tinyurl.com/uiuportal

powered by UIU Memes

Course: CSE217

Faculty: Nurul Huda

Note courtesy: Shaira Tabassum

Data Structure: It is used to organize data so that data can be accessed easily.

Different types of data structure:

- → Array
- → Linked List
- → File
- → Stack
- → Queue
- → Structure

Sorting Technique:

- ⊛ Insertion sort
- ⊛ Bubble sort
- ⊛ Quick sort
- ⊛ Selection sort
- ⊛ Replacement sort
- ⊛ Merge sort
- ⊛ Heap sort
- ⊛ Radix sort
- ⊛ Bucket sort

Insertion Sort

A	10	8	6	7
	1	2	3	4

$n = 4$

insertion sort always starts from 2nd index.

so, $j = 2$. And $i = j - 1$. So, $i = 1$. $t = A[j]$, so $t = 8$

Condition:

$i >= 1$ AND $A[i] > t$ [for ascending]

$i >= 1$ AND $A[i] < t$ [for descending]

Mechanism:

A	10	8	6	7
	1	2	3	4

→ ascending order

$j = 2$ $t = 8$ $i = j - 1 = 1$

$i >= 1$ AND $A[i] > t$

⇒ $1 >= 1$ AND $A[1] > 8$ = True

A		10	6	7
	1	2	3	4

$i = i - 1$
$= 0$

$0 >= 1$ AND $A[0] > 8$ = False

$i = i + 1$
$= 1$

A	8	10	6	7
	1	2	3	4

$j = 3$ $t = 6$ $i = j - 1 = 2$

$2 >= 1$ AND $A[2] > 6$ = True

A

8		10	7
1	2	3	4

$i = i - 1$
$= 1$

$1 >= 1$ AND $A[1] > 6$ = True

A

	8	10	7
1	2	3	4

$i = i - 1$
$= 0$

$0 >= 1$ AND $A[0] > 6$ = False

$i = i + 1$
$= 1$

A

6	8	10	7
1	2	3	4

$j = 4$ $t = 7$ $i = j - 1 = 3$

$3 >= 1$ AND $A[3] > 7$ = True

A

6	8		10
1	2	3	4

$i = i - 1$
$= 2$

$2 >= 1$ AND $A[2] > 7$ = True

A

6		8	10
1	2	3	4

$i = i - 1$
$= 1$

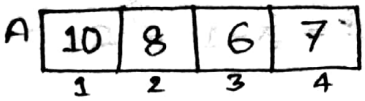
$1 >= 1$ AND $A[1] > 7$ = False

$i = i + 1 = 2$

A

6	7	8	10

Mechanism:



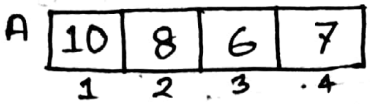
→ descending order

$j = 2$ $\quad t = 8$ $\quad i = j - 1 = 1$

$i >= 1$ AND $A[i] < t$

$\Rightarrow 1 >= 1$ AND $A[1] < 8$ = False

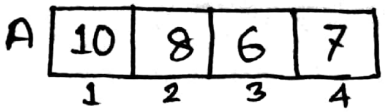
$i = i + 1$
$= 2$



$j = 3$ $\quad t = 6$ $\quad i = j - 1 = 2$

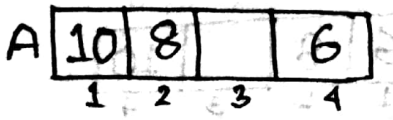
$2 >= 1$ AND $A[2] < 6$ = False

$i = i + 1$
$= 3$



$j = 4$ $\quad t = 7$ $\quad i = j - 1 = 3$

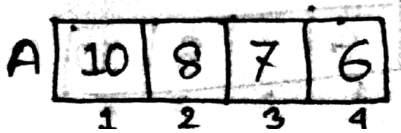
$3 >= 1$ AND $A[3] < 7$ = True



$i = i - 1$
$= 2$

$2 >= 1$ AND $A[2] < 7$ = False

$i = i + 1$
$= 3$



Algorithm:

ascending

```
for j = 2 to n
    t = A[j]
    i = j - 1
    while ((i >= 1) AND (A[i] > t))
        A[i+1] = A[i]
        i = i - 1
    end while
    A[i+1] = t
end for
```

descending

```
for j = 2 to n
    t = A[j]
    i = j - 1
    while ((i >= 1) AND (A[i] < t))
        A[i+1] = A[i]
        i = i - 1
    end while
    A[i+1] = t
end for
```

Bubble Sort

Ascending

A

8	10	9	7
1	2	3	4

$n = 4$

		$A[j] > A[j+1]$	$A[j] \leftrightarrow A[j+1]$
$i=1$	$j=1$	$A[1] > A[2] \Rightarrow F$	x
	$j=2$	$A[2] > A[3] \Rightarrow T$	A: 8 | 9 | 10 | 7 (1 2 3 4)
	$j=3$ ($n-i$)	$A[3] > A[4] \Rightarrow T$	A: 8 | 9 | 7 | 10 (1 2 3 4)
$i=2$	$j=1$	$A[1] > A[2] \Rightarrow F$	x
	$j=2$ ($n-i$)	$A[2] > A[3] \Rightarrow T$	A: 8 | 7 | 9 | 10 (1 2 3 4)
$i=3$	$j=1$ ($n-i$)	$A[1] > A[2] \Rightarrow T$	A: 7 | 8 | 9 | 10 (1 2 3 4)

Descending:

A

8	10	9	7
1	2	3	4

$n = 4$

		$A[j] < A[j+1]$	$A[j] \leftrightarrow A[j+1]$
$i = 1$	$j = 1$	$A[1] < A[2] \Rightarrow T$	A: 10 8 9 7
	$j = 2$	$A[2] < A[3] \Rightarrow T$	A: 10 9 8 7
	$j = 3$	$A[3] < A[4] \Rightarrow F$	×
$i = 2$	$j = 1$	$A[1] < A[2] \Rightarrow F$	×
	$j = 2$	$A[2] < A[3] \Rightarrow F$	×
$i = 3$	$j = 1$	$A[1] < A[2] \Rightarrow F$	×

Algorithm:

ascending

```
for i = 1 to n-1
    for j = 1 to n-i
        if (A[j] > A[j+1])
            A[j] ↔ A[j+1]
        end if
    end for
end for
```

descending

```
for i = 1 to n-1
    for j = 1 to n-i
        if (A[j] < A[j+1])
            A[j] ↔ A[j+1]
        end if
    end for
end for
```

Selection Sort

Ascending:

A

8	10	9	7
1	2	3	4

$n = 4$

$A[k] > A[t]$ $\quad t = k$

$j = 4^{\rightarrow n}$ $\quad t = 1$ $\quad k = 2$ $\quad A[2] > A[1] \Rightarrow T$ $\quad t = 2$

$k = 3$ $\quad A[3] > A[2] \Rightarrow F$ $\quad \times$

$k = 4^{\rightarrow j}$ $\quad A[4] > A[2] \Rightarrow F$ $\quad \times$

$A[t] \leftrightarrow A[j]$

$\Rightarrow A[2] \leftrightarrow A[4]$

A

8	7	9	10
1	2	3	4

$j = 3$ $\quad t = 1$ $\quad k = 2$ $\quad A[2] > A[1] \Rightarrow F$ $\quad \times$

$k = 3^{\rightarrow j}$ $\quad A[3] > A[1] \Rightarrow T$ $\quad t = 3$

$A[3] \leftrightarrow A[3]$

A

8	7	9	10
1	2	3	4

$j = 2$ $\quad t = 1$ $\quad k = 2^{\rightarrow j}$ $\quad A[2] > A[1] \Rightarrow F$ $\quad \times$

$A[1] \leftrightarrow A[2]$

A

7	8	9	10
1	2	3	4

Descending:

8	10	9	7
1	2	3	4

			A[k] < A[t]	t = k
j = 4	t = 1	k = 2	A[2] < A[1] ⇒ F	×
		k = 3	A[3] < A[1] ⇒ F	×
		k = 4	A[4] < A[1] ⇒ T	t = 4

A[t] ↔ A[j]
⇒ A[4] ↔ A[4]

A

8	10	9	7
1	2	3	4

j = 3	t = 1	k = 2	A[2] < A[1] ⇒ F	×
		k = 3	A[3] < A[1] ⇒ F	×

A[1] ↔ A[3]

A

9	10	8	7
1	2	3	4

j = 2	t = 1	k = 2	A[2] < A[1] ⇒ F	×

A[1] ↔ A[2]

A

10	9	8	7
1	2	3	4

Algorithm : Replacement Sort

ascending

```
for j = n  downto  2
    t = 1
    for k = 2 to j
        if (A[k] > A[t])
            t = k
        end if
    end for
    A[t] ↔ A[j]
end for
```

descending

```
for j = n  downto  2
    t = 1
    for k = 2 to j
        if (A[k] < A[t])
            t = k
        end if
    end for
    A[t] ↔ A[j]
end for
```

Replacement Sort

Ascending:

A

10	9	15	8
1	2	3	4

$n = 4$

		$A[i] > A[j]$	$A[i] \leftrightarrow A[j]$
$i=1$	$j = i+1 = 2$	$A[1] > A[2]$ ⇒ true	A: 9 10 15 8
	$j = 3$	$A[1] > A[3]$ ⇒ false	×
	$j = ④^{n}$	$A[1] > A[4]$ ⇒ true	A: 8 10 15 9
$i=2$	$j = i+1 = 3$	$A[2] > A[3]$ ⇒ false	×
	$j = 4$	$A[2] > A[4]$ ⇒ true	A: 8 9 15 10
$i=③^{n-1}$	$j = i+1 = 4$	$A[3] > A[4]$ ⇒ true	A: 8 9 10 15

Descending:

A	10	9	15	8
	1	2	3	4

$n = 4$

		$A[i] < A[j]$	$A[i] \leftrightarrow A[j]$
$i = 1$	$j = i+1 = 2$	$A[1] < A[2]$ ⇒ false	×
	$j = 3$	$A[1] < A[3]$ ⇒ true	A: 15 | 9 | 10 | 8 (1 2 3 4)
	$j = 4$	$A[1] < A[4]$ ⇒ false	×
$i = 2$	$j = i+1 = 3$	$A[2] < A[3]$ ⇒ true	A: 15 | 10 | 9 | 8 (1 2 3 4)
	$j = 4$	$A[2] < A[4]$ ⇒ false	×
$i = 3$	$j = i+1 = 4$	$A[3] < A[4]$ ⇒ false	×

Algorithm:

ascending

```
for i=1 to n-1
    for j=i+1 to n
        if (A[i] > A[j])
            A[i] ↔ A[j]
        end if
    end for
end for
```

descending

```
for i=1 to n-1
    for j=i+1 to n
        if (A[i] < A[j])
            A[i] ↔ A[j]
        end if
    end for
end for
```

Quick Sort

27	8	9	55	70	19	32	18	42	31	23
f=1	~~i=2~~	~~i=3~~	i=4							l=11, j=11

$A[f] \geq A[i]$

$27 \geq 8 \Rightarrow$ true $\rightarrow i=3$

$27 \geq 9 \Rightarrow$ true $\rightarrow i=4$

$27 \geq 55 \Rightarrow$ false $\rightarrow$ x

$A[f] \leq A[j]$

$27 \leq 23 \Rightarrow$ false $\rightarrow$ x

$A[i] \leftrightarrow A[j]$

$\Rightarrow A[4] \leftrightarrow A[11]$

$i = i+1 = 5$ and $j = j-1 = 10$

27	8	9	23	70	19	32	18	42	31	55
f=1			i=5					j=8	~~j=9~~	~~j=10~~ l=11

$A[f] \geq A[i]$

$27 \geq 70 \Rightarrow$ false $\rightarrow$ x

$A[f] \leq A[j]$

$27 \leq 31 \Rightarrow$ true $\rightarrow j=9$

$27 \leq 42 \Rightarrow$ true $\rightarrow j=8$

$27 \leq 18 \Rightarrow$ false $\rightarrow$ x

$A[i] \leftrightarrow A[j]$

$\Rightarrow A[5] \leftrightarrow A[8]$

$i = i+1 = 6$ and $j = j-1 = 7$

27	8	9	23	18	19	32	70	42	31	55
f=1					~~i=6~~ j=6	~~j=7~~ i=7				l=11

$A[i] \geq A[1]$ $\quad$ $A[j] \leq A[1]$

$27 \geq 19 \Rightarrow$ true $\rightarrow i = 7$ $\quad$ $27 \leq 32 \Rightarrow$ true $\rightarrow j = 6$

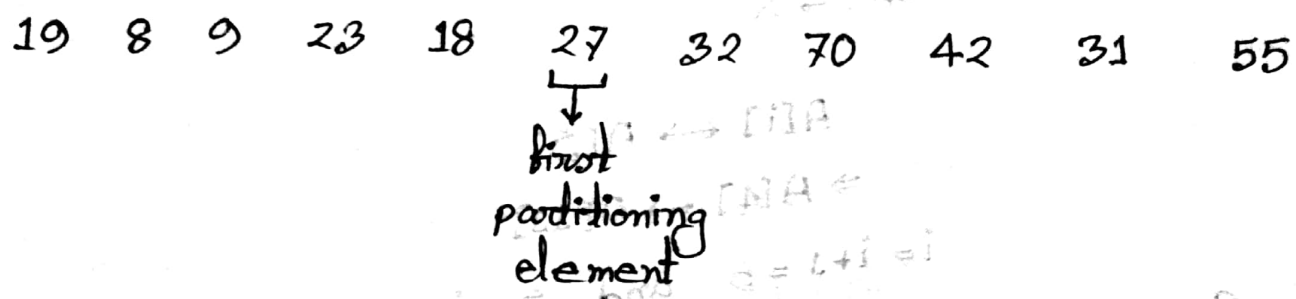
$27 \geq 32 \Rightarrow$ false $\rightarrow$ X $\quad$ $27 \leq 19 \Rightarrow$ false $\rightarrow$ X

