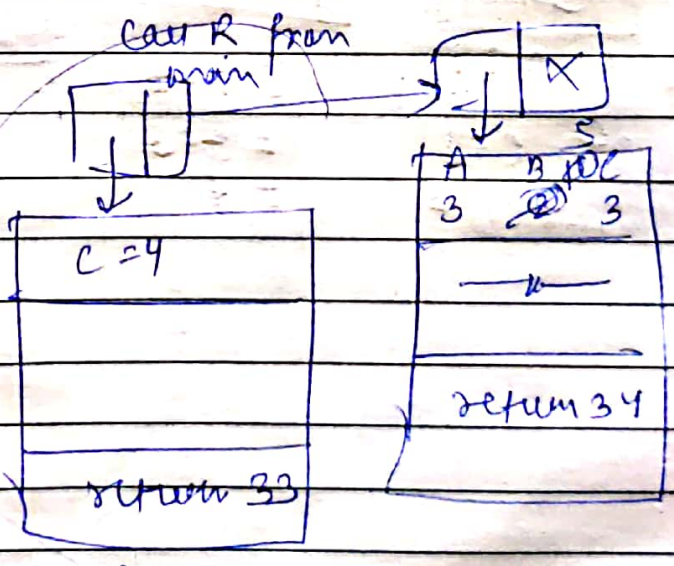


from main
A = A + 2
= 1 + 2
= 3

from R C = C + 2
= 2 + 2
= 4

Call Q from R

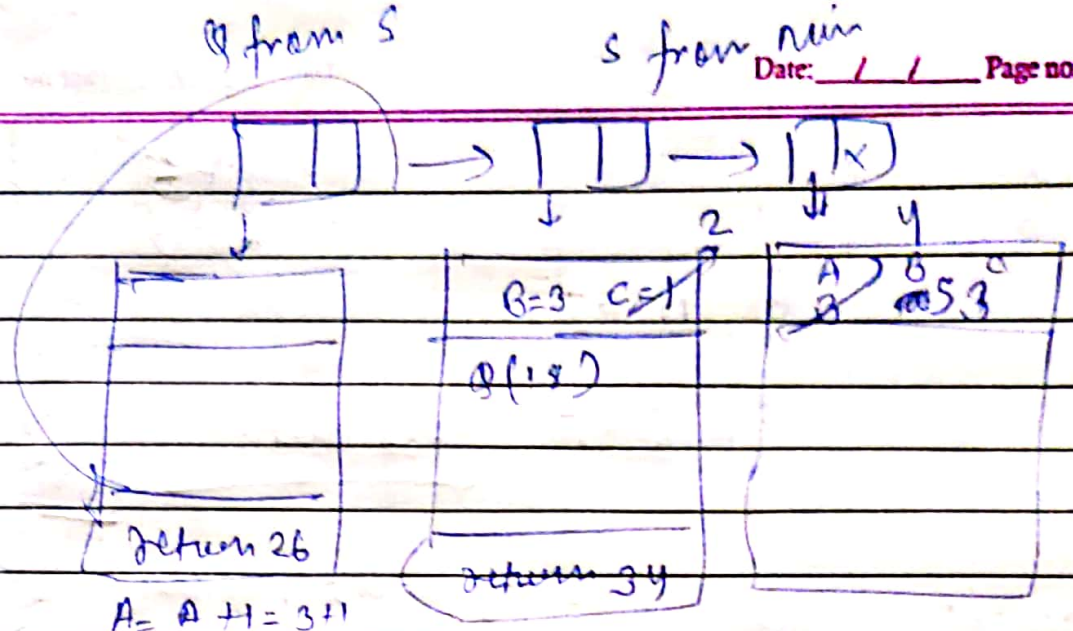
Call R from main



B = A + B
= 3 + 2
= 5

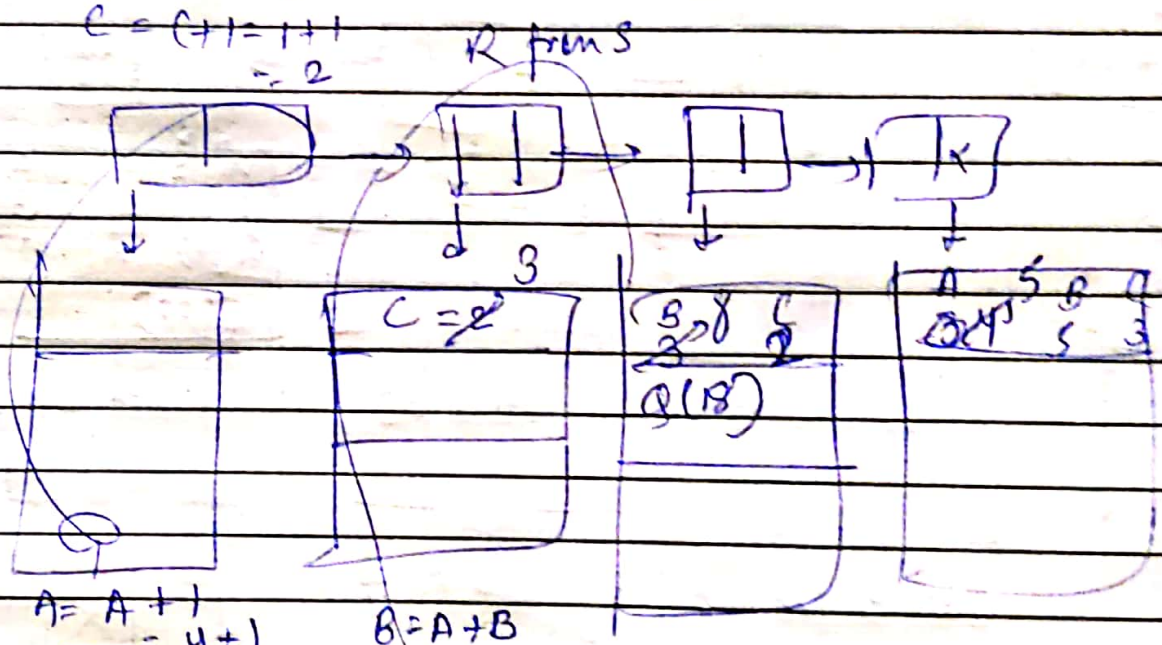
Pop

Date: / / Page no:



$A = A + 1 = 3 + 1$
$= 4$

$C = C + 1 = 1 + 1$
$= 2$



$A = A + 1$
$= 4 + 1$

$C = C + 1$
$2 + 1 = 3$

$B = A + B$
$= 5 + 3$
$= 8$

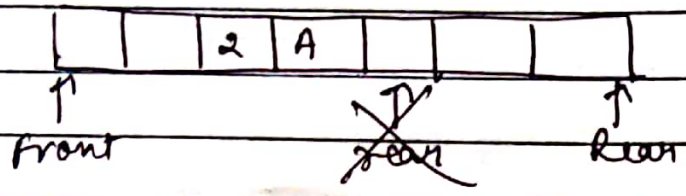
Scanned with CamScanner

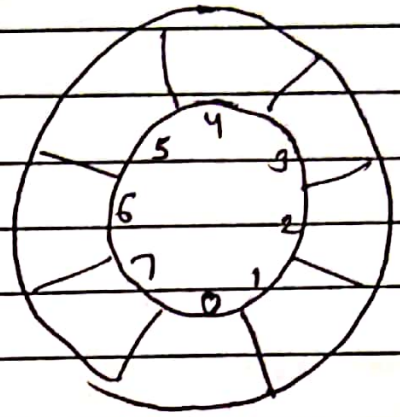
12.09.2022
Monday

Date: / / Page no:

FIFO
First in First out

QUEUES





adding an element:
Check → queue should ~~not~~ be empty

Deletion
{ ~~Add~~ from front, ~~de~~ addition at end } circular Queue

Implementation using arrays

(Rear + 1) mod N

size of an array

(Front + 1) mod N

```
typedef struct {
        int count;
        int Front;
        int Rear;
    Item type Items [max queue size];
} Queue;
```

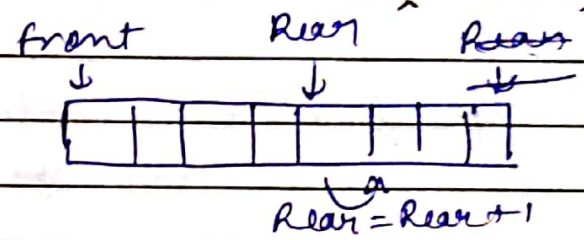
linked list
isFull()
no use

```
void Initialize Queue (Queue *Q) {
        Q → count = 0;
        Q → Front = 0;
        Q → Rear = 0;
}

int Empty (Queue *Q) {
        return (Q → count == 0);
}
```

Scanned with CamScanner

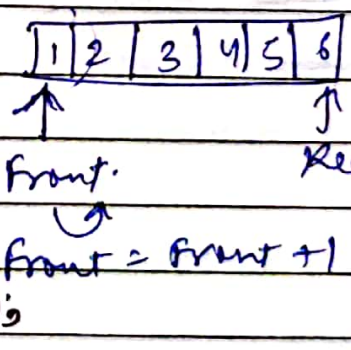
```
Insert (ItemType R, Queue *Q) {
    if  Q -> count == MaxQueueSize {
            Error "Queue is Full";
        }
    else {
        Q -> Items [Q -> Rear] = R;
        Q -> Rear = (Q -> Rear + 1) % MaxQueueSize;
        ++ (Q -> count);
    }
}
```

can use is empty() function



or Removing

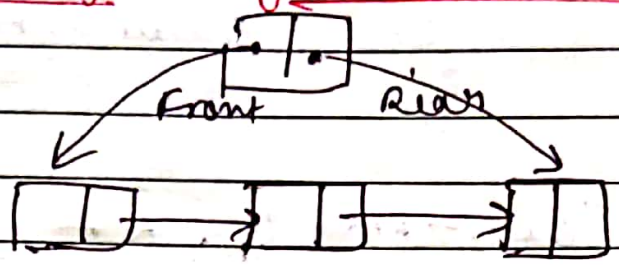
```
int Deleting (ItemType F, Queue *Q) {   //from front
    if ( Q -> count == 0 ) {
            Error " No element present";
        }
    else {
        Items k = Q -> Items [Q -> Front]
        Q -> Front = (Q -> Front + 1) % Max Queue size;
        --(Q -> count);   return k.
    }
```

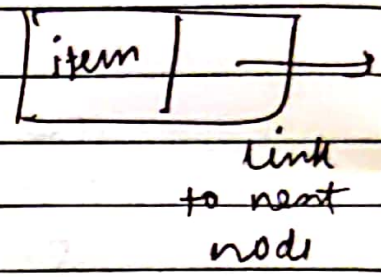


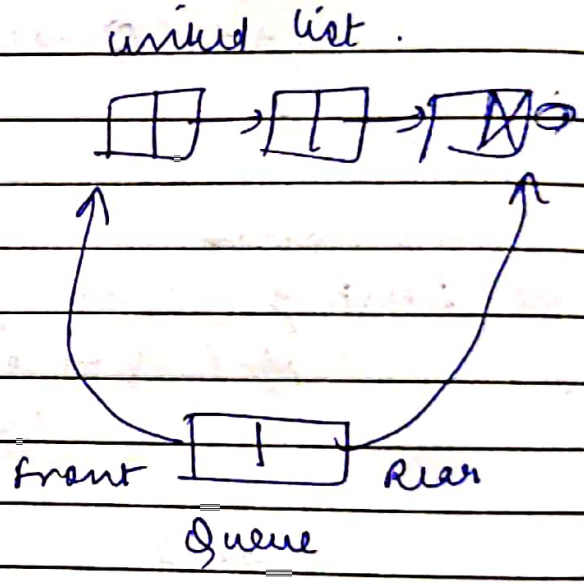

```
int Full (Queue *Q) {
    return (Q -> count ==
                Max Que size);
}
```

Scanned with CamScanner

Implementing using linked list








```
typedef struct QNode {
        item Type item;
        Struct Qnode *link;
    } Queuenode;

typedef struct {
        Queue node * Front;
        Queue node * Rear;
    } Queue;
void initialize Queue (Queue *Q) {
    Q → Front = Null;
    Q → Rear = NULL;

}
void initialize Queue Node (Queue * Q) {


int bool isEmpty (
```

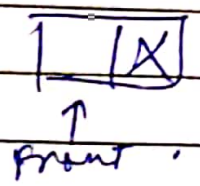
Scanned with CamScanner

(ii)

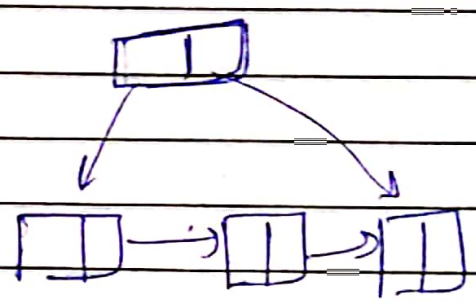
if memory is allocated then ~~delete it~~

// Adding (i) getNode() → ~~[illegible]~~ → enqueue

// Front (i) check front → next == NULL → dequeue
then ~~no need to~~ there is only one node

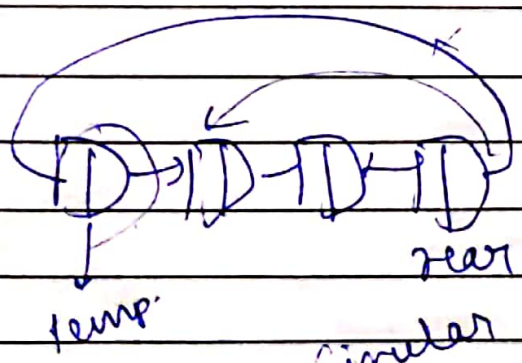



```
class Node {
    int data;
    Node * next;
    Node() {
        data = val;
        next = NULL;}
};
```



Rear → null

n → next = n;



rear → next = front

circular

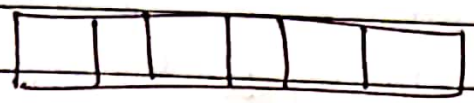
//

Date: / / Page no:

Implement a Queue using Stacks

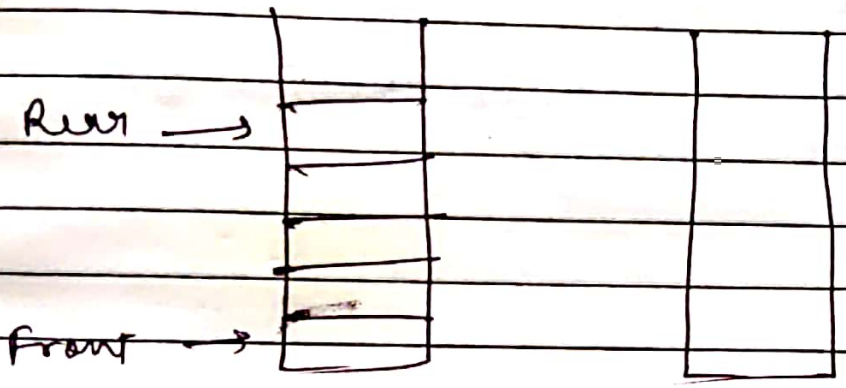
Adding → O(1)
Deletion → O(n)

Queue



Using 2 Stacks

To avoid overflow



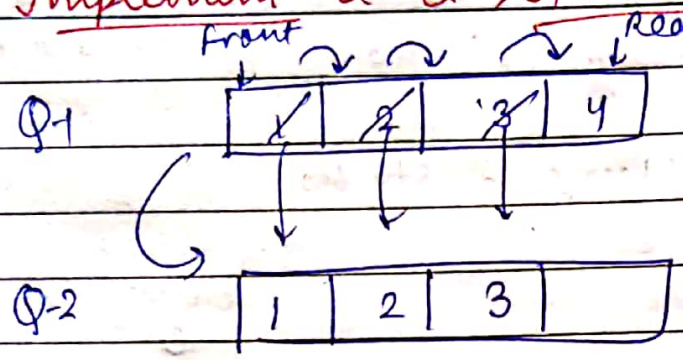
Addition in Queue == Adding in Stack.

Deleting in Queue in front
push element from stack-1 to stack-2 until empty
& then use stack-2.top()

```
class Queue {
    ~~int~~ stack<int> S1;
            stack<int> S2;
    ~~Queue()~~ {
    }
    void enqueue(int val) {
        S1.push(val);
    }

    void dequeue() {
        if(S1.empty() && S2.empty())
        { cout << "Empty Queue" }
            return;
        if (!S1.empty()) {
            while(!S1.empty()) {
                S2.~~empt~~push(S1.top())
                S1.pop()};
        return S2.top();
    }
```

Scanned with CamScanner

Implement a Stack using Queue



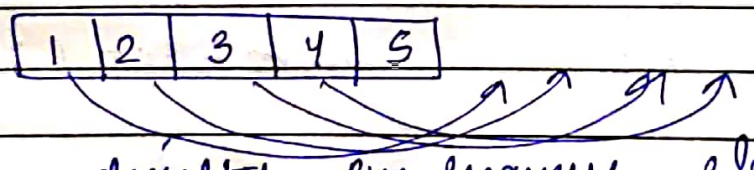
Push → O(1)
pop → O(n)

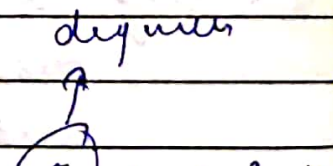
Delete 4

Stack.pop()

swap names of Q-1 & Q-2

Single Queues:





~~Enqueue~~ push: directly pro enqueue element

pop: (i) ~~enq~~ dequeue (n-1) elements
(ii) enqueue each element
(iii) dequeue element to

Scanned with CamScanner

front back/rear.

Date: / / Page no:

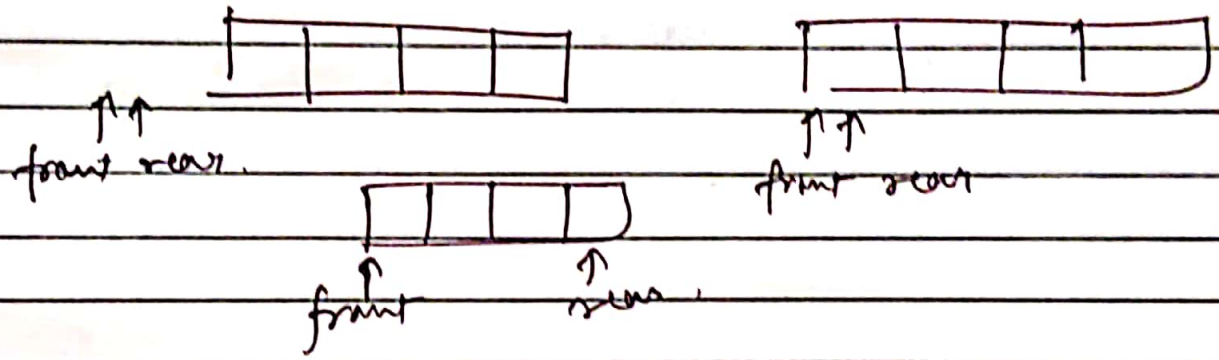
Queue

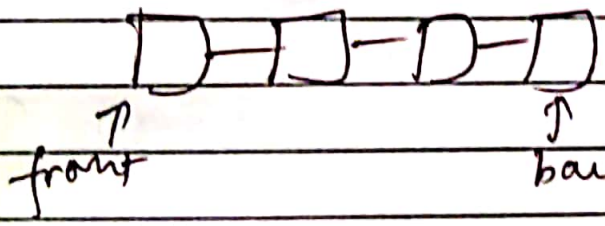
① enqueue(x) → push element at back → check for size and → rear++;

② dequeue() → pop an element from front → Check for empty → else front++

③ peek() → returns front value

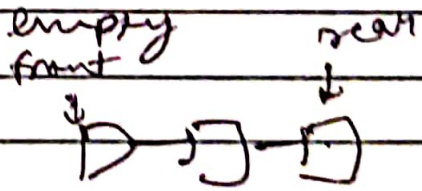
④ empty() → return queue is empty or not





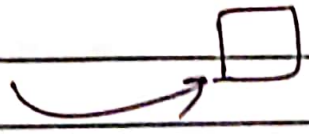
front = NULL

empty

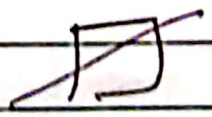


rear->next = n

enque (x)