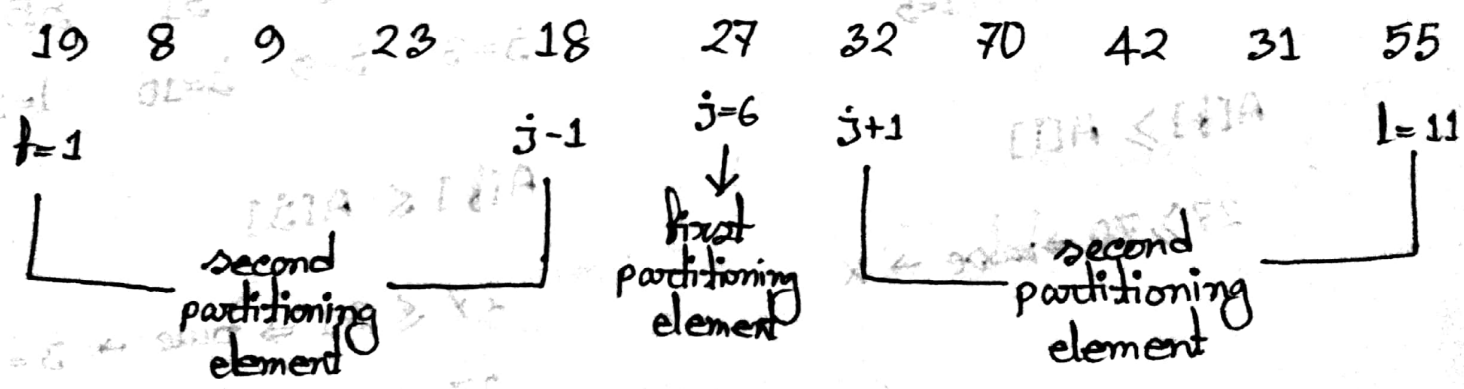
Now, $i = 7$ and $j = 6$

$\because i \geq j$ (break)

$A[1] \leftrightarrow A[j]$

$\Rightarrow A[1] \leftrightarrow A[6]$





19	8	9	23	18
f=1	~~i=2~~	~~i=3~~	i=4	j=5
				l=5

$A[f] \geq A[i]$

$19 \geq 8 \Rightarrow$ true $\rightarrow i=3$

$19 \geq \Rightarrow$ true $\rightarrow i=4$

$19 \geq 23 \Rightarrow$ false $\rightarrow x$

$A[f] \leq A[j]$

$19 \leq 18 \Rightarrow$ false $\rightarrow x$

$A[i] \leftrightarrow A[j]$

$\Rightarrow A[4] \leftrightarrow A[5]$

$i = i+1 = 5$ and $j = j-1 = 4$

19	8	9	18	23
f=1			j=4	l=5
				i=5

Now, $i=5$ and $j=4$

$\because i > j$

$A[f] \leftrightarrow A[j]$

$\Rightarrow A[1] \leftrightarrow A[4]$

18	8	9	19	23

↓

second partitioning element

32	70	42	31	55
f=1	i=2		j=4	l=5
				~~j=5~~

$A[f] \geq A[i]$

$32 \geq 70 \Rightarrow$ false $\rightarrow$ x

$A[f] \leq A[j]$

$32 \leq 55 \Rightarrow$ ~~true~~ $\rightarrow j = 4$

$32 \leq 31 \Rightarrow$ false $\rightarrow$ x

$A[i] \leftrightarrow A[j]$

$\Rightarrow A[2] \leftrightarrow A[4]$

$i = i+1 = 3$ and $j = j-1 = 3$

32	31	42	70	55
f=1	j=2	i=3		l=5
		~~j=3~~		

$A[f] \geq A[i]$

$32 \geq 42 \Rightarrow$ false $\rightarrow$ x

$A[f] \leq A[j]$

$32 \leq 42 \Rightarrow$ ~~true~~ $\rightarrow j = 2$

$32 \leq 31 \Rightarrow$ false $\rightarrow$ x

Now, $i = 3$ and $j = 2$

$\because i >= j$ (break)

$A[f] \leftrightarrow A[j]$

$\Rightarrow A[1] \leftrightarrow A[2]$

31	32	42	70	55

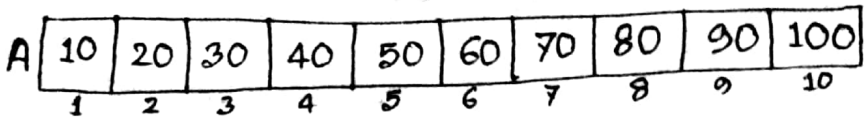
↓

second partitioning element

Algorithm:

```
void Quick (int A[], int f, int l)
{
    int i,j;
    i=f+1;
    j=l;
    if (l>f)
    {
        while (i<j)
        {
            while((A[f] >= A[i]) && i<l)
                i=i+1;
            while((A[f]<= A[j]) && j>=f)
                j=j-1;
            if(i>=j) break;
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i=i+1;
            j = j-1;
        }
        temp = A[f];
        A[f] = A[j];
        A[j] = temp;
        Quick (A, f, j-1);
        Quick (A, j+1, l);
    }
}
```

Binary Search



$n = 10$

⊛ Data should be inserted in ascending, descending or alphabetical order.

key : 70

l	h	mid = $\frac{l+h}{2}$	A[mid] = key	A[mid] > key (changes 'h')	A[mid] < key (changes 'l')
1	10	5	A[5] = 70 ⇒ false	A[5] > 70 ⇒ false	A[5] < 70 ⇒ true
mid+1 6	10	8	A[8] = 70 ⇒ false	A[8] > 70 ⇒ true	x
6	mid-1 7	6	A[6] = 70 ⇒ false	A[6] > 70 ⇒ false	A[6] < 70 ⇒ true
mid+1 7	7	7	A[7] = 70 ⇒ true	x	x

⇓

'Found'
position : 7

key : 35

l	h	mid = $\frac{l+h}{2}$	A[mid] = key	A[mid] > key	A[mid] < key
1	10	5	A[5] = 35 ⇒ false	A[5] > 35 ⇒ true	×
1	4	2	A[2] = 35 ⇒ false	A[2] > 35 ⇒ false	A[2] < 35 ⇒ true
3	4	3	A[3] = 35 ⇒ false	A[3] > 35 ⇒ false	A[3] < 35 ⇒ true
4	4	4	A[4] = 35 ⇒ false	A[4] > 35 ⇒ true	×
4	3				

Since, l > h

'Not Found"

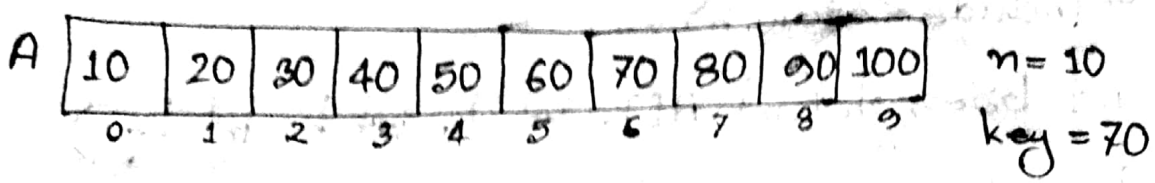
Algorithm (Iterative):

```
l = 1
h = n
while (l<=h)
    mid = ⌊(l+h)/2⌋
    if (A[mid] = key)
        print*, "Found", mid, exit loop
    else if (A[mid] > key)
        h = mid - 1
    else if (A[mid] < key)
        l = mid + 1
end while
if (l>h)
    print*, "Not Found"
end if
```

Algorithm (Recursive):

```
#include<stdio.h>
int BSearch(int A[], int l, int h, int key)
{
    int mid;
    if(l>h) return -1;
    mid=(l+h)/2;
    if(A[mid]==key) return mid;
    else if(A[mid]>key) return BSearch(A, l, mid-1, key);
    else return BSearch(A, mid+1, h, key);
}

int main()
{
    // Declare variable for n, array A, flag and key
    // Read n and data for array A
    // Read search key
    flag = BSearch(A, 0, n-1, key);
    if(flag==-1)
        printf("Not Found");
    else
        printf("Found at %d", flag);
    return 0;
}
```

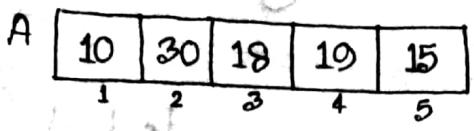
Recursion Tree:



flag = BSearch(A, 0, 9, 70) → 6

- if(0>9) false
- mid = $\frac{0+9}{2}$ = 4
- if(A[4]==70) false
- else if(A[mid]>70) false
- else true → return BSearch(A, 5, 9, 70) → 6
  - if(5>9) false
  - mid = $\frac{5+9}{2}$ = 7
  - if(A[7]==70) false
  - else if(A[7]>70) true → return BSearch(A, 5, 6, 70) → 6
    - if(5>6) false
    - mid = $\frac{5+6}{2}$ = 5
    - if(A[5]==70) false
    - else if(A[5]>70) false
    - else true → return BSearch(A, 6, 6, 70) → 6
      - if(6>6) false
      - mid = $\frac{6+6}{2}$ = 6
      - if(A[6]==70) return mid → 6
      - X
      - X
  - else X
- else X

Linear Search



$n = 5$

key = 19, 17

i	A[i] = key
1	A[1] = 19 ⇒ F
2	A[2] = 19 ⇒ F
3	A[3] = 19 ⇒ F
4	A[4] = 19 ⇒ T
5	X

'Found' → print 'i'

i	A[i] = key
1	A[1] = 17 ⇒ F
2	A[2] = 17 ⇒ F
3	A[3] = 17 ⇒ F
4	A[4] = 17 ⇒ F
5	A[5] = 17 ⇒ F
6	↳ i > n, 'Not Found'

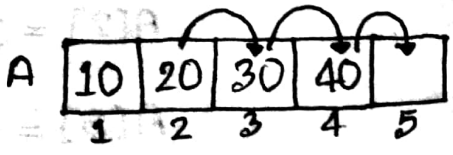
Algorithm:

```
for i = 1 to n
    if (A[i] = key)
        print*, "Found", i
        exit loop
    end if
end for
if (i > n)
    print*, "Not Found"
```

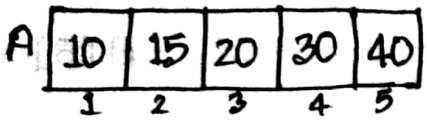
Distinguish between Array and Linked List:

Array

1. Insertion difficult.

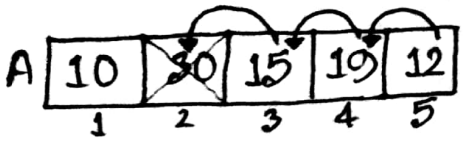


item = 15



Data movement : 3

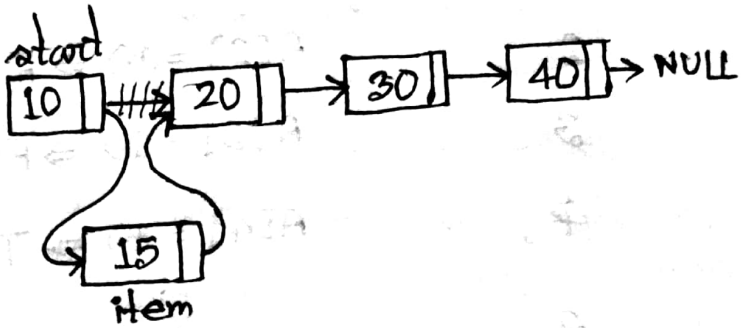
2. Deletion difficult.



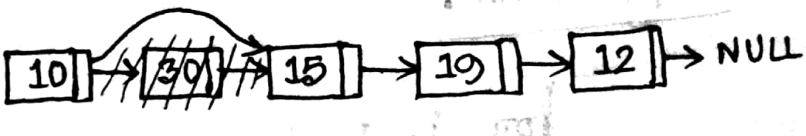
3. Binary search easy.
(index available)

Linked List

1. Insertion easy.

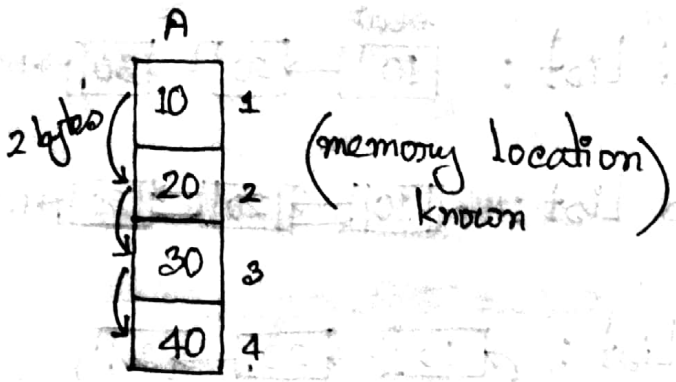


2. Deletion easy.



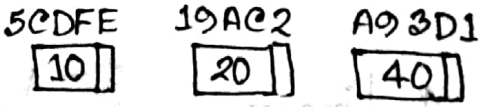
3. Binary search difficult.
(no index)

4. Sequential memory allocation.



$loc(A[4]) = loc(A[1]) + 2 \text{ bytes} + 2 \text{ bytes} + 2 \text{ bytes}$

4. Random memory allocation.



5. Static memory allocation.

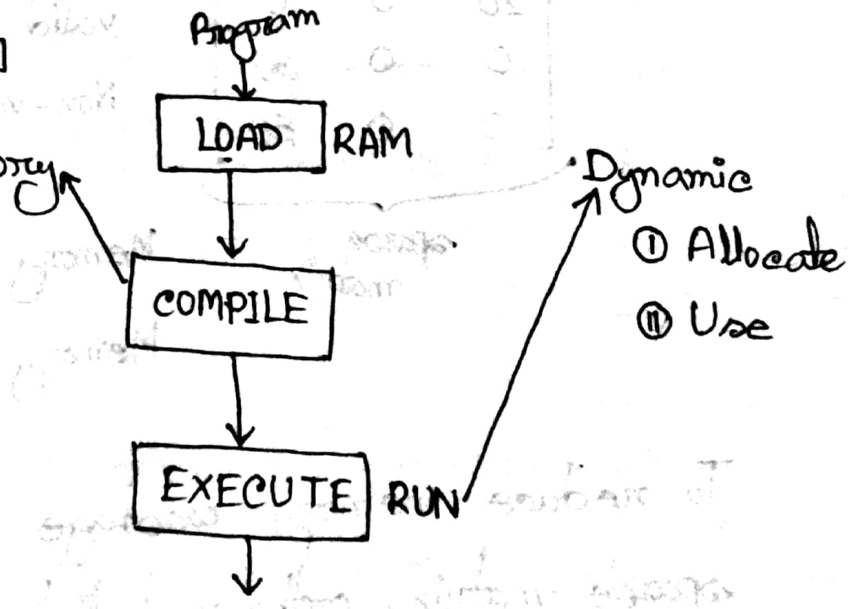
Array: A[10]