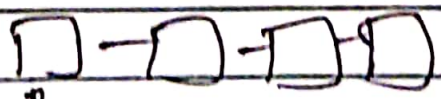


empty



front → NULL.

dequeue()

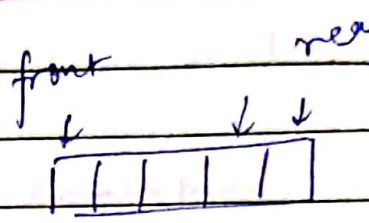


front

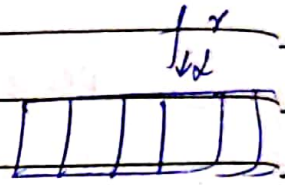
front = front → next

Date: / / Page no:

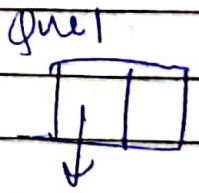
Stack using Single Queue




```
class Stack {
        queue <int> q;
    void push( int val ) {
            q.enqueue (val);
    }
    void pop ( ) {
            while ( front != rear ) {
                int q.enqueue (q.peek()); 
                    q.dequeue;
            }
```



```
typedef struct Stack {
        Queue * Que ;
        int size; } stack;
```




```
Stack * Create Stack ( ) {
    Stack * Stk = (Stack *) malloc (sizeof (Stack));
    Stk -> que1 = create Queue ( ); // already been discussed here
    return Stk; }
```

creating memory for structure stack

```
void stack push (stack * stk, int value) {
        enqueue (stk -> que1, value)
        stk -> size = stk -> size + 1; }

int stack
void pop ( Stack * stk) {
        int value = 0, s = stk -> size;
while s>0 {
        value = dequeue (stk -> que1);
        if (s>1)
            enqueue (stk -> que1, value);
            s--; }
stk -> size = stk -> size - 1;
return value;
}
```

Application of Queues :-

1. Operating System
2. Carry order ~~Transport~~ & Transport
3. Breadth first search.
4. Priority Queue priority Scheduled tasks.

P_1	P_2	P_2	P_3	P_4	P_5

Highest priority processes at the top

Scanned with CamScanner

Always try to give answer in O(1) time complexity

Date: / / Page no:

Priority Queue

P_1	P_2	P_5	P_3	P_6	4
1	2	3	5		

Insert

In arrays:

Advantage: [Searching for right location will be efficient O(log n) → Binary

Disadvantage: Insertion takes O(n) time

In Linked lists:

Advantage: Insertion will take O(1) time

Disadvantage: Traversal to find location will take O(n) time.

Josephus Problem :-

n persons are standing in a queue & they are to be executed & only one person will get freedom. Counting starts from front.

Every time (k-1) persons are removed from front and add at end k th person is executed

Last Remaining person is free. Find ~~his~~ location

Date: / / Page no:

1 2 ~~3~~ 4 5 6 7 8 $k = 3$

4 5 ~~6~~ 7 8 1 2

7 8 ~~1~~ 2 4 5

2 4 ~~5~~ 7 8

7 8 ~~2~~ 4

// Remaining method

4 7 ~~8~~

// Queue method 4 7

1 2 ~~3~~ 4 5 ~~6~~ 7 8

7 8 ~~1~~ 2 4 5

2 4 ~~5~~ 7 8

7 8 ~~2~~ 4

4 7 ~~8~~

4 ~~8~~ 7 ~~4~~ 7

Sol: int main ( ) {

int n; cin>>n;

int k; cin>>k;

1 ~~2~~ 3 ~~4~~ 5

5 ~~1~~ 3

3 ~~5~~

n = 5 k = 2

1 ~~2~~ 3 4 5

3 ~~4~~ 5 1 3

5 ~~1~~ 3 3

3 5

1 2 ~~3~~ 4 5 6 7 k=3

7 1 ~~2~~ 4 5

4 5 ~~1~~

1 4 ~~5~~

1 4 ~~1~~ 4 — safe.

1 2 ~~3~~ 4 5 6 7

0 1 2 3

n-1

1 ~~2~~ 3 4 5 6 7

15.09.2022 Thursday.

Date: / / Page no:

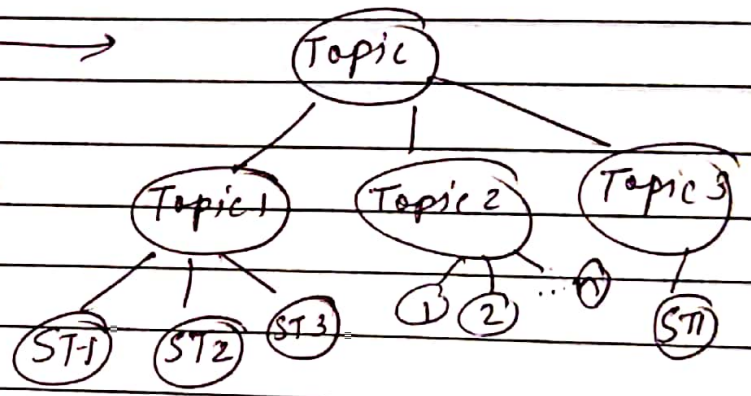
TREES

→ Searching an element heirarchally

- Topic 1
  - ST1 (subtopic)
  - ST2
- Topic 2
  - ST1
  - ⋮
  - STn
- Topic 3
  - ST1

heirarchical pattern of data reduces Time

- Topic 1
- ST1
- ST2
- Topic 2
- ⋮



//Non linear Data structure

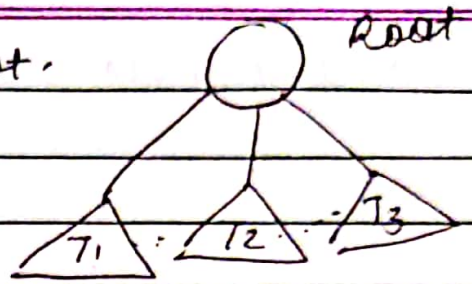
Dictionary search operations

- Static → search
- Dynamic → Insertion, Deletion, Search
- → Trees ~~can~~ is better options

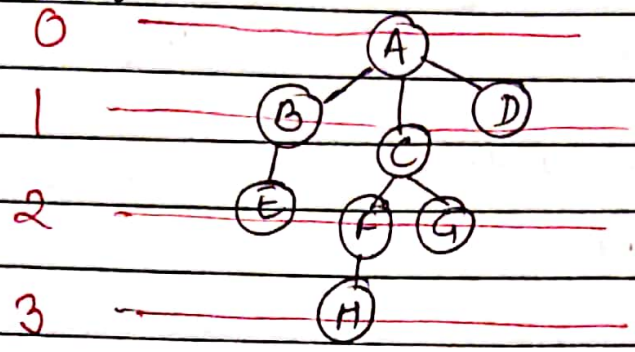
Tree:- Consists of one or more items called nodes. One specially designated node called the ~~node~~ root. and a set of zero or more non empty subset of n nodes denoted $T_1, T_2, \dots T_k$ where each itself is a Tree.

Scanned with CamScanner

These are called subtrees of the root.



T1, T2, ... Tn are child nodes of the Root.



B, C, D are siblings

Children of the same node.

E, F, G are grandchild of A

or A is the grandparent of E, F, G

Depth of a Node : Distance of a Node from root along the unique path.

Depth of E = 2 Depth of H = 3

Depth of B = 1

Depth of A = 0

Height : The maximum of distance of root from any leaf node. 3

Distance of A from H = 3

Degree : No. of children of a Node.

Levels : ~~Maximum~~ Height of the tree is called levels

NOTE: ① The leaves are at degree zero

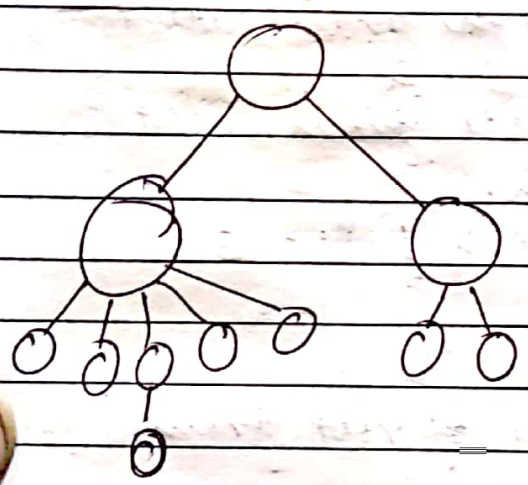
Scanned with CamScanner

Binary Trees — where each node has 2 child

n-ary Trees — where each node have atmost n children

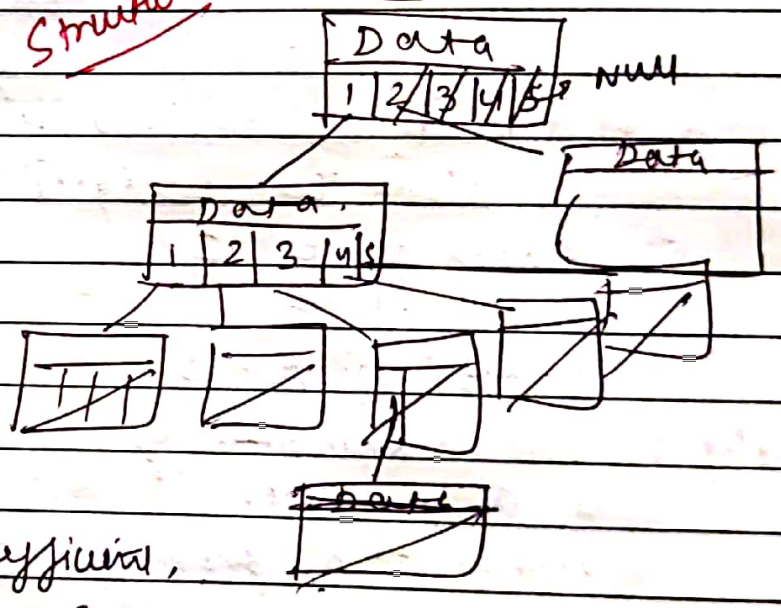
↳ even a single node has n child then too it is n-ary Trees.

How Trees can be stored?



(5 ary tree)

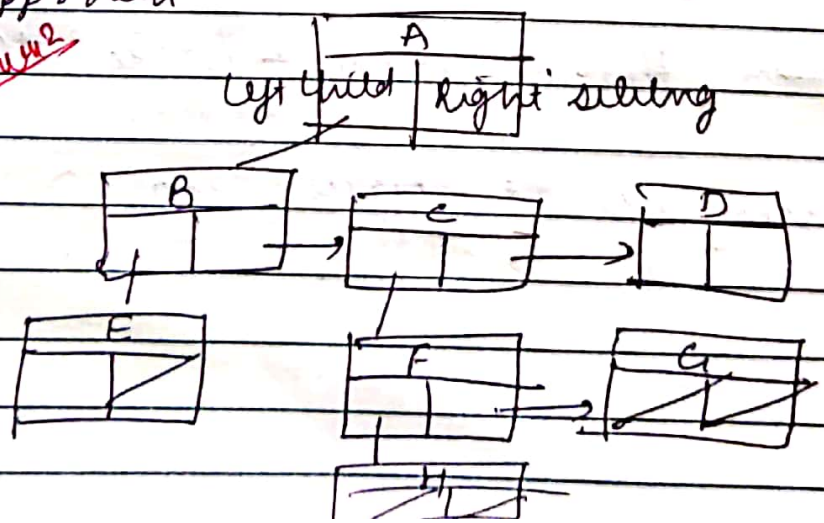
Structure 1



Time efficient, Space inefficient way

Better Approach

Structure 2



Scanned with CamScanner

2nd Approach is Space Efficient but it compromises on Time (Hierarchy

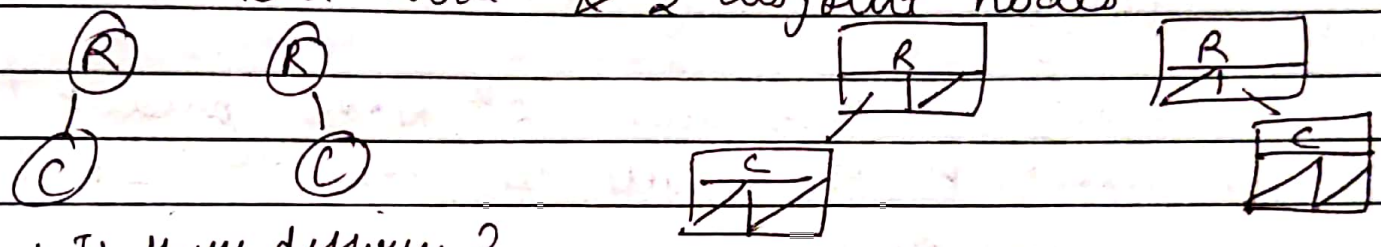
→ Ordered Tree

Binary Tree A Tree ~~without~~ even with 0 zero Node is Binary Tree

At most 2 Child

~~Particular~~ There should be clear distinction betn left & right child.

Root Node & 2 disjoint nodes



Is these different?

These two are different.

Structure of Node of Binary Tree

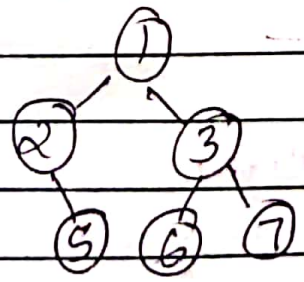
Data	
Left Link	Right Link

Linked list structure of BT

Array Representation

1	2						

~~2pt~~



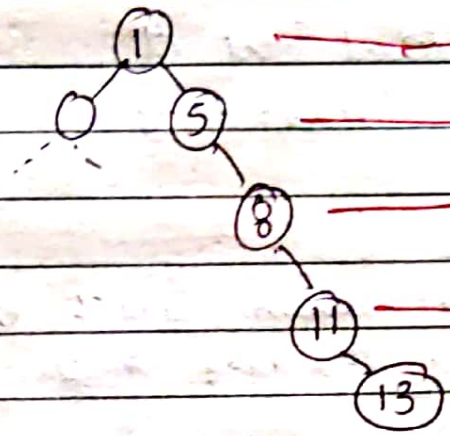
Root/Node

Left child (2 * Root/Node)

Right child (2 * Root/Node + 1)

Scanned with CamScanner

Date: / / Page no: ~~node~~ of space



$2^0 = 1$ (Max^m Number ~~of nodes~~ of space at a given level)

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

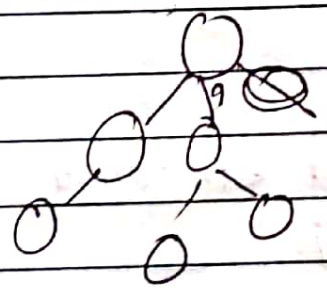
$2^4 = 16$

$2^0 + 2^1 + 2^2 + \dots 2^4$

$= 31$ ~~node~~ space is required

NOTE:

1. On level K, there will 2^k nodes present
2. For height = K there will be total $2^{(K+1)} - 1$ nodes present.



2^0

2^1

$\vdots$

2^k

Total No. of Nodes =

$2^0 + 2^1 + 2^2 + \dots 2^n$

$= 2^0 ($

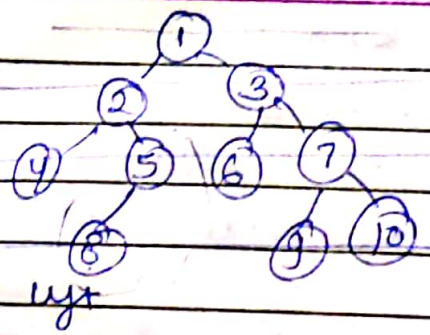
Traversal Technique

1. Inorder Traversal (Left Root Right) Always LR
2. Pre Order Traversal (Root Left Right)
3. Post Order Traversal (Left Right Root)

Scanned with CamScanner

16.09.2022
Friday.

Date: / / Page no:



① 4 2 ~~5~~ 8 1 6 3 9 7 10
② 1 2 4 5 8 3 6 7 9 10
③ 4 8 5 2 6 9 10 7 3 1

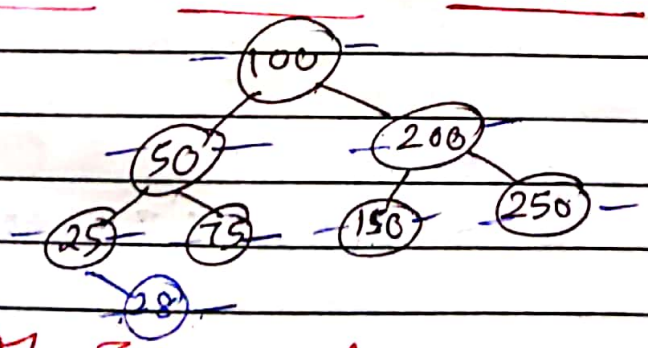
BFS search 1 2 3 4 5 6 7 8 9 10
(Level wise)
Breadth

BT Array has searching benefit.
Sorted Array O(log n) → Binary Search
Disadvantage : Insertion → O(n)

Linked list has Insertion/Deletion benefit → O(1)
Disadvantage : Traversing O(n)

To get benefit of both we use Tree
Binary Search Tree

BINARY SEARCH TREE



Preorder
100 50 25 28 75 200 150 250

28 25 75 50

3 actions
Move left
Visit Node
Move Right

All combⁿ give diff. Traversals but convention said ① left & ② Right

Types of Traversal

① Preorder
② Inorder (only for Binary Tree)
③ Postorder
④ Level order Traversal

Scanned with CamScanner

Inorder	Preorder	Postorder
move left	Visit Node	move left
Visit Node	move left	move Right
move Right	move Right	Visit Node

```
Inorder(BSTnode *v)
if v=NULL return;
Inorder(v.left())
Print v->data
Inorder(v.right())
```

```
Preorder(BSTNode *v)
if v=NULL return;
print v->data
Preorder(v.left())
Preorder(v.right());
```

```
Postorder(BSTNode *v);
if v=NULL return;
postorder(v.left())
postorder(v.right())
print(v->data);
```

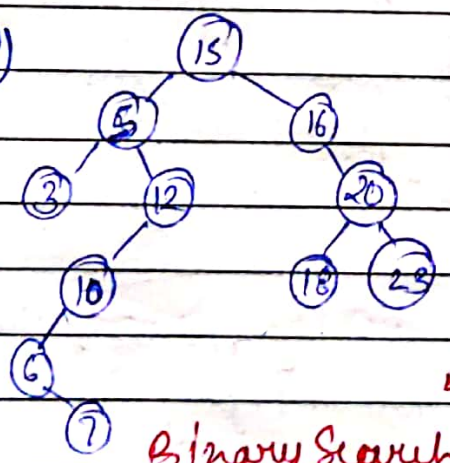
Inorder
25, 28, 50, 75, 100, 150, 200, 250

Postorder
28, 25, 75, 50, 150, 250, 200, 100

Preorder 100, 50, 25, 28, 75, 200, 150, 250

If we want to traverse using iterative way then we need to create parent [ ] array to traverse (stack will be used)

Q)



Binary Search Tree (BST)

Sol: Inorder
3, 5, 6, 7, 10, 12, 15, 16, 18, 20, 23

Preorder
15, 5, 3, 12, 10, 6, 7, 16, 20, 18, 23

Postorder
3, 7, 6, 10, 12, 5, 18, 23, 20, 16, 15

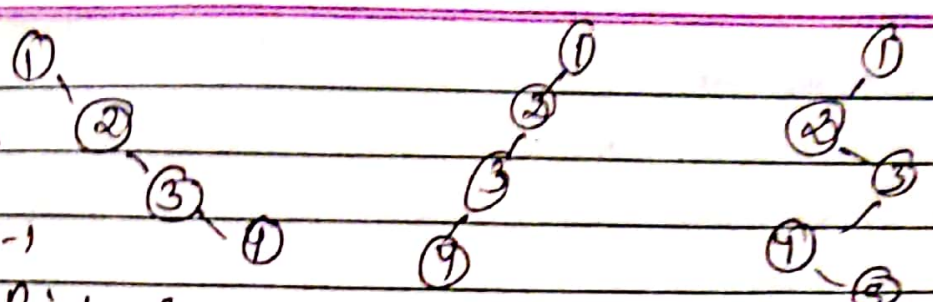
NOTE:
To get elements in sorted order from BST we use Inorder Traversal

18.03.2022
Sunday

Date: / / Page no:

Nodes: n

Height: n-1



Right Skew BST | Left Skew BST | BST.