Static allocated memory
= 10 * 2 bytes
= 20 bytes

Used memory
= 3 * 2 bytes
= 6 bytes

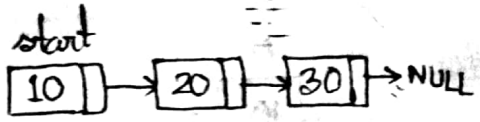
wastage = (20 - 6)
= 14 bytes

5. Dynamic memory allocation.

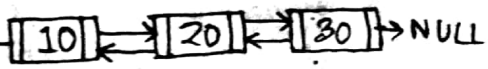


Different types of Linked List :

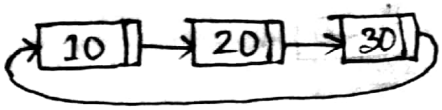
1. Linear or Singular Linked List :



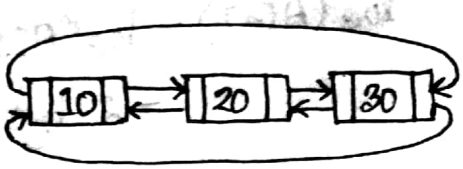
2. Double or Two way Linked List :



3. Linear Circular Linked List :



4. Two way circular Linked List:



5. Orthogonal Linked List :

$$\underbrace{\begin{bmatrix} 20 & 0 & 0 \\ 0 & 0 & 30 \\ 0 & 0 & 50 \end{bmatrix}}_{\text{sparse matrix}}$$

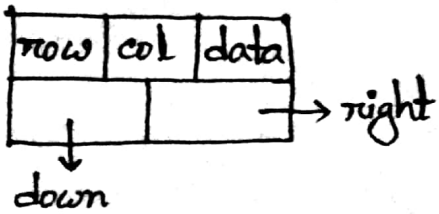
valid data = 3

Non-valid data = 6 (zero)

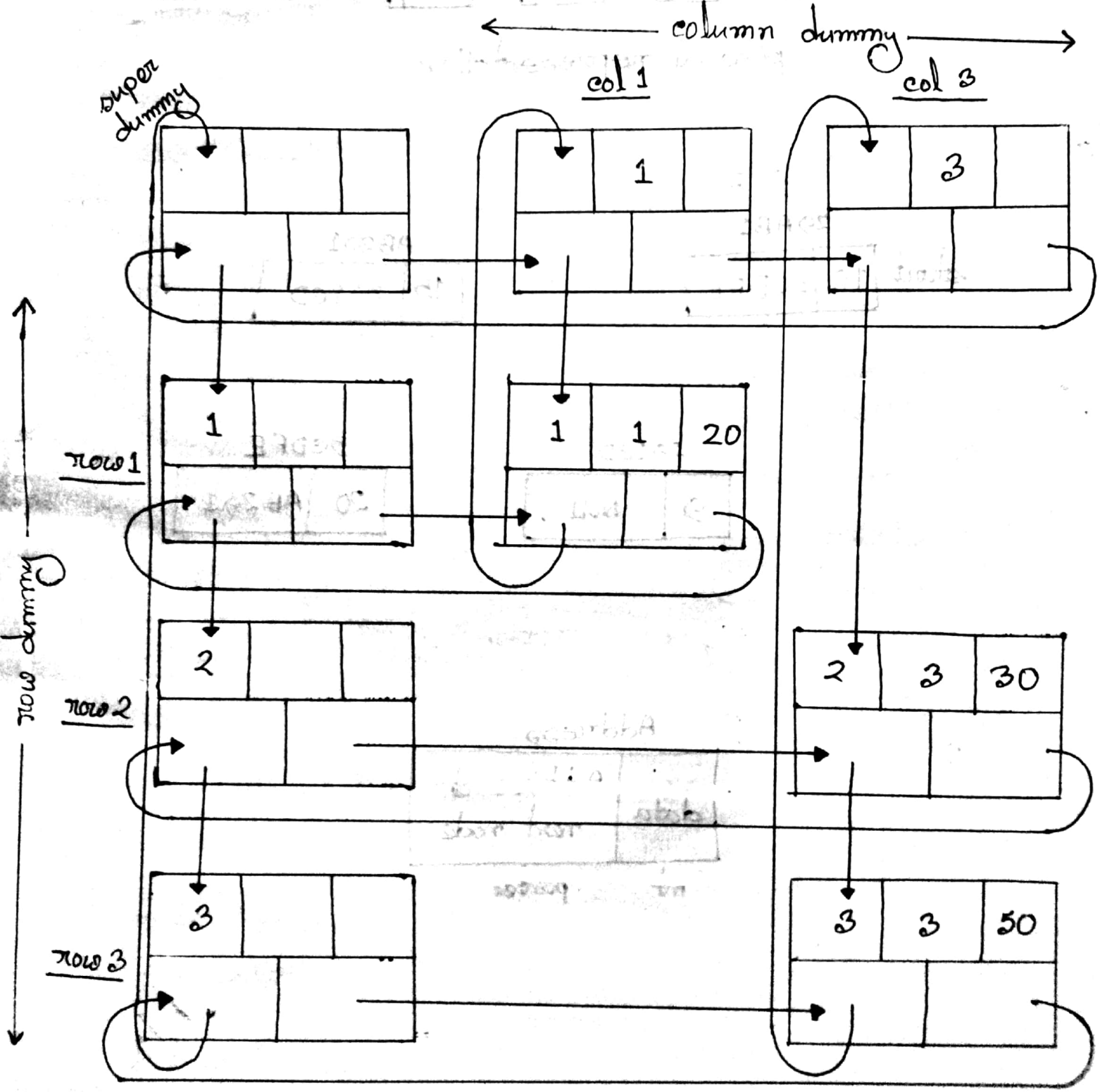
Memory for valid data = 3*2 bytes

Memory for non-valid data = $\underbrace{6*2}_{\text{wastage}}$ bytes

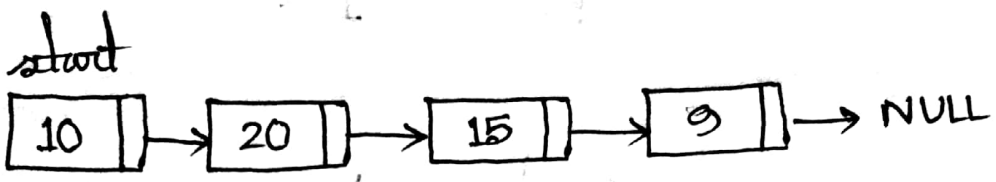
To reduce memory wastage in sparse matrix, orthogonal linked list is used



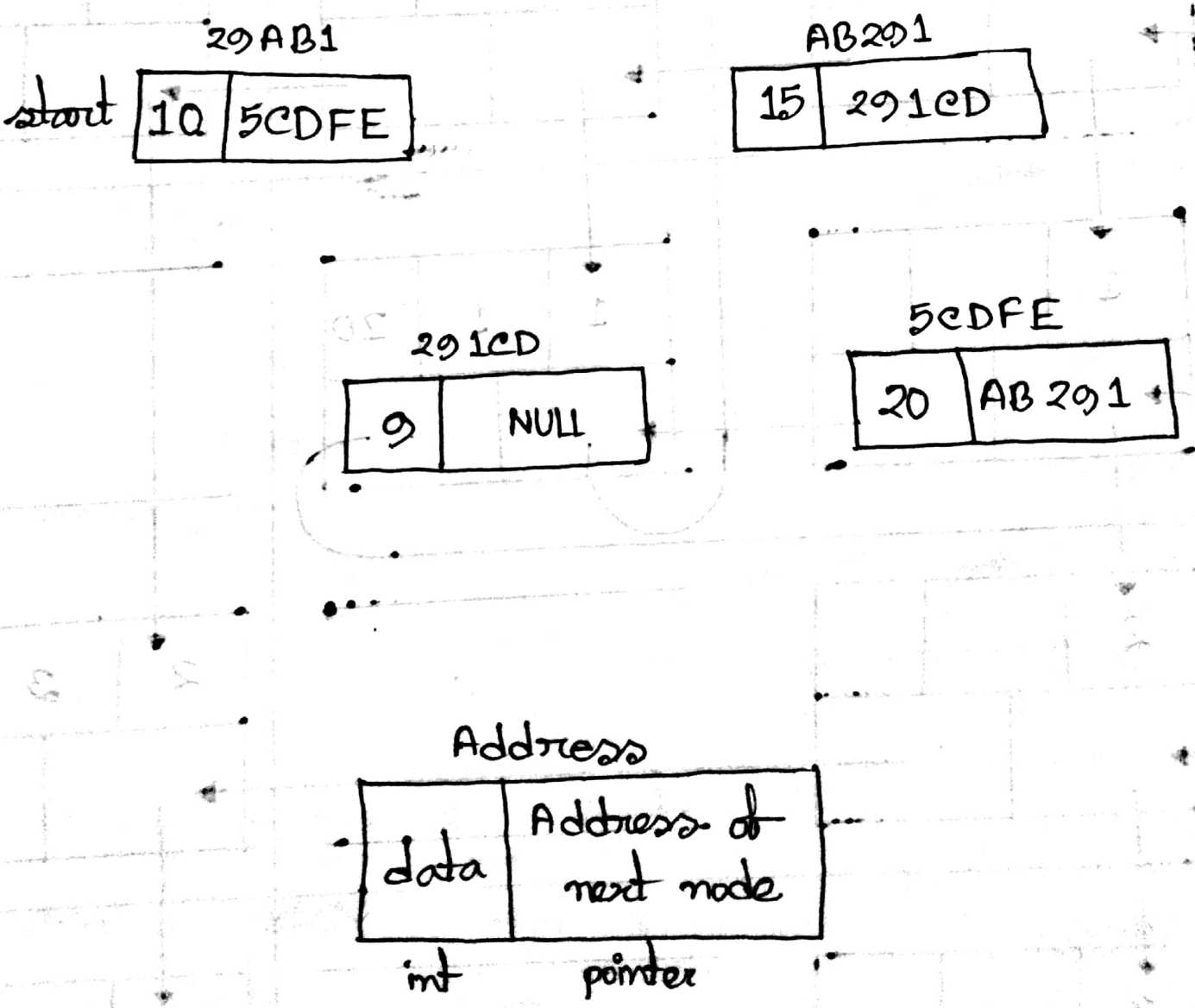
	col1	col2	col3
row1	20	0	0
row2	0	0	30
row3	0	0	50



Actual memory representation for linear linked list :

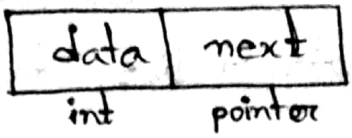


pictorial representation



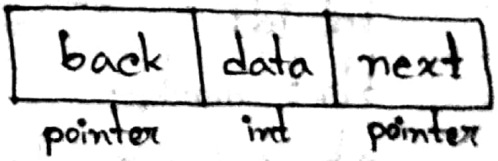
Variable declaration:

```
struct list
{
   int data;
   struct list *next;
};

typedef struct list node;
int main()
{
  node *start;
  ..........
  ..........
   return 0;
}
```



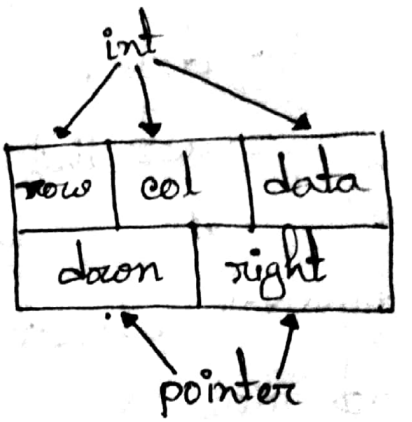
Double:

```
struct list
{
   struct list *back, *next;
   int data;
};

typedef struct list node;
int main()
{
  node *head;
   ..........
   ..........
   return 0;
}
```



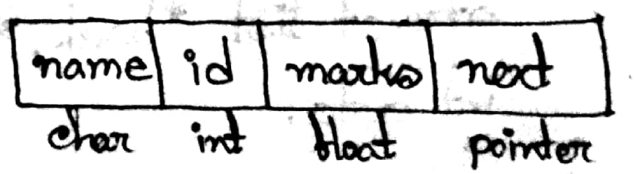
Orthogonal linked list:

```
struct list
{
    int row, col, data;
    struct list *down, *right;
};
typedef struct list node;
int main()
{
    node *super;
    ..........
    ..........
    return 0;
}
```