In case of BST, ~~Traversing~~ ~~finding~~ searching takes O(n) time.

In Case of Balanced BST, searching take O(logn) time

Level Order Traversal

Queue, vector<vector>

↓

push(root)

↓

1 (Queue empty) for (0 to size of Queue). → front → left, right → push.

↓

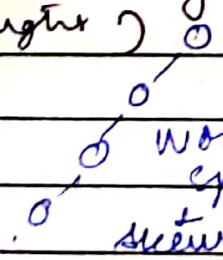
~~push~~ pop.

put a level
Remove them
put neighbours.

Maximum Depth of Binary Tree / Height of a Tree

Recursion / level order Traversal

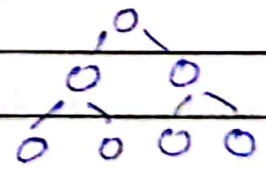
Auxiliary space O(height)



Worst space
skew Tree

↳ dissadvantage → Queue DS O(N)

worst case when last level is having all nodes



Recursive

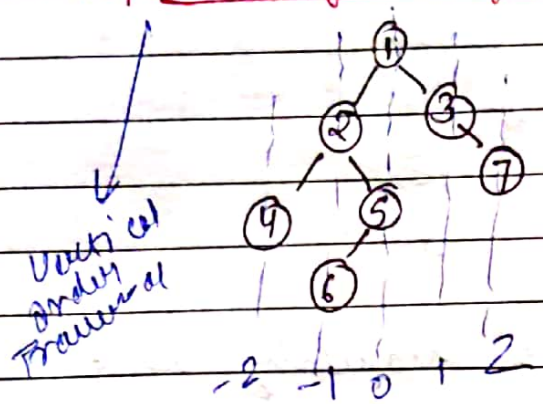
1 + max(l, r)

Date: ~~ordered~~ Page no:

BFS

queues ordered map

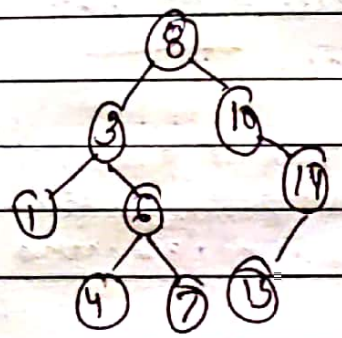
Top View of Binary Tree

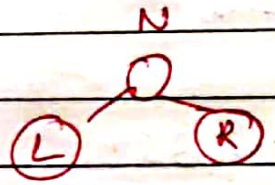


Vertical order traversal

4 2 1 3 7

Binary Search Tree





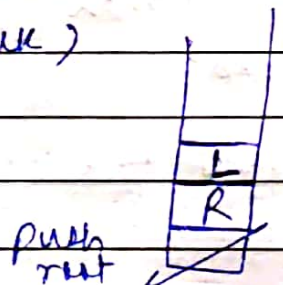
$\boxed{L < N < R}$

left subtree → BST
right subtree → BST

O(N)

Iterative Preorder (Root L R)

(Stack)



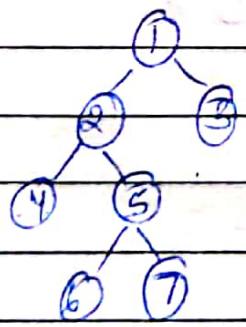
{ opp order

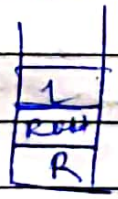
push root

remove after pushing L & R

O(N)

Iterative Inorder (L Root R)





~~root left~~ right

root

root → ~~right~~ left

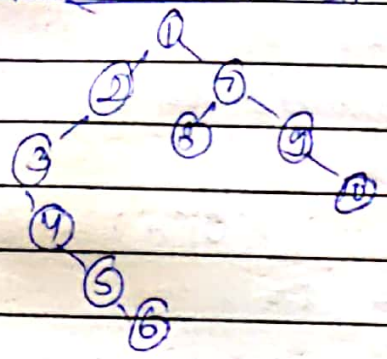
4 2 6 5 7 1 3

Scanned with CamScanner

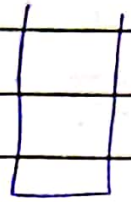
O(2N)

Iterative Postorder

L R Root



left → +
right → +
print



6 ~~8~~ 6 5 4 3 2 8 10 9 7 1

Minimum depth of a BT

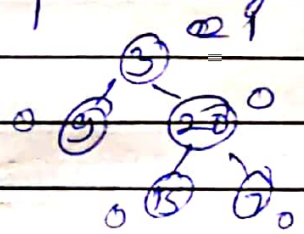
~~BFS~~

Recursive Way.

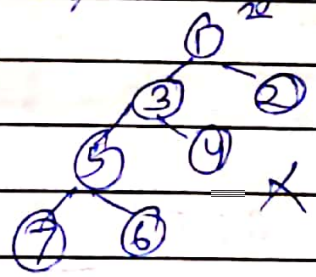
```
// BC /
if (root == Null)   return 0;
int lf = func(root -> left)
int rt = func(root -> right)
return (1+ min(lf, rt));
```

Balanced Binary Tree

for every node →, height (left) − height (right) <= 1



BBT ✓



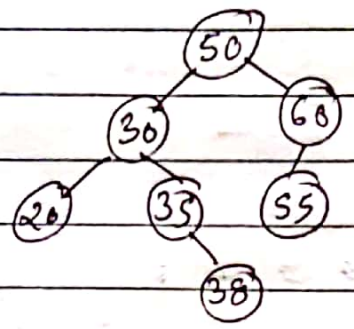
(5−1)

✗ BBT

23.09.2022
Friday

Date: / / Page no:

BINARY SEARCH TREE

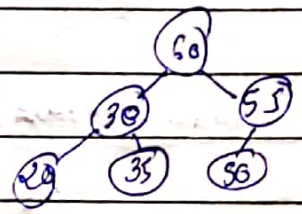


Inorder Traversal of BST gives elements in Sorted order

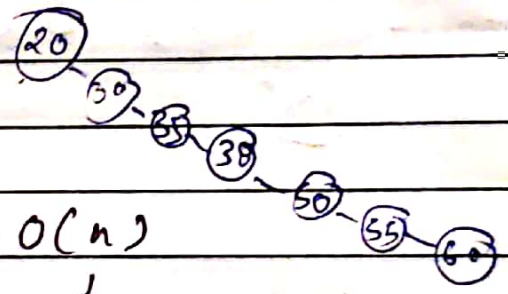
Inorder Traversal → $O(n)$

20, 30, 35, 50, 55, 60

60, 30, 20, 55, 50, 35, 38



20, 30, 35, 38, 50, 55, 60



Construction of BST = $O(n*h) + O(n)$

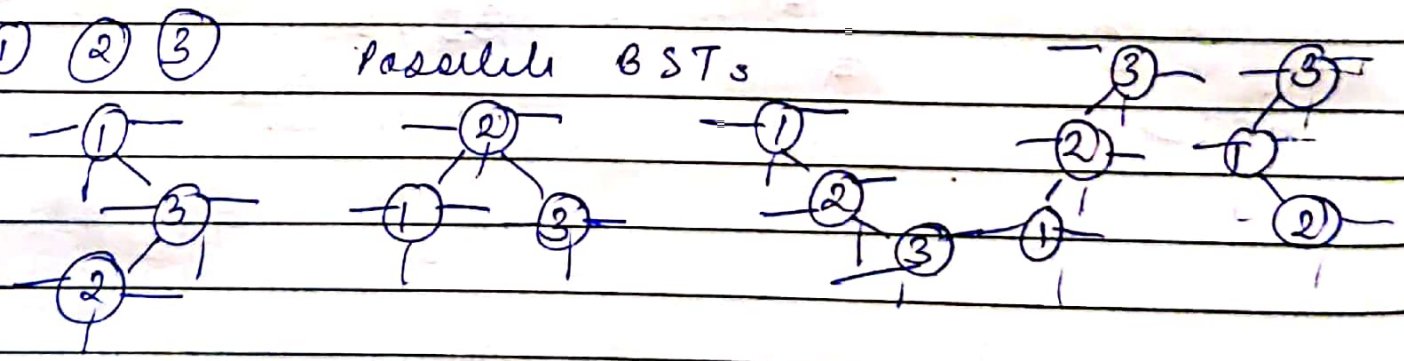
h → height, putting in Tree

$O(n)$ → Traversal of array elements

Construction of Balanced BST = $O(n * \log n) + O(n)$

Traversal → $O(n)$

1 2 3 Possible BSTs



Preorder	1 3 2	2 1 3	1 2 3	3 2 1	3 1 2
Inorder	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
Postorder	2 3 1	1 3 2	3 2 1	1 2 3	2 1 3

Scanned with CamScanner

We can construct a unique tree → Preorder Traversal

NOTE: Equivalence of ~~tre~~ BST → Preorder Traversal

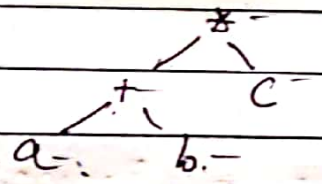
↳ complexity = $O(3n) = n$ postorder traversal

$O(n) + O(n) + O(n)$

Preorder of 1 | Preorder of 2 | ~~Preorder of 3.~~ comparing vals

Application of Binary Search Tree

ARITHMETIC EXPRESSION EVALUATION



Postorder

a b + c − (Postfix)

~~Inorder~~ Preorder

* + a b c (Prefix)

In Preorder

a + b * c (Infix)

Fully Parenthesis Infix :

```
Infix(t)
{
  if (t) ≠ NULL {
    print '('
    Infix( t → left child)
    print t → data
    Infix ( t → right child)
    print ')'
  }
};
```

(((a) + (b)) * (c))

Scanned with CamScanner

24.09.2022
Saturday

Date: / / Page no:

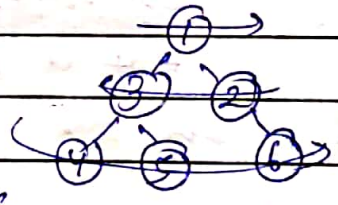
Counting No. of nodes
add count ++ after print
by any traversal technique.