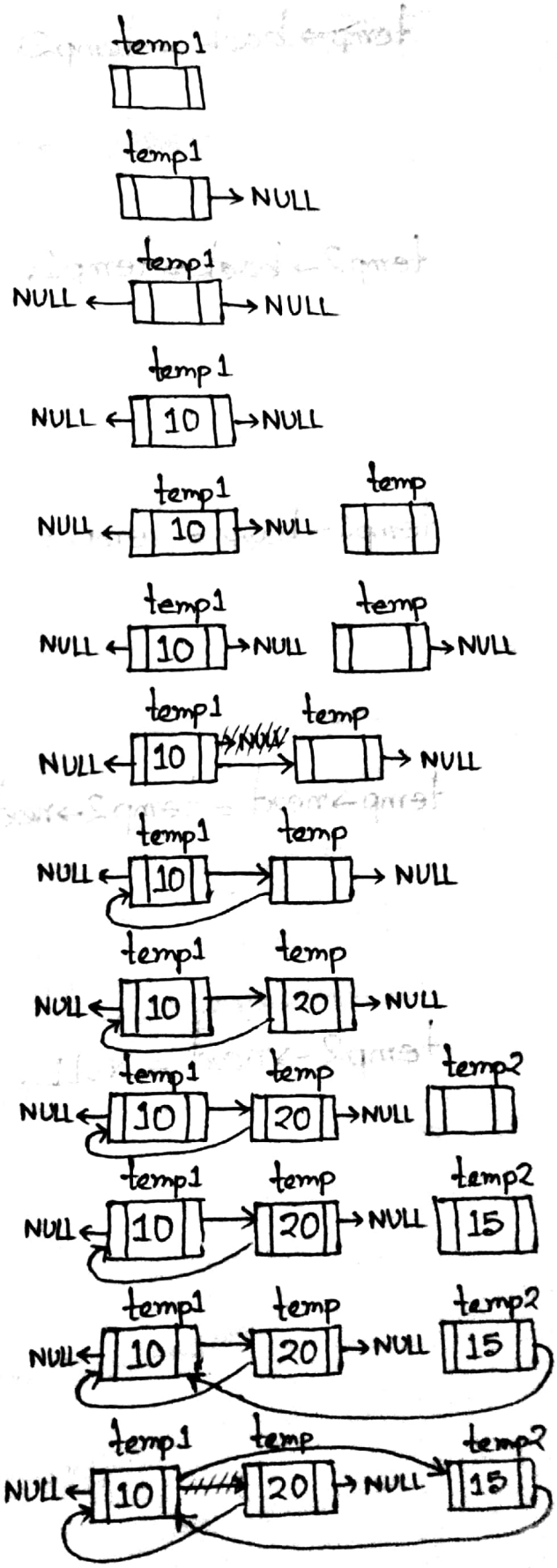
Multiple field:

```
struct list
{
    char name[30];
    int id;
    float marks;
    struct list *next;
};
typedef struct list node;
int main()
{
    node *temp;
    ..........
    ..........
    return 0;
}
```

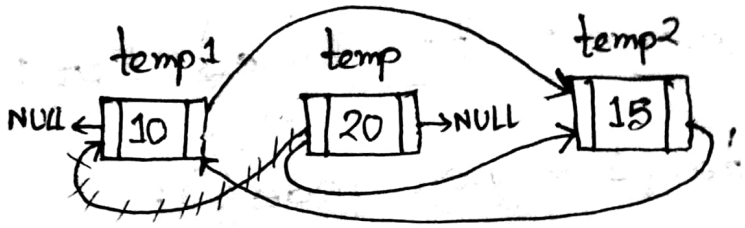


Statement for linked list :

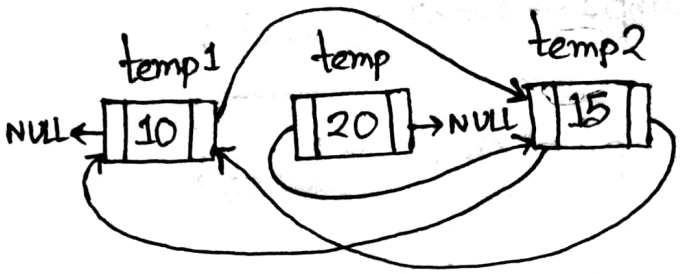
```
temp1 = new node();
temp1->next = NULL;
temp1->back = NULL;
temp1->data = 10;
temp = new node();
temp->next = NULL;
temp1->next = temp;
temp->back = temp1;
temp->data = 20;
temp2 = new node();
temp2->data = 15;
temp2->next = temp1->next;
temp1->next = temp2;
```



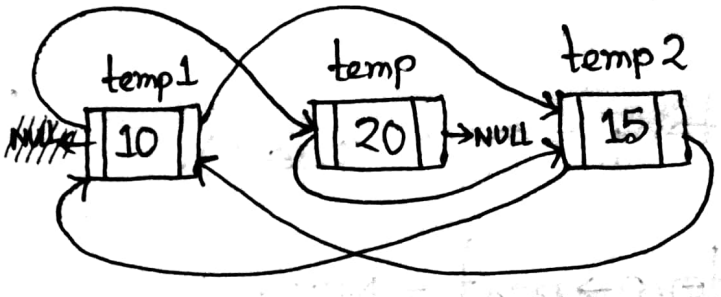
temp->back = temp2;



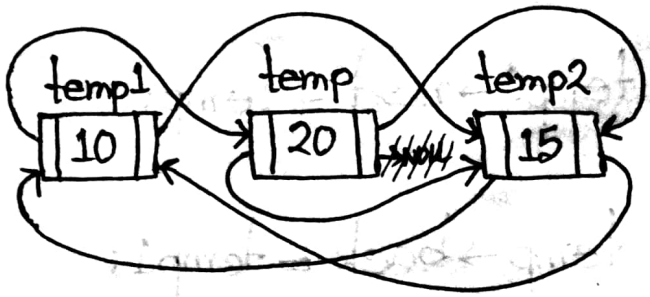
temp2->back = temp1;



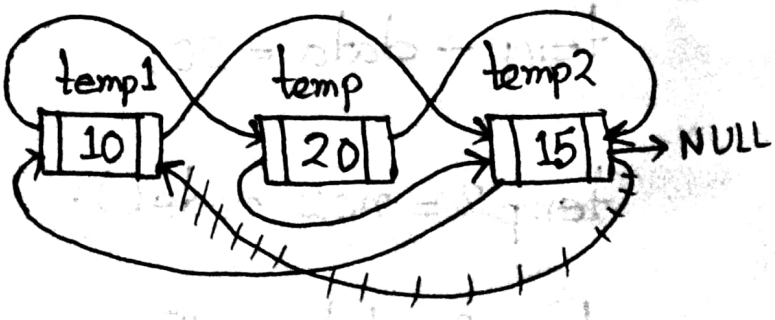
temp1->back = temp;



temp->next = temp2->next;



temp2->next = NULL;



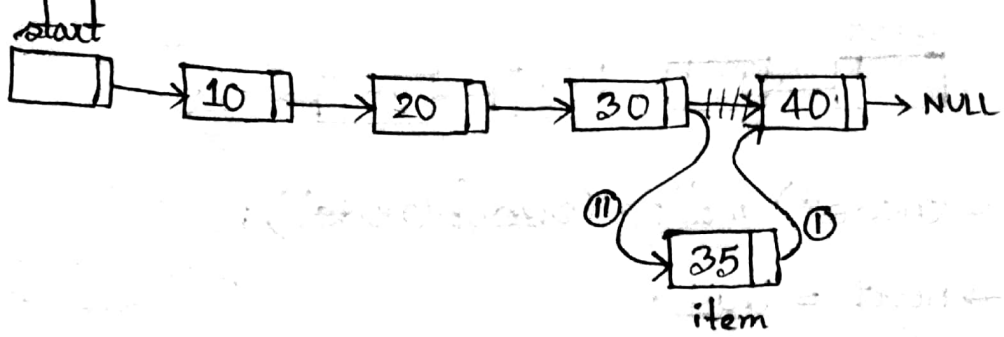
Linear Linked List Creation :


```
start = (node*) malloc (sizeof(node));
start->next = NULL;
prev = start;
for (int i= 10;  i<= 30;  i= i+ 10)
{
    temp = (node*) malloc (sizeof(node));
    temp->data = i;
    temp->next = NULL;
    prev->next = temp;
    prev = temp;
}
```

Display :

```
temp = start->next;
while (temp != NULL)
{
    printf ("%d", temp->data);
    temp = temp->next;
}
```

Insertion in a linear linked list :




```
temp = start;
while((temp->next != NULL) && (temp->next->data <= item))
    temp = temp->next;
temp1 = (node*) malloc(sizeof(node));
temp1->data = item;
```

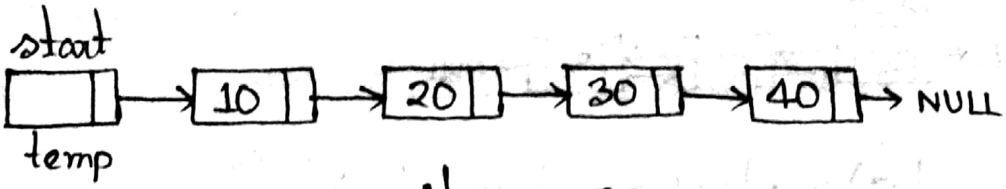
① `temp1->next = temp->next;`

② `temp->next = temp1;`

Insertion function :

```
node *Insertion (node *start, int item)
{
    node *temp, *temp1;
    // Insertion logic
    return start;
}
```

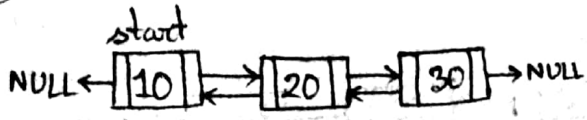
Deletion from linear linked list :



item = 30

```
temp = start;
while ((temp->next != NULL) && (temp -> next -> data != item))
    temp = temp -> next;

if (temp -> next == NULL)
    printf ("Not Found");

else
{
    temp1 = temp -> next;
    temp -> next = temp1 -> next;
    free (temp1);
}
```

Doubly linked list creation:

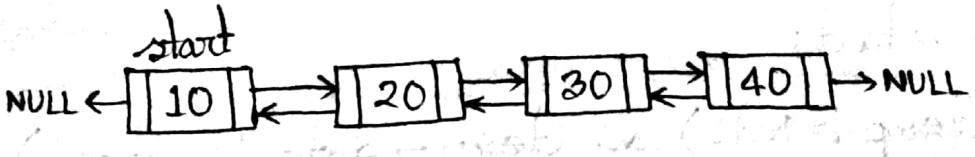



```
start = (node*) malloc (sizeof (node));
temp→data = i;
temp→next = NULL;
start→back = NULL;
prev = start;
for (int i=20; i<=30; i=i+10)
{
    temp = (node*) malloc (sizeof (node));
    temp→data = i;
    temp→next = NULL;
    prev→next = temp;
    temp→back = prev;
    prev = temp;
}
```

Display:

```
temp = start;
while (temp != NULL)
{
    printf ("%d", temp→data);
    temp = temp→next;
}
```

Deletion from doubly linked list:



start = NULL

```
if (start == NULL)
    printf ("No Linked List");
```

First Deletion

```
if (start → data == item)
{
    start = start → next;
    free (start → back);
    start → back = NULL;
}
```

Not Found

```
temp = start;
while((temp != NULL) && (temp → data != item))
    temp = temp → next;
if (temp == NULL)
    printf ("Not Found");
```

Middle Deletion:

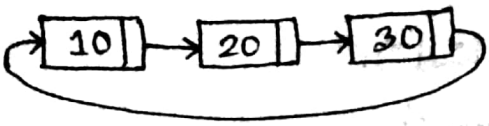
```
temp = start;
while((temp != NULL) && (temp -> data != item))
    temp = temp -> next;
temp -> next -> back = temp -> back;
temp -> back -> next = temp -> next;
free (temp);
```

Last Deletion:

```
temp = start;
while ((temp != NULL) && (temp -> data != item))
    temp = temp -> next;
if (temp -> next == NULL)
{
    temp -> back -> next = NULL;
    free (temp);
}
```

```
node *deletion (node *start. int item)
{
    node *temp;
    if (start == NULL)
        printf ("No Linked List");
    else if (start->data == item)
    {
        start = start->next;
        free (start->back);
        start->back = NULL;
    }
    else
    {
        temp= start;
        while((temp != NULL) && (temp->data != item))
            temp = temp->next;
        if (temp == NULL)
            printf ("Not Found");
        else
        {
            if (temp->next == NULL)
            {
                temp->back->next = NULL;
                free (temp);
            }
            else
            {
                temp->next->back = temp->back;
                temp->back->next = temp->next;
                free (temp);
            }
        }
    }
    return start;
}
```

Linear Circular Linked List Creation:

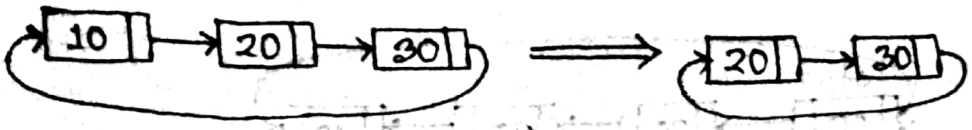



```
start = (node*) malloc (sizeof (node));
start → data = 10;
prev = start;
for (int i=20; i<= 30; i= i+10)
{
    temp = (node*) malloc (sizeof (node));
    temp → data = i ;
    prev → next = temp;
    prev = temp;
}
temp → next = start;
```

Display:

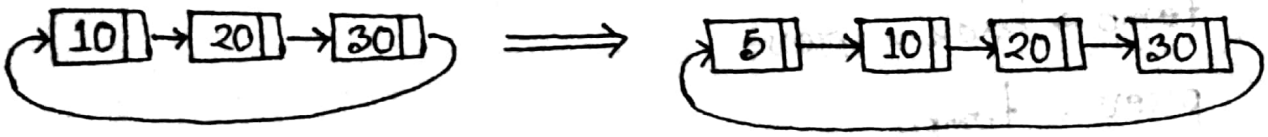
```
temp = start;
while (temp → next != start)
{
   printf ("%d", temp → data);
   temp = temp → next;
}
printf ("%d", temp → data);
```

First Deletion for Linear Circular Linked List:



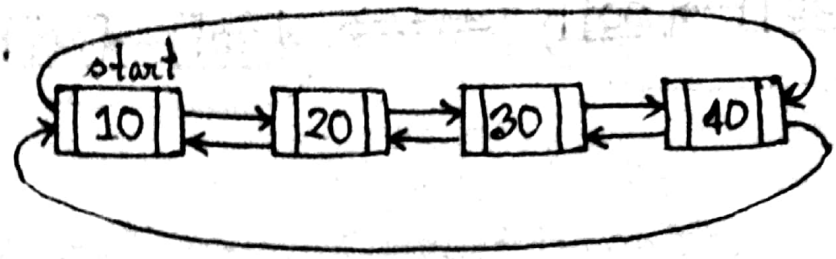

```
temp = start;
while (temp->next != start)
    temp = temp->next;
start = start->next;
free (temp->next);
temp->next = start;
```

First Insertion for Linear Circular Linked List:



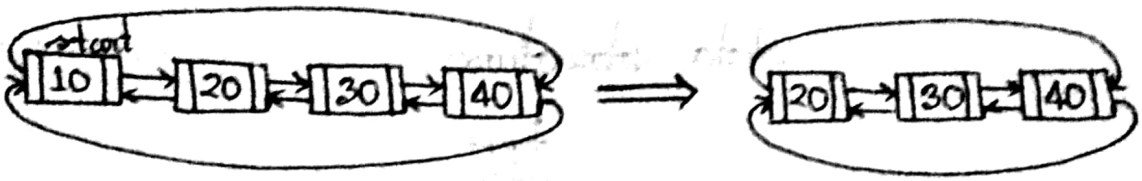

```
temp = start;
while (temp->next != start)
    temp = temp->next;
temp1 = (node*) malloc (sizeof (node));
temp1->data = 5;
temp->next = temp1;
temp1->next = start;
start = temp1;
```

Doubly Circular Linked List Creation :



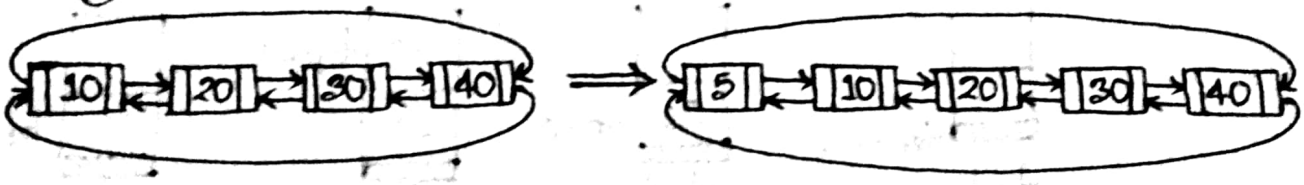

```
start = (node*) malloc (sizeof (node));
start → data = 10;
prev = start;
for (int i = 20; i <= 40; i = i + 10)
{
    temp = (node*) malloc (sizeof (node));
    temp → data = i;
    prev → next = temp;
    temp → back = prev;
    prev = temp;
}
temp → next = start;
start → back = temp → next;
```

Doubly Circular Linked List (first deletion):



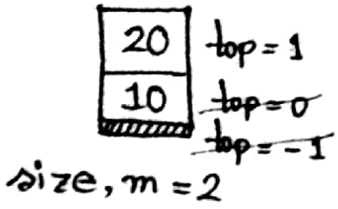

```
temp = start;
while (temp->next != start)
    temp = temp->next;
start = start->next;
free (temp->next);
temp->next = start;
start->back = temp->next;
```

Doubly Circular Linked List (first insertion):




```
temp = start;
while (temp->next != start)
    temp = temp->next;
temp1 = (node*) malloc (sizeof (node));
temp1->data = 5;
temp->next = temp1;
temp1->back = temp->next;
temp1->next = start;
start->back = temp1->next;
start = temp1;
```

Stack : A stack is a LIFO (Last In First Out) data structure.



top++

push : to insert something into stack

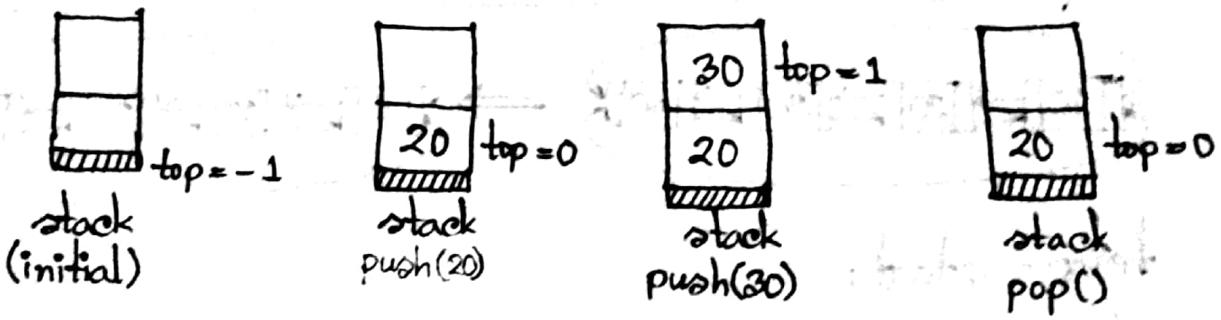
pop : to remove something from stack

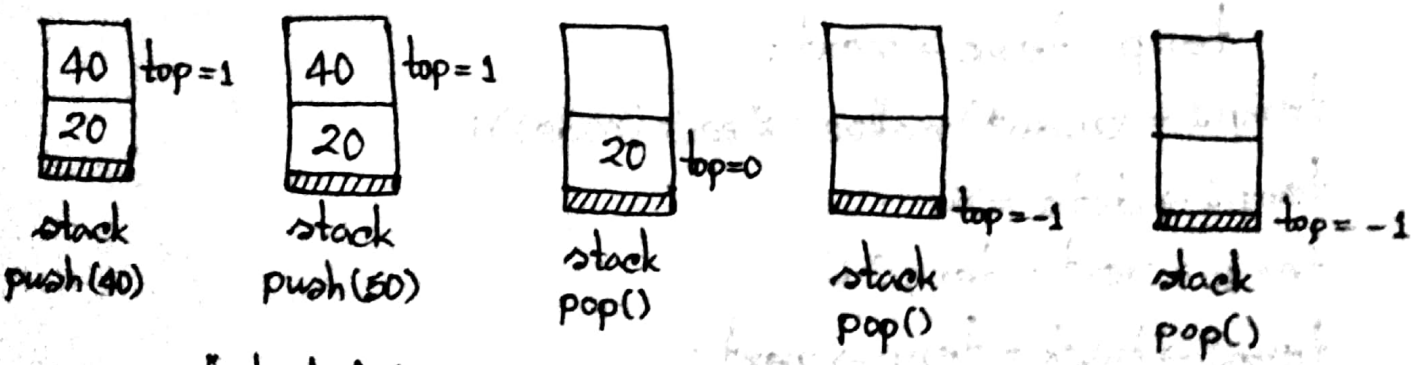
top--

Show stack status for each of the following stack operations using array assuming stack size, m = 2

push(20), push(30), pop(), push(40), push(50), pop(), pop(), pop()

Ans.:





"stack full" or "stack overflow"

"stack empty" or "stack underflow"

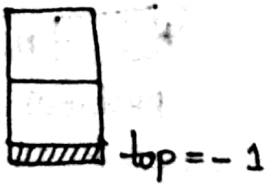
Using Array:

push (stack ⇐ x)

size, m = 2

```
if (top+1 >= m)
    printf ("Stack Overflow");
else
{
    stack[top+1] = x;
    top = top+1;
}
```

pop (stack ⇒ x)




```
if (top == -1)
    printf ("Stack Underflow");
else
{
    stack [top] = 0;
    top = top - 1;
}
```

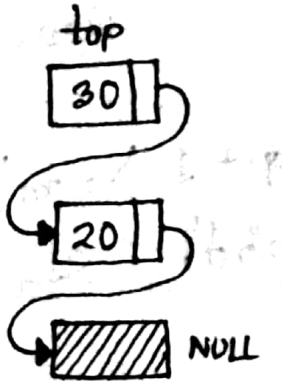
Show stack status for each of the following stack operations using linked list.

push (20), push (30), pop(), push(40), push(50), pop(), pop(), pop(), pop()

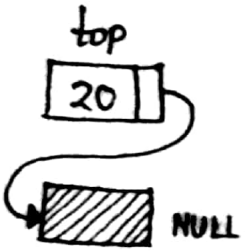
Ans.:

(initial)

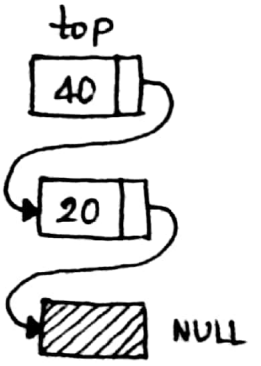
push(20)



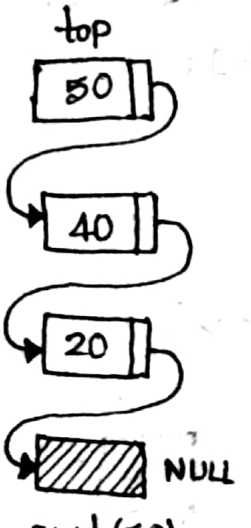
push(30)



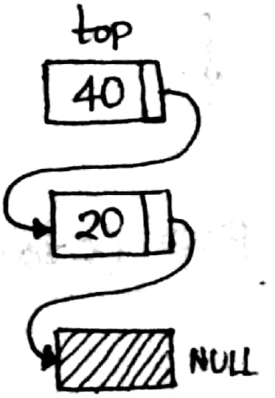
pop()



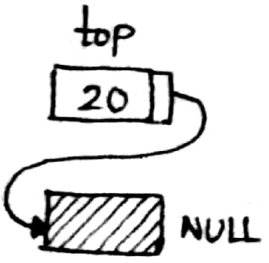
push(40)



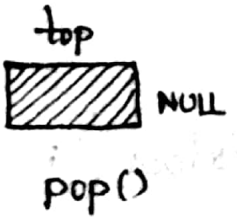
push(50)



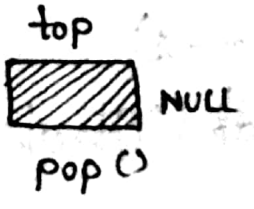
pop()



pop()



pop()



pop()

"stack underflow"

or

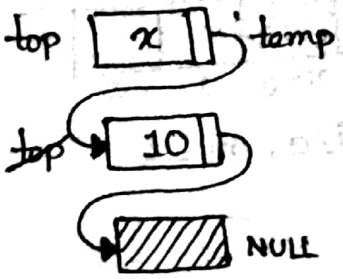
"stack empty"

Using Linked List :

Queue :

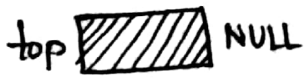
push (stack ⇐ x)

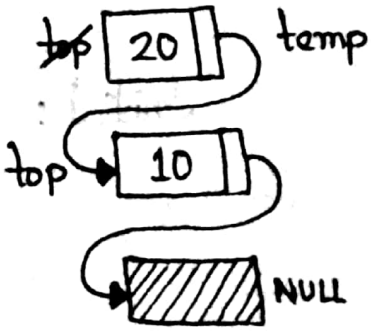
```
temp = new node();
temp → data = x;
temp → next = top;
top = temp;
```



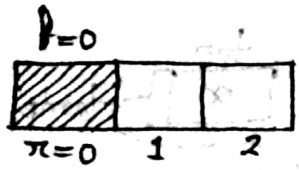
pop (stack ⇒ x)

```
if (top == NULL)
    printf ("Stack underflow");
else
{
    temp = top;
    top = top → next;
    delete (temp);
}
```





Queue: A Queue is a FIFO (First In First Out) data structure queue operation.



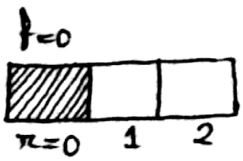
f = front → delete
r = rear → insert

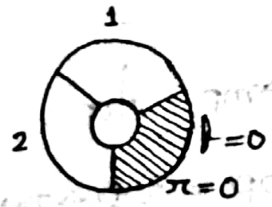
size, m = 2

Show queue status for each of the following operations using array assuming size, m = 2
Insert (10), Insert (20), Delete (), Insert (30), Insert (40), Delete (), Delete (), Delete ()

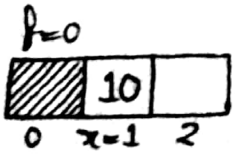
Ans. :

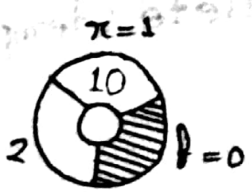
Initial:



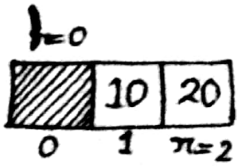


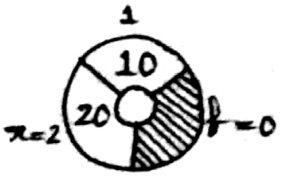
Insert(10):



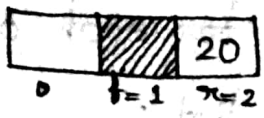


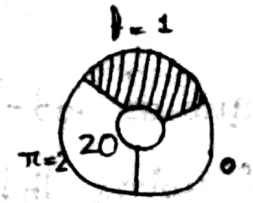
Insert (20):



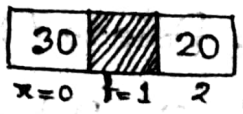


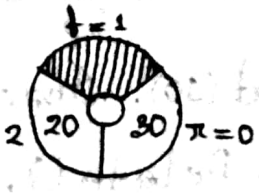
Delete ():





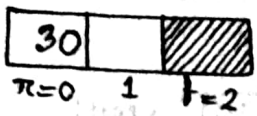
Insert (30):

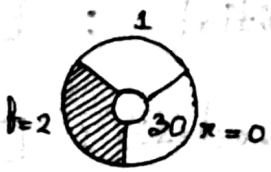




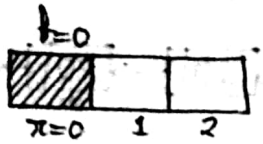
Insert (40): Queue overflow

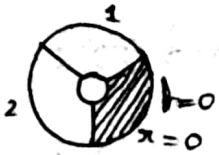
Delete ():





Delete ():





Delete (): Queue underflow

Using Array:

Insert (Queue ⇐ x)

```
s = (r+1) % (m+1);
if (s==f)
   printf ("Queue overflow");
else
{
   Queue [s] = x;
   r = s;
}
```