① Diameter/width of a Binary Tree = longest path betn 2 node
(does not pass ~~unit~~ by root ~~OR~~)
logic of ht. of tree. (dfs)

② Maximum Path Sum.
ht. of tree. (dfs)

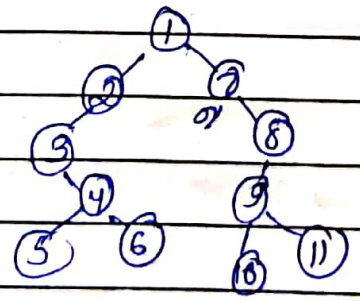
③ Two Trees are identical or not
Any. Traversal

④ Zig-Zag Traversal



BFS 1 2 3 ~~5~~ 4 5 6
flag = Left To Right

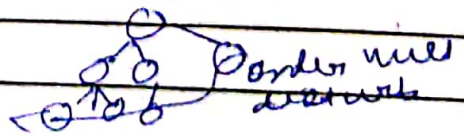
⑤ Boundary Traversal



① Left Boundary excluding leaf
② Leaf Node (Inorder)
③ Right Boundary excluding leaf (Reverse order)

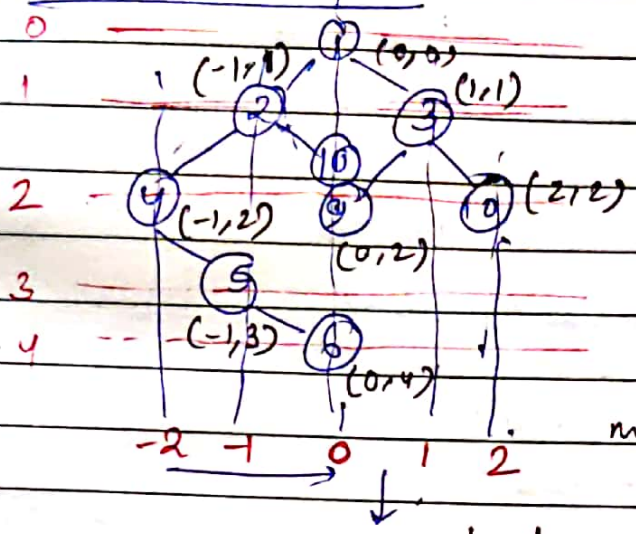
Left
~~1,2,3,4,5,6,7,8,9,10,11~~
1,2,3,4,5,6,10,11,9,8,7,1
Acw

Not BFS



Date: / / Page no:

⑥ Vertical Order Traversal



4
2 5
1 9 10 6
3
10.

smaller value ahead

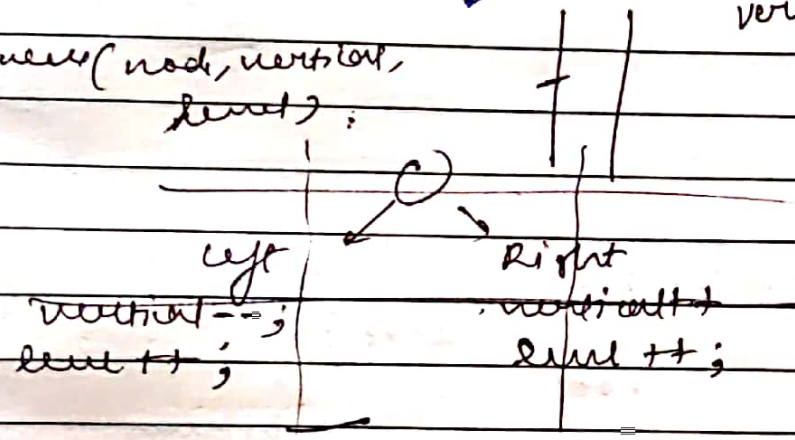
map<int, map<int, ~~multimap<int, int~~ multiset<int>>>

vertical — level — ~~multiple nodes~~ on same level

10 & 9 & they can be diff

∴ multiset & not set.

Queue(node, vertical, level);



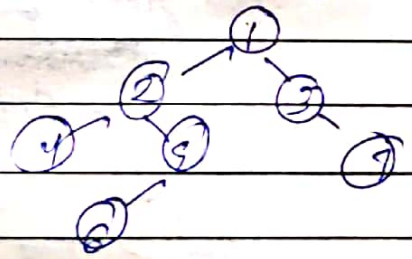
left
vertical--;
level++;

Right
vertical++
level++;

⑦ Top View of Binary Tree:

4 2 1 3 7

1st Node on any vertical

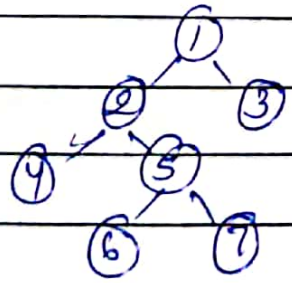


⑧ Root to Node Path

Node = 7

Inorder Traversal array

1, 2, 5, 7



Scanned with CamScanner

27.09.2022
Tuesday

Date: / / Page no:

Lowest Common Ancestor (LCA)

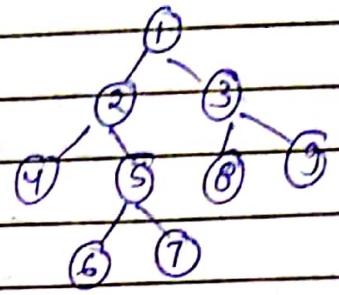
lca (4,7) = 2

lca (5,9) = 1

lca (2,6) = 2

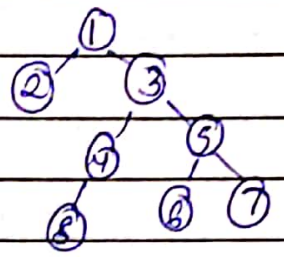
deepest ancestor

5, 4, 1



~~Inorder traversal~~

Path problem



lca (7,8) = 3

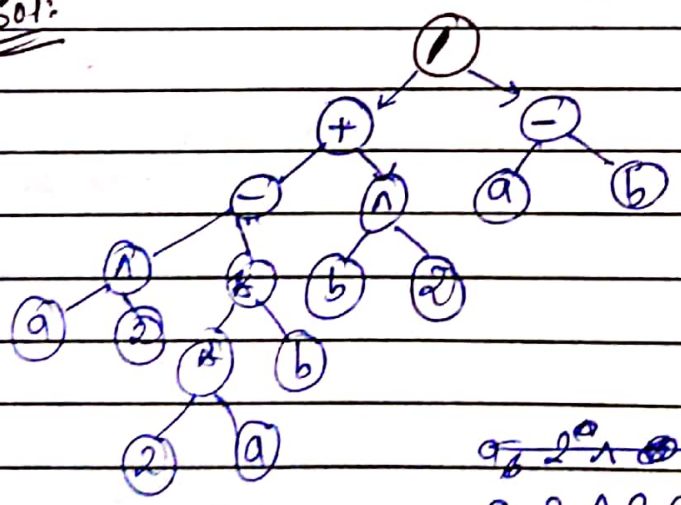
lca (

Q) a 2 ^ 2 a * b * - b 2 ^ + a b - | (Postorder).

(Postfix)

root

Sol:

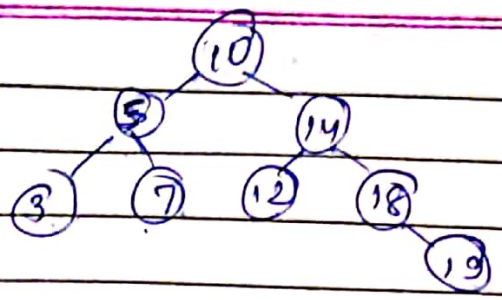


Whenever we find operator the first operand after operand will be left others in both will be right

~~a 2 ^ 2~~

a 2 ^ 2 a * b * - b 2 ^ + a b - |

Postorder - L R



[3, 7, 5] | [12, 19, 18, 14] | [10]

root

Scan & find 1st value smaller than root.

1st part → Right

2nd part → Left

[3, 7, 5]

It's my root

Tree

[12, 19, 18, 14]

It's my root

Tree

3 < 5.

Left → 3

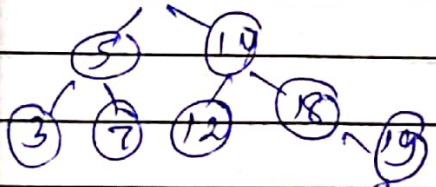
7, → Rt

Similar Algo

12 < 14

12 → Left

Others Right



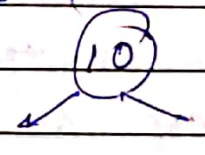
[19 | 18]

root

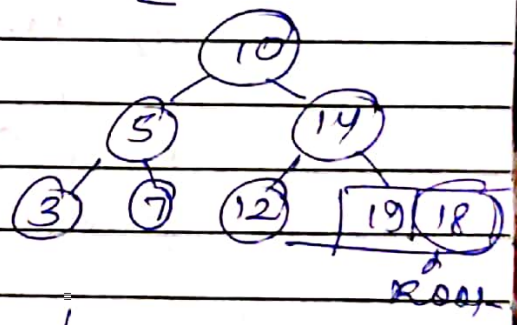
No element smaller than 18

Left = NULL

Right = 19



1 2



Scanned with CamScanner

Date: without recursion Page no:

∴ our own stack.

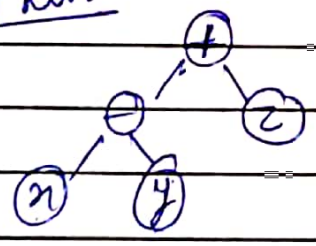
(Iterative) Preorder Traversal.