Delete (Queue ⇒ x)

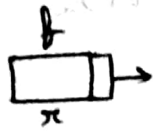
```
if (f==r)
   printf ("Queue underflow");
else
{
   f = (f+1) % (m+1);
   Queue [f] = 0;
}
```

Show queue status for each of the following operations using linked list :

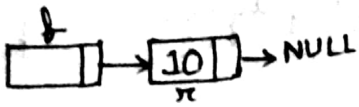
Insert (10). Insert (20). Delete (). Insert (30). Insert (40). Delete (). Delete (). Delete (), Delete ()

Ans. :

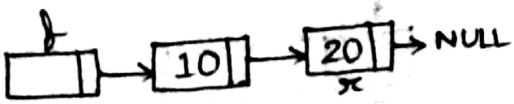
Initial :



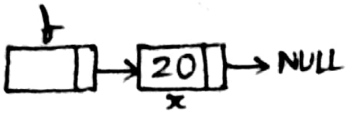
Insert (10) :



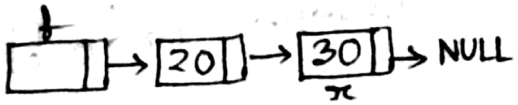
Insert (20) :



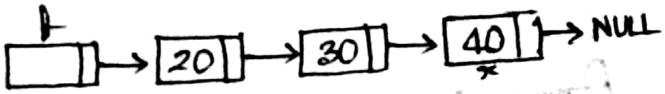
Delete () :



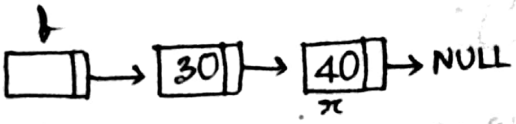
Insert (30) :



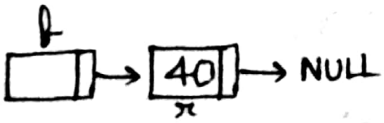
Insert (40) :



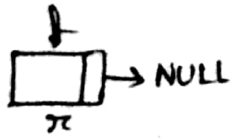
Delete () :



Delete () :



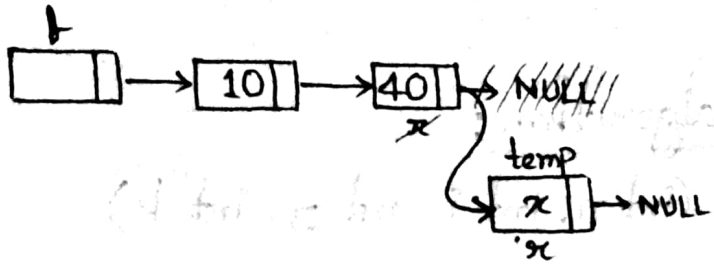
Delete () :



Delete () : Queue underflow

Using Linked List

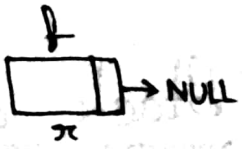
Insert (Queue ⇐ x)

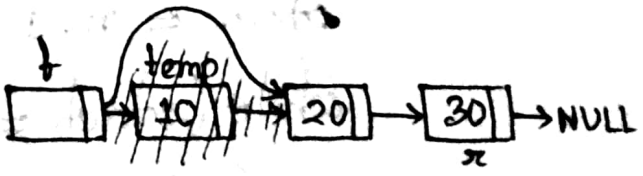


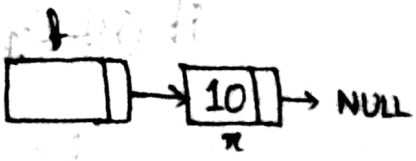

```
temp = (node* ) malloc (sizeof (node));
temp → data = x;
temp → next = NULL;
r → next = temp;
r = temp;
```

Delete (Queue ⇒ x)

```
if (f == r)
    printf ("Queue underflow");
else
{
    temp = f → next;
    f → next = temp → next;
    free (temp);
    if (f → next == NULL)
        r = f;
}
```







Tower Of Hanoi

Recursive algorithm:

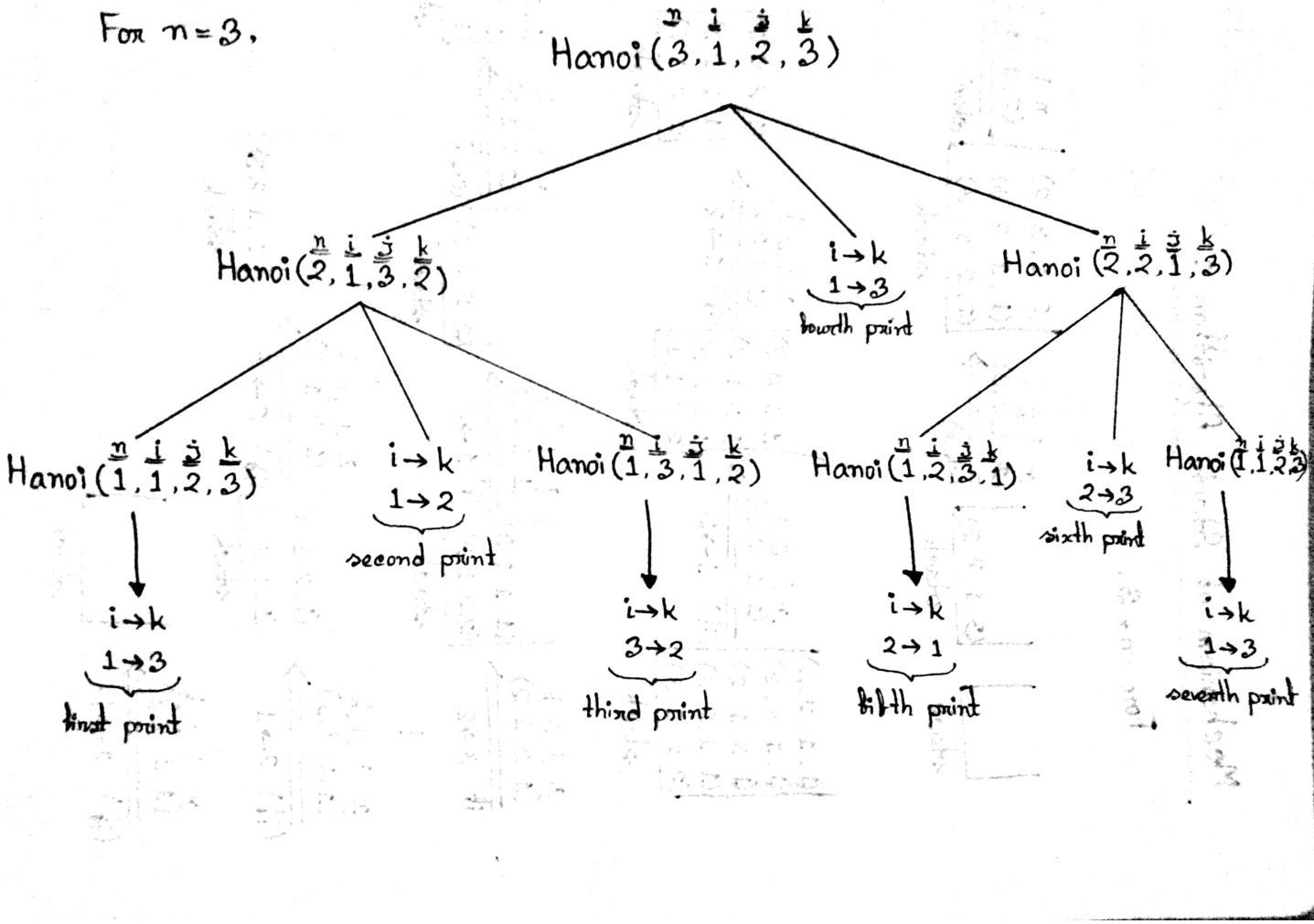
```
void Hanoi (int n, int i, int j, int k)
{
    if (n==1)
        printf ("%d → %d", i, k);
    else
    {
        Hanoi (n-1, i, k, j);
        printf ("%d → %d", i, k);
        Hanoi (n-1, j, i, k);
    }
}
```

Stack algorithm:

```
stack ⇐ n, 1, 2, 3
while (stack ≠ empty)
    (n, i, j, k) ⇐ stack
    if (n==1)
        printf ("%d → %d", i, k);
    else
        stack ⇐ (n-1, j, i, k);
        stack ⇐ (1, i, j, k);
        stack ⇐ (n-1, i, k, j);
end while
```

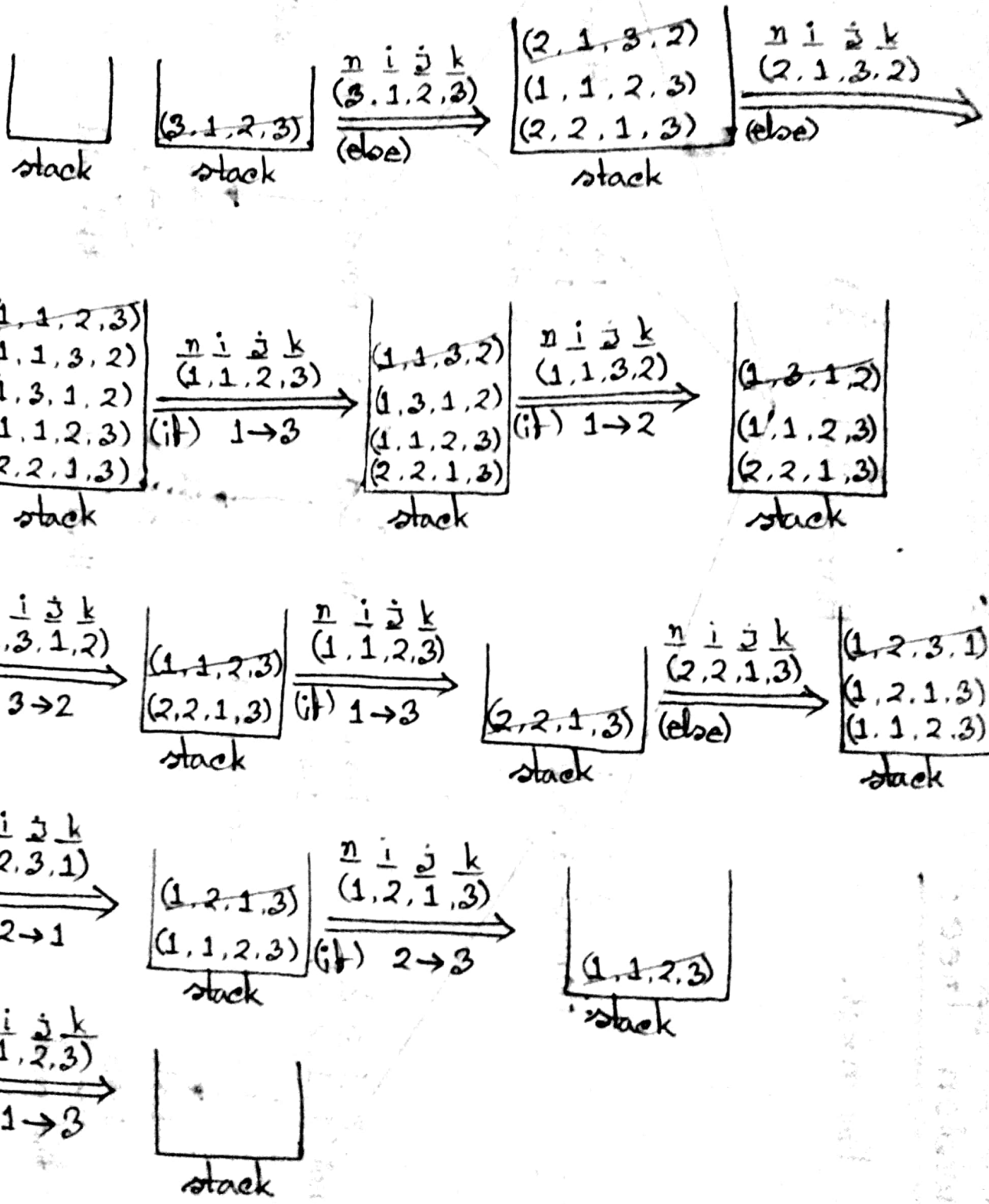
Recursion Tree:

For n = 3,



Mechanism of Stack Algorithm :

For $n = 3$.



Postfix Expression :

a] Convert the following infix expression to postfix

$(a+(b-c+d/e \uparrow 2*f)+a*d)+f$

b] Evaluate your postfix expression for

$a=2,\ b=5,\ c=1,\ d=6,\ e=1,\ f=2$

c] Verify your result

Ans (a)

Infix expression	stack	postfix expression
(a+(b-c+d/e↑2*f)+a*d)+f		
a+(b-c+d/e↑2*f)+a*d)+f	(	
+(b-c+d/e↑2*f)+a*d)+f	(	a
(b-c+d/e↑2*f)+a*d)+f	(+	a
b-c+d/e↑2*f)+a*d)+f	(+(	a
-c+d/e↑2*f)+a*d)+f	(+(	ab
c+d/e↑2*f)+a*d)+f	(+(-	ab
+d/e↑2*f)+a*d)+f	(+(-	abc
d/e↑2*f)+a*d)+f	(+(+	abc-

/e↑2*f)+a*d)+f	(+(+:	abc-d
e↑2*f)+a*d)+f	(+(+/	abc-d
↑2*f)+a*d)+f	(+(+/	abc-de
2*f)+a*d)+f	(+(+/↑	abc-de
*f)+a*d)+f	(+(+/↑	abc-de2
f)+a*d)+f	(+(+*	abc-de2↑/
)+a*d)+f	(+(+*	abc-de2↑/f
+a*d)+f	(+	abc-de2↑/f*+
a*d)+f	(+	abc-de2↑/f*++
*d)+f	(+	abc-de2↑/f*++a
d)+f	(+*	abc-de2↑/f*++a
)+f	(+*	abc-de2↑/f*++ad
+f		abc-de2↑/f*++ad*+
f	+	abc-de2↑/f*++ad*+
	+	abc-de2↑/f*++ad*+f
Empty	Empty	abc-de2↑/f*++ad*+f+

Ans (b):

Postfix	stack	Operation
abc-de2↑f*++ad*+f+		
bc-de2↑f*++ad*+f+	a	
c-de2↑f*++ad*+f+	ab	
-de2↑f*++ad*+f+	abc	
de2↑f*++ad*+f+	a4	b-c = 5-1 = 4
e2↑f*++ad*+f+	a4d	
2↑f*++ad*+f+	a4de	
↑f*++ad*+f+	a4de2	
/f*++ad*+f+	a4d1	e↑2 = 1^2 = 1
f*++ad*+f+	a46	d/1 = 6/1 = 6
++ad+f+	a46f	
++ad*+f+	a412	6*f = 6*2 = 12
+ad*+f+	a16	4+12 = 16
ad*+f+	18	a+16 = 2+16 = 18
d*+f+	18a	
*+f+	18ad	
+f+	1812	a*d = 2*6 = 12
f+	30	18+12 = 30
+	30f	
Empty	32	30+f = 30+2 = 32

Ans (c) :

$$(a+(b-c+d/e\uparrow 2*f)+a*d)+f$$

$\Rightarrow (2+(5-1+6/1\uparrow 2*2)+2*6)+2$

$\Rightarrow (2+(5-1+6/1*2)+2*6)+2$

$\Rightarrow (2+(5-1+6*2)+2*6)+2$

$\Rightarrow (2+(5-1+12)+2*6)+2$

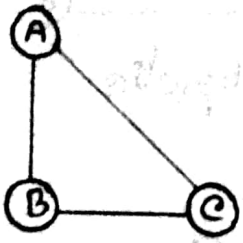
$\Rightarrow (2+16+2*6)+2$

$\Rightarrow (2+16+12)+2$

$\Rightarrow 30+2$

$\Rightarrow 32$

GRAPH



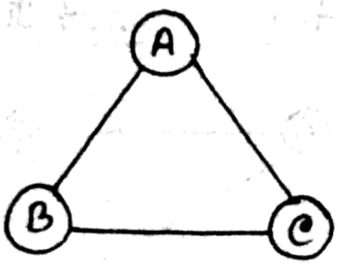
Vertices, $V = \{A, B, C\}$ $\quad |V| = 3$

Edges, $E = \{AB, AC, BC\}$ $\quad |E| = 3$

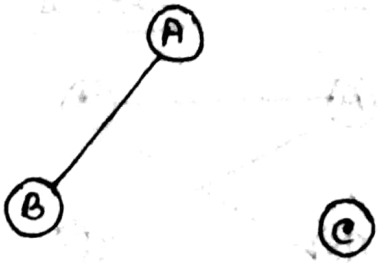
$= \{BA, CA, CB\}$

Graph, $G = (V, E)$

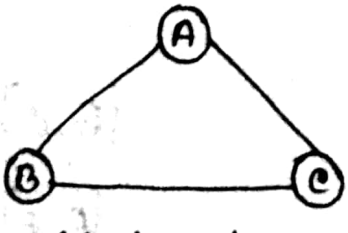
Types of graph :



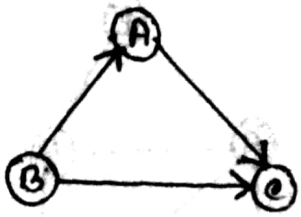
connected



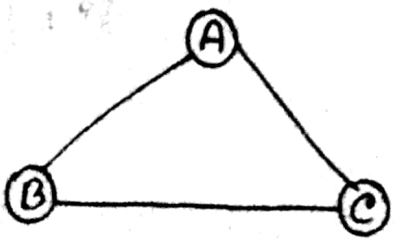
disconnected



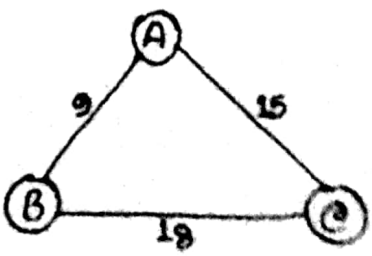
Undirected



Directed

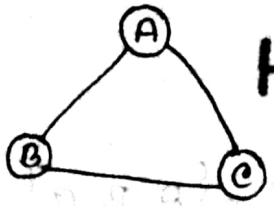


Unweighted

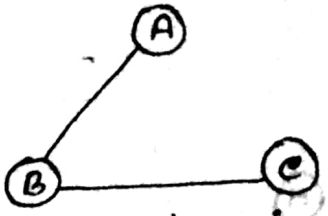


Weighted

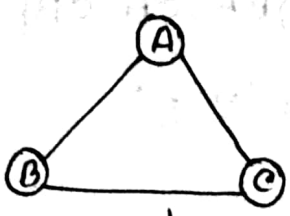
GRAPH



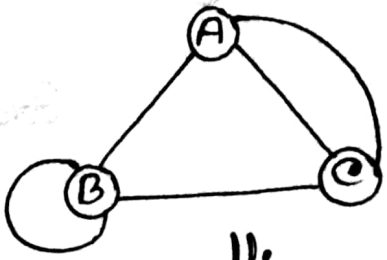
Cyclic



Acyclic

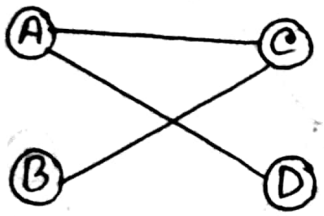


simple



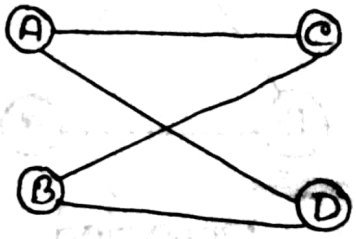
multi

set I set II

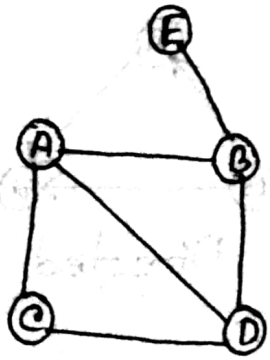


Bipartite

set I set II

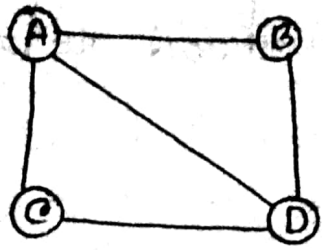


Complete Bipartite



{A, B, D}
{A, C, D}
} Clique

{B, C}
{A, E}
{D, E}
{C, E}
} Independent Set



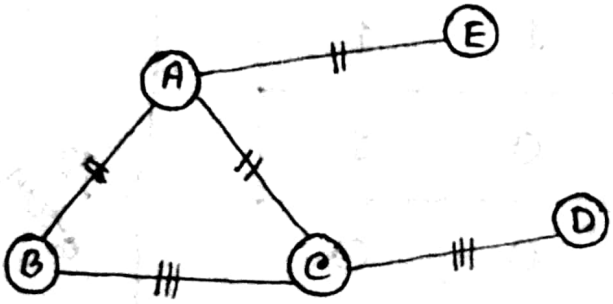
$|E| = 5$

Degree of vertex A : 3
Degree of vertex B : 2
Degree of vertex C : 2
Degree of vertex D : 3

10
⇒ 2×5
⇒ 2×|E|

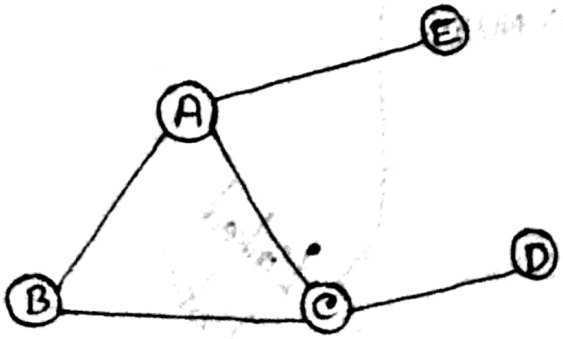
Total degree = 2 × Total edges

Node cover or Vertex cover :



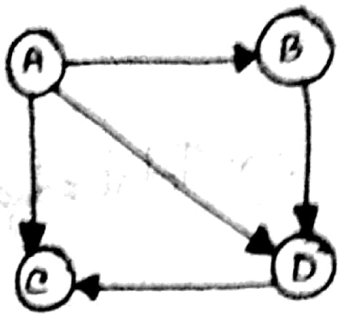
minimum vertex cover : {A, C}

Edge covex :



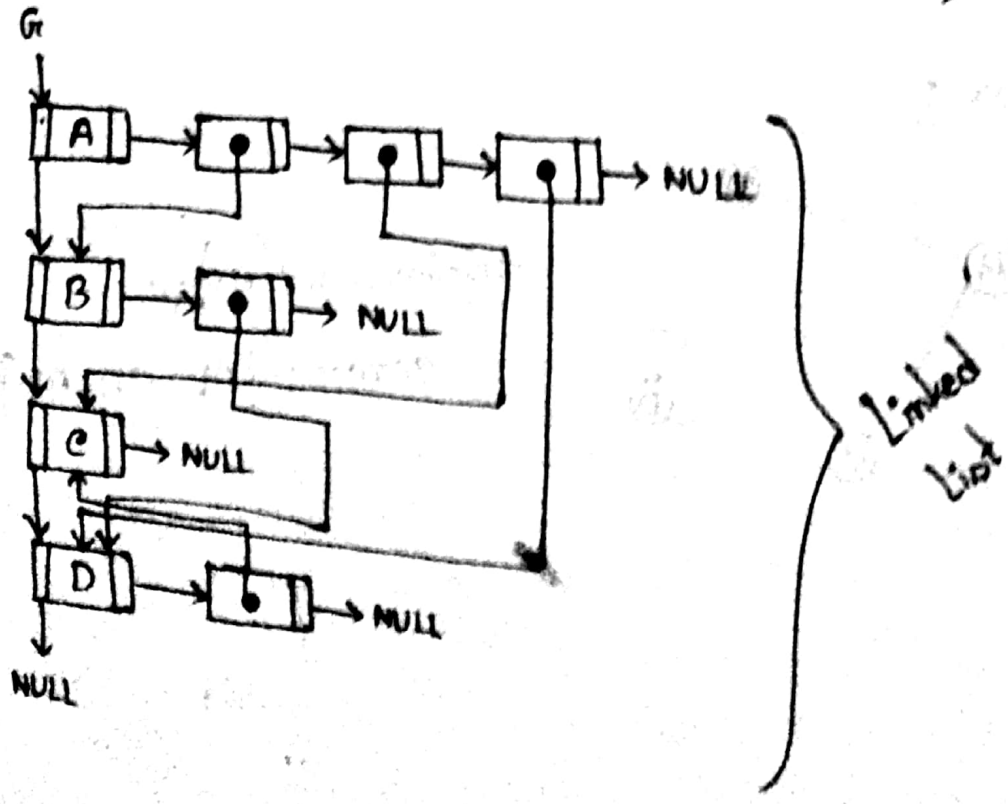
minimum edge cover : {AE, CD, AB}

Adjacency structure or Neighborhood relation or Graph representation :



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

2D array

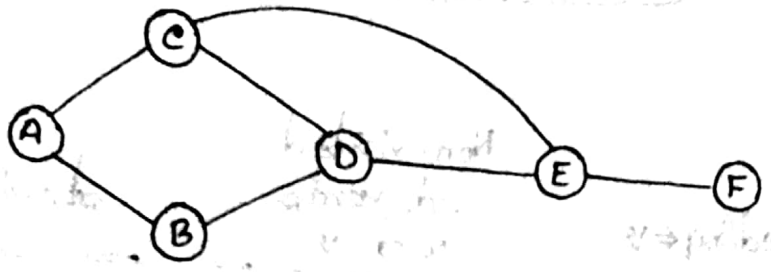


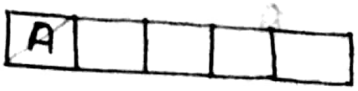
Graph Traversal Techniques:

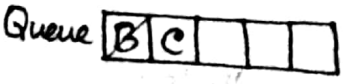
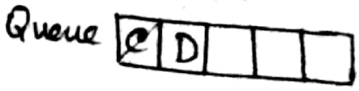
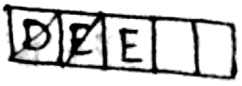
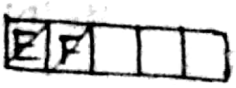
→ BFS (Breadth First Search)

→ DFS (Depth First Search)

BFS:

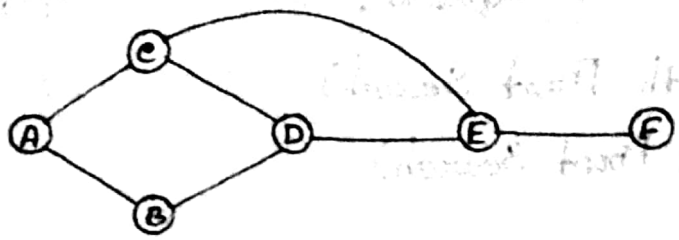


Queue 

v ← Queue	visit v	VisitedSeq ← v	Non-visited adjacents u of v	Queue ← u
v = 'A'	A	A	u = {B, C}	Queue 
v = 'B'	B	AB	u = {~~A~~, D}	Queue 
v = 'C'	C	ABC	u = {~~A~~, D, E}	Queue 
v = 'D'	D	ABCD	u = {~~B~~, ~~C~~, E}	Queue 
v = 'D'	x	x	x	x
v = 'E'	E	ABCDE	u = {~~C~~, ~~D~~, F}	Queue 
v = 'E'	x	x	x	x
v = 'F'	F	ABCDEF	u = {~~E~~}	x

BFS Sequence : ABCDEF

DFS :



stack [A | | | | |]

v ← stack	visit v	VisitedSeq ← v	Non-visited adjacents u of v	stack ← u
v = 'A'	A	A	u = {B, C}	stack [B C | | |]
v = 'C'	C	AC	u = {A, D, E}	stack [B D E | |]
v = 'E'	E	ACE	u = {C, D, F}	stack [B D D F |]
v = 'F'	F	ACEF	u = {E}	x
v = 'D'	D	ACEFD	u = {B, C, E}	stack [B D B | |]
v = 'B'	B	ACEFDB	u = {A, D}	x
v = 'D'	D	x	x	x
v = 'B'	B	x	x	x