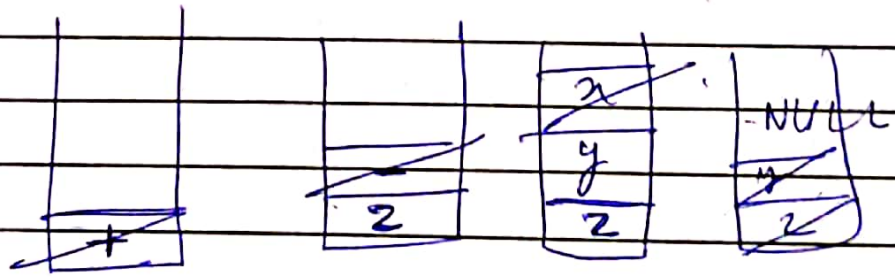
Method: Stack

```
Preorder_Traversal (TreNode * T){
    stack {
        TreNode * N;
        Initialise_Stack (&S);
        push (T, & S);
    while (Not Empty(& S)) {
            pop (& S, &N);
            if (N ≠ NULL) {
                print 'N -> data
            push (N- Rlink , & S):
            push (N - Llink , & S); }
    }
```

// Last in first out

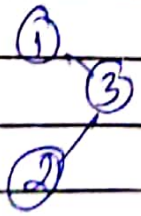
Dry Run

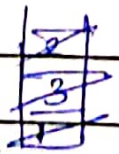




pointer to nodes
(but for ease we will write direct data)

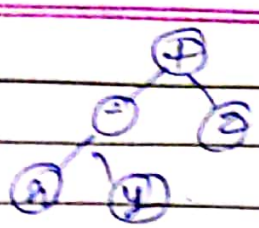
$\boxed{+ - x\ y\ 2}$





1 3 2

Scanned with CamScanner



+	z	-	y	x	

+ z - y z x

+	-	z	x	y

+ - z x y

~~Traver~~ BREADTH FIRST SEARCH (BFS)

Level Order Traversal

Method : Queue (BFS)

```
Traversal (TreNode * T) {
  Queue Q ;        TreNode * T;
  Initialise Queue (&Q);
  Insert (T, &Q);
  while (! Empty (&Q)) {
        Remove (&Q, &N);

  if (N ≠ NULL) {
        print (N -> Data);
        Insert (N->Link, &Q)
        Insert (N->Rlink, &Q);
  } }
}
```

+	-	z	x	y	NULL	NULL

+ - z x y

N

+	-	z	x	y

+	-	z	

N

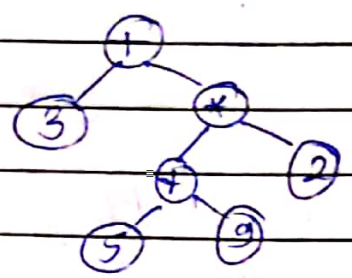
+ -

N

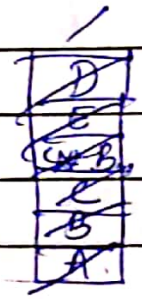
Scanned with CamScanner

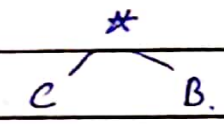
Expression Tree (Stack)

3 + ((5 + 9) * 2)



A B C * + D /


D
E
C*B
C
B
A

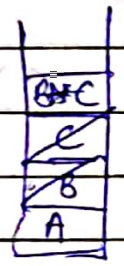


→

~~B*~~

C ~~* B + A / D~~

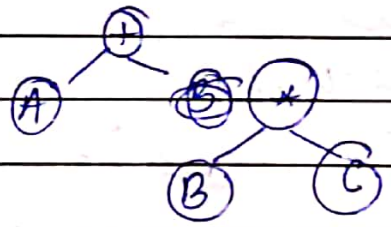
E



~~B * C + A =~~

A * B + C / D

A + B * C →



Left Root Right

Inorder

A + B * C

Leaves | External Nodes = operands

internal Nodes = operator

M.1

Infix → BT

M.2 → Postfix

^ > (*, /) > (+, -)

R-L L-R L-R

Root evaluates at last

(check priority & associativity)

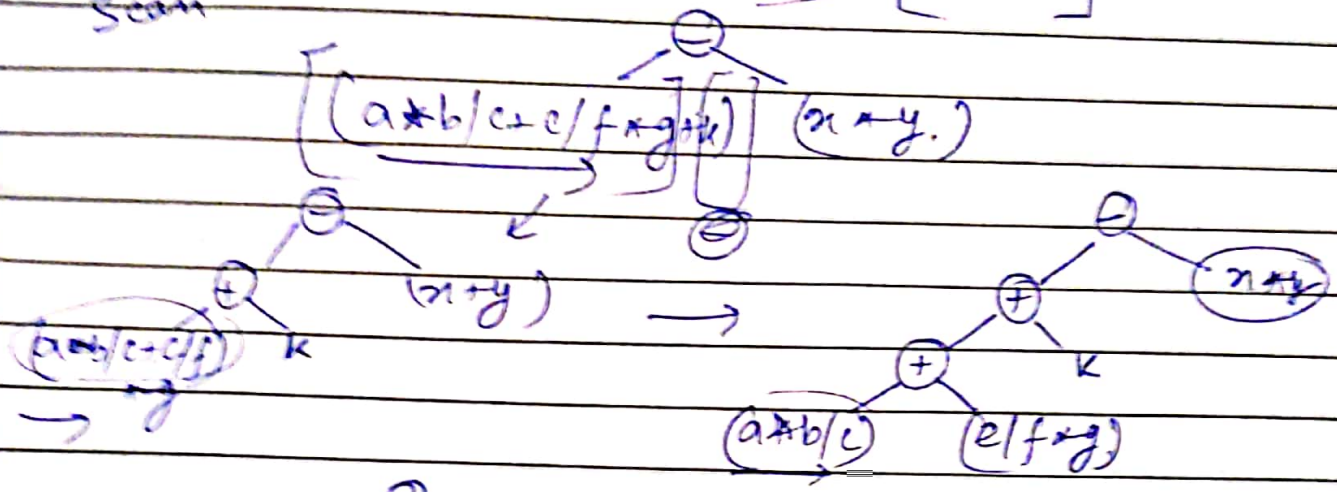
Scanned with CamScanner

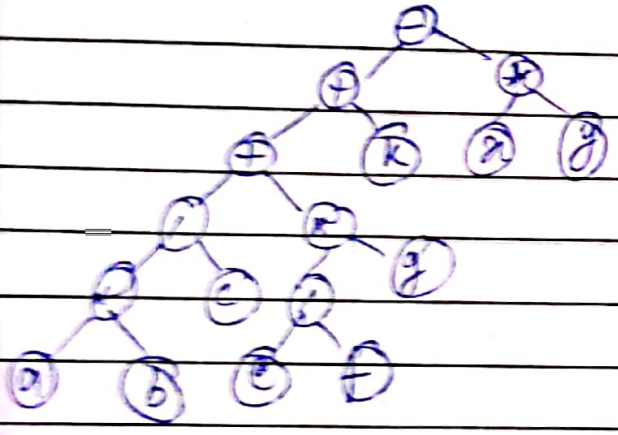
LR Associativity
∴ दोनों बराबर

Date: / / Page no:

Q) $[a * b / c + e / f * g + k] - [x * y]$

Scan





Preorder (R'LR)
- + + / * a b c * / e f g k *
x y

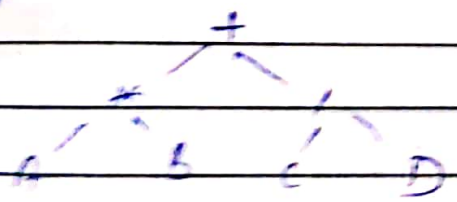
Prefix

Postorder (LRR')
a b * c / e f / g * + k + x y * -

Postfix

Inorder (LR'R)
a * b / c + e / f * g + k - x * y

Q) A*B+C/D : Infix



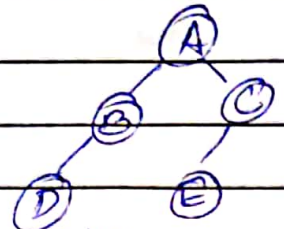
Prefix : Preorder
+ * A B / C D

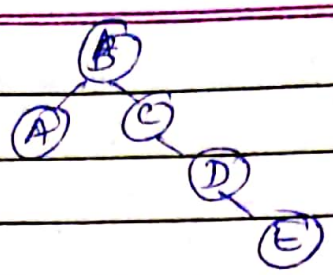
Postfix : Postorder
A B * C D / +

Scanned with CamScanner

29/09/2024 Thursday

Date: / / Page no:





For Binary search Tree we require either Prefix or Postfix only.

Postfix

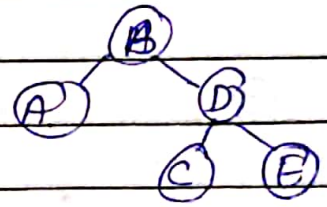
D B E C A

A E D C B

Infix

D B A E C

To construct a Binary Tree we need (Infix & Postfix) or (Infix & Prefix) ~~to~~ :



Postfix ~~at~~ End → Root

D B E C A

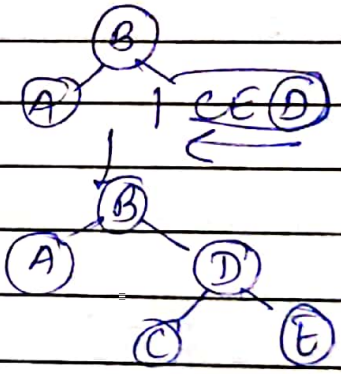
A C E D B

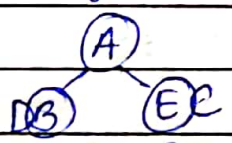
← root

check in Infix.

D B A E C

Left Right





D B E C A

root root

check in Infix.

~~inorder~~

BINARY HEAP

Priority Queue → using queues.

Delete

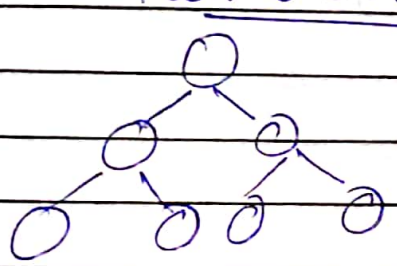
8	7	6	5	1

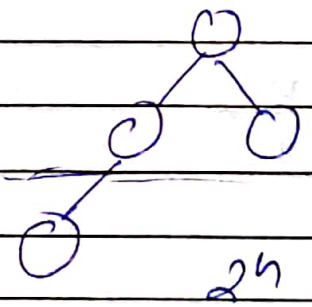
costly

Insertion = $O(n)$

Deletion = $O(1)$

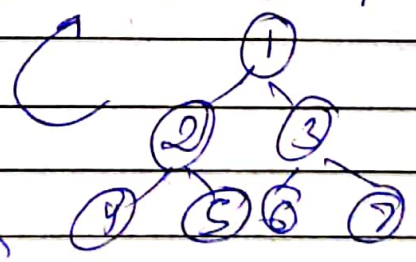
Full Binary Tree is Complete Binary Tree full upto $n-1$





Complete Binary Tree of 2^h to $2^{h+1}-1$

1	2	3	4	5	6	7
1	2	3	4	5	6	7



movement from child to parent is a very fast and in of time

Array Representation better in CBT

No space wastage.

If $(index)_{parent} = p$

$\Rightarrow (index)_{left\ child} = 2*p$

& $(index)_{right\ child} = 2*p+1$

If $(index)_{child} = i$

$\Rightarrow (index)_{parent} = \lfloor i/2 \rfloor$

Scanned with CamScanner

Binary Heap

(1) Complement Binary Tree : Structural property
(2) stored as an Array

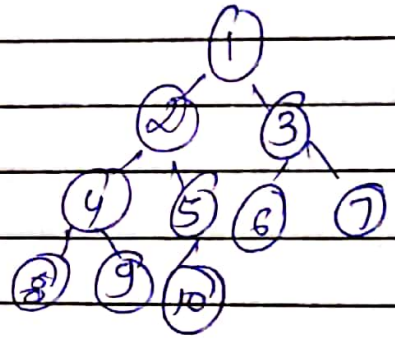
Height of Binary Tree

No. of nodes, $n \le 2^{h+1} - 1$

$\Rightarrow \boxed{h \ge \log(n+1) - 1}$

$$\boxed{h = \log(n+1) - 1}$$

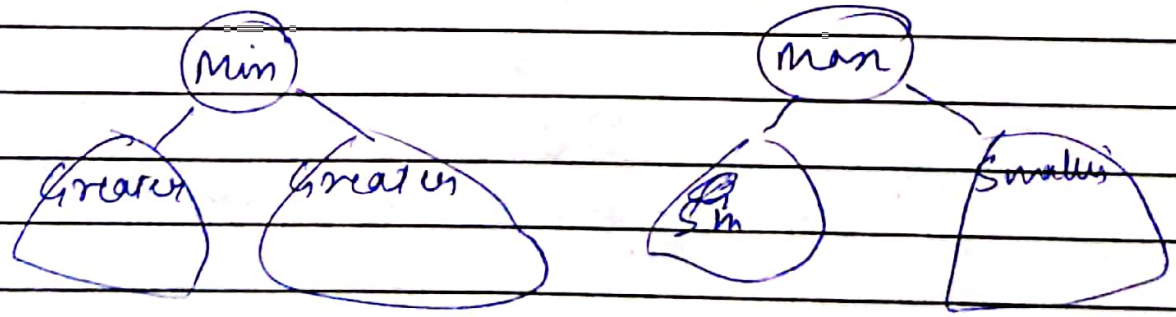
or $\boxed{h = \log n}$



Traversing any one path from root to leaf
$= O(\log n)$
$(= O(h))$

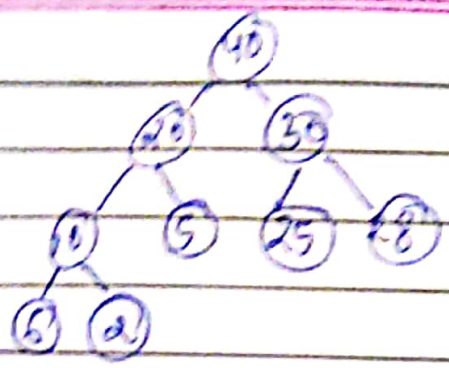
and

Min Heap & Max Heap



Chalk
(1) Structural property — CBT
(2) Ordering property — Ordering
Max Heap / Min Heap

Scanned with CamScanner



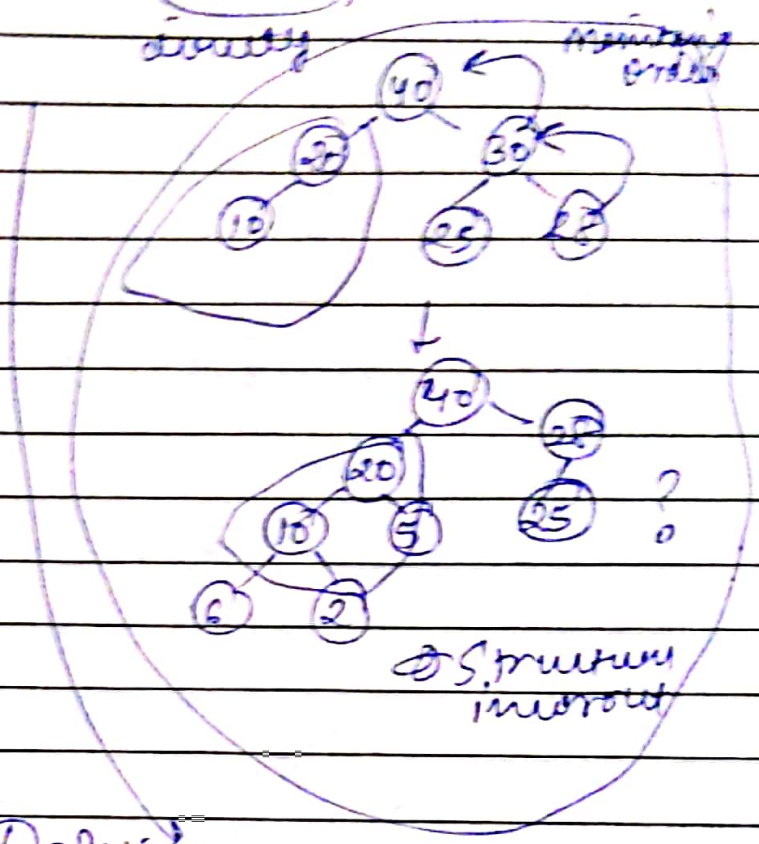
Max Heap

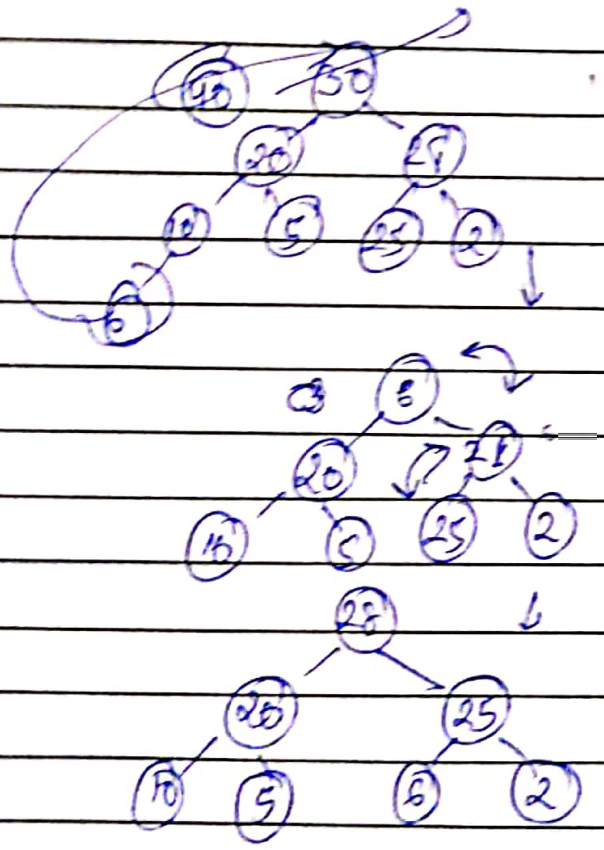
Structure ✓

Order ✓

Deletion

Heapify





Deletion

① We find last element
↳ O(1) → array

② swap with root & delete last element O(1) → array

③ compare & maintain order → O(log n)
(height proportional)

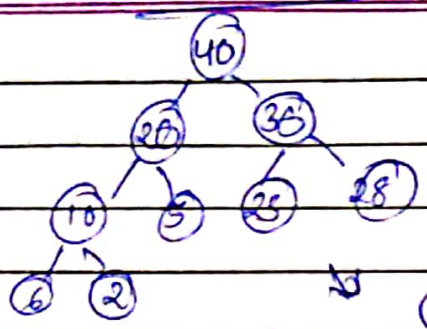
Deletion max	→ O(log n)
Insertion	→ O(log n)

Scanned with CamScanner

$i \rightarrow (i/2)$

Date: / / Page no:

INSERTION

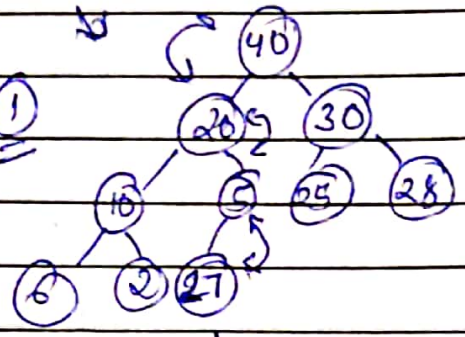


Insert 27

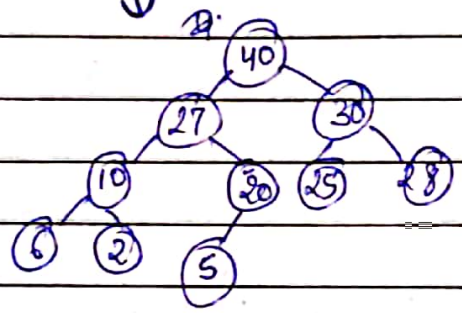
1. Insert at end $\rightarrow O(1)$
2. Maintain order by continuous swapping with parents $\rightarrow O(\log n)$ height

$\rightarrow$ array representation $\therefore$ const. time direct indexing

1



2



Scanned with CamScanner

Date: / / Page no:

Converting Binary Tree from Inorder & Preorder.

inorder = [40 20 50 10 60 30]

Preorder = [10 20 40 50 30 60]

root

10

Inorder 40 20 50 60 30

Preorder 20 40 50 30 60

Scanned with CamScanner