DFS Sequence : ACEFDB

BFS Algorithm :

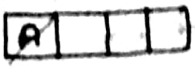
```
Queue ⇐ start
while Queue ≠ empty
    v ⇐ Queue
    if v is not visited
        visit v
        VisitedSeq ⇐ v
        for all non-visited adjacency u of v
            Queue ⇐ u
        end for
    end if
end while
```

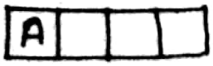
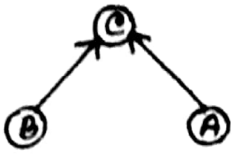
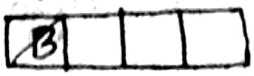
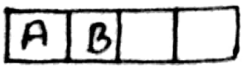
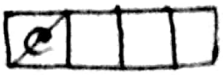
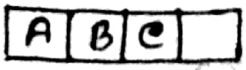
DFS Algorithm :

```
stack ⇐ start
while stack ≠ empty
    v ⇐ stack
    if v is not visited
        visit v
        VisitedSeq ⇐ v
        for all non-visited adjacency u of v
            stack ⇐ u
        end for
    end if
end while
```

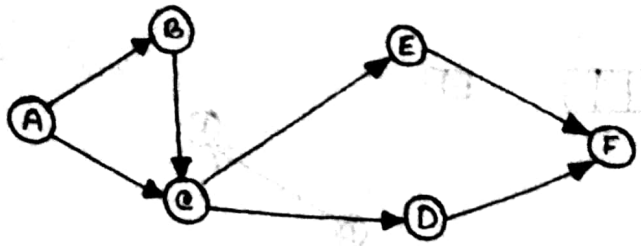
Topological Ordering:

- Topological Ordering can only be applied to DAG (Directed Acylic Graph)
- Set is a 'queue'
- Nodes with incoming edge can't be written in Set

Set 

m ← Set	Order ← m	mn	Disconnected mn from graph	Set ← n
m='A'		AB		
		AC		X
m='B'		BC	Empty	
m='C'		X	X	X

Topological Order : A → B → C



Queue

Set | A | | | | |

m ← set

Order ← m

mn

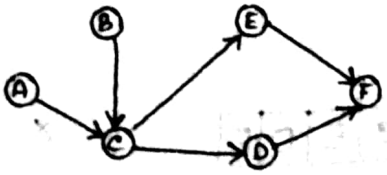
Disconnected mn from graph

Set ← n

m = 'A'

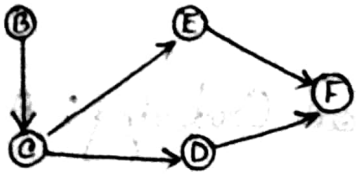
| A | | | | | |

AB



Set | B | | | | |

AC

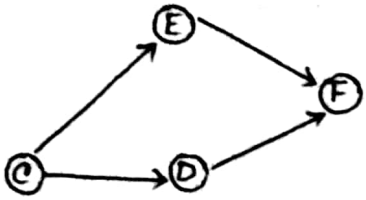


×

m = 'B'

| A | B | | | | |

BC

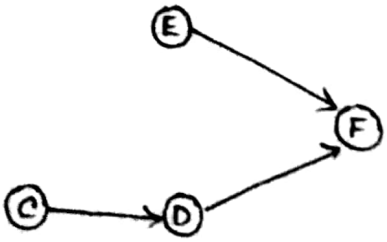


Set | C | | | | |

m = 'C'

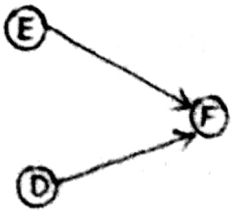
| A | B | C | | | |

CE



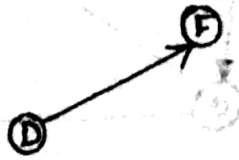
Set | E | | | | |

CD



Set | E | D | | | |

m = 'E' | A | B | C | E | | | EF



X

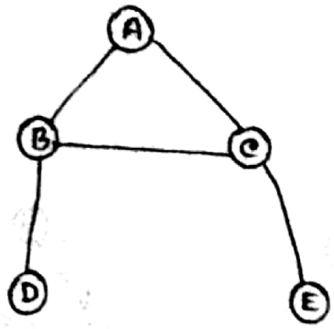
m = 'D' | A | B | C | E | D | | DF

Empty Set | F | | | |

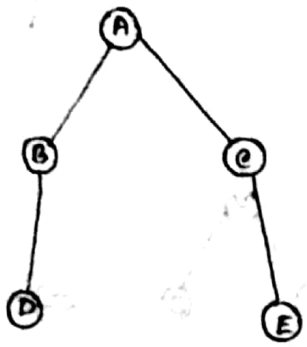
m = 'F' | A | B | C | E | D | F | X X X

Topological Ordering: A → B → C → E → D → F

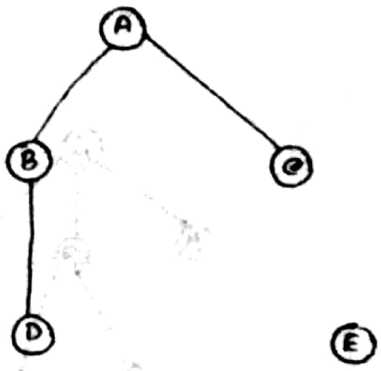
TREE: A Tree is an acyclic connected graph.



connected
cyclic
(not a tree)

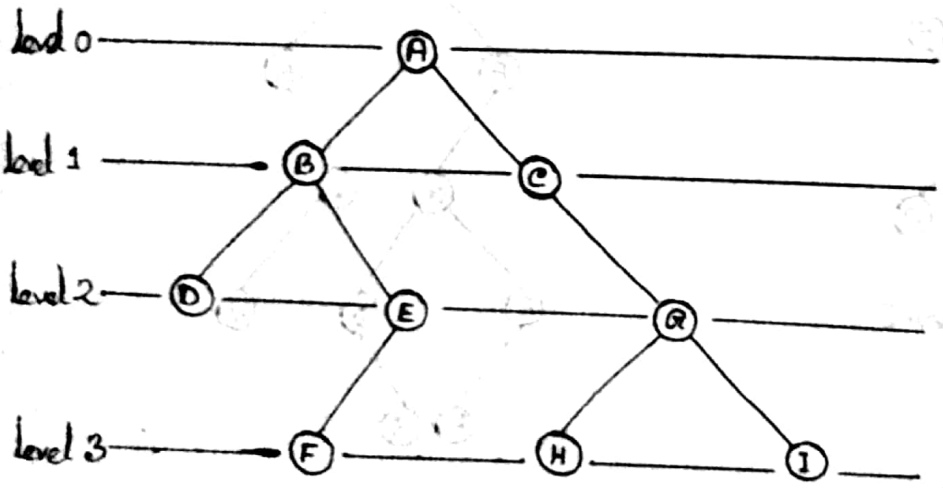


connected
acyclic
(tree)



disconnected
acyclic
(not a tree)

Family Structure of Tree:



A root of tree

D, F, H, I (No child) ⇒ leaf node

children of A ⇒ B, C (2)
" " B ⇒ D, E (2)
" " E ⇒ F (1)
" " C ⇒ G (1)
" " G ⇒ H, I (2)

maximum children: 2

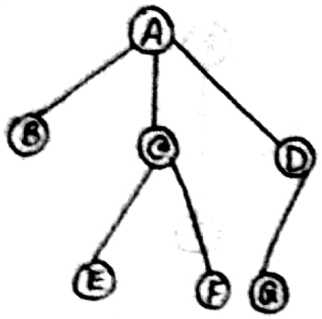
height: 3

B ↔ C (siblings)
D ↔ E (siblings)
H ↔ I (siblings)
F ↔ A (ancestor)

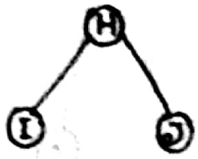
2-array tree
or
binary tree

Descendant of A ⇒ B, C, D, E F, G, H, I

Forest ⟹ Binary Tree

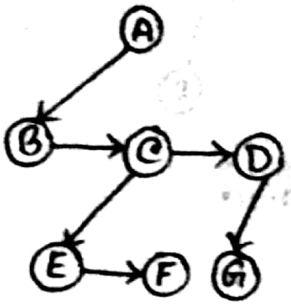


Tree 1

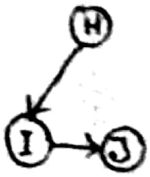


Tree 2

⟹

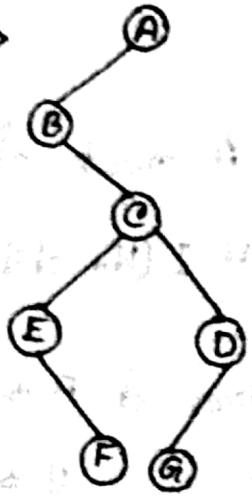


Tree 1

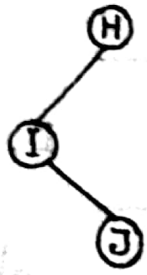


Tree 2

⟹

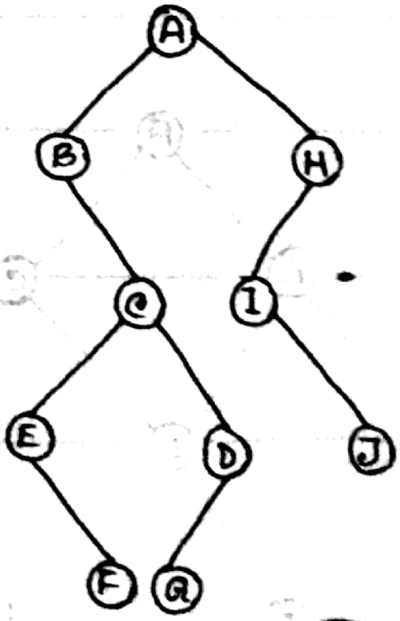


Tree 1



Tree 2

⟹



Family member(7)

Family member(3)

Binary Tree

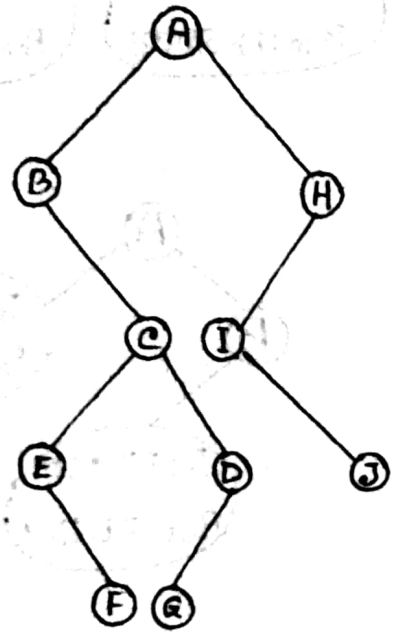
Tree traversal technique :

→ Pre - Order (root - left - right)

→ In - Order (left - root - right)

→ Post - Order (left - right - root)

→ Level Order (level by level)



PreOrder : ABCEFDGHIJ

InOrder : BEFCGDAIJH

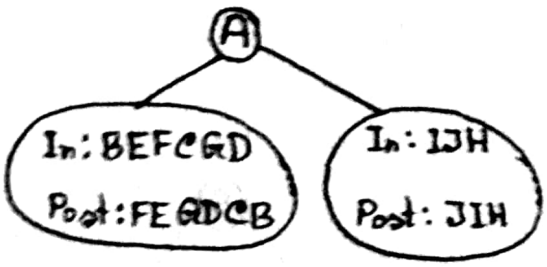
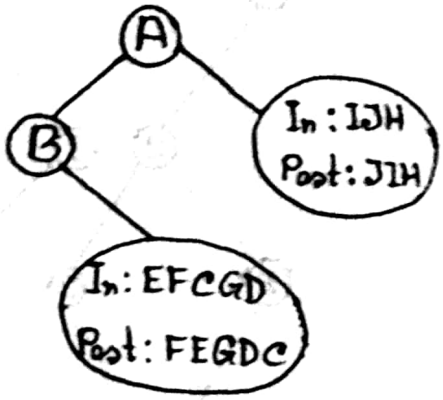
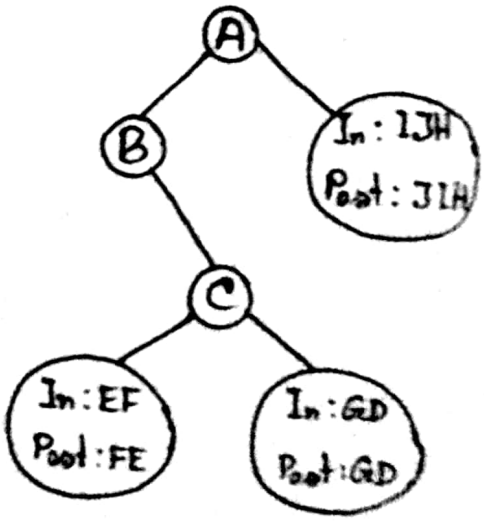
Post Order : FEGDCBJIHA

Level Order : ABHCIEDJFG

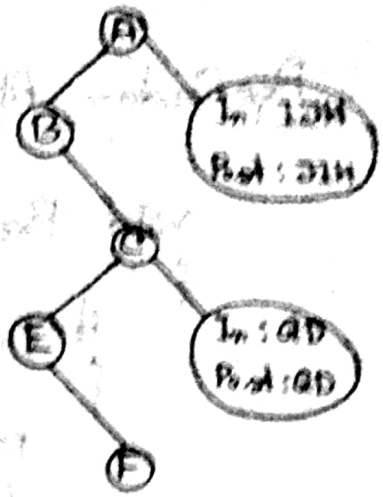
Construct a binary tree from following sequence:

In order : BEFCGDAIJH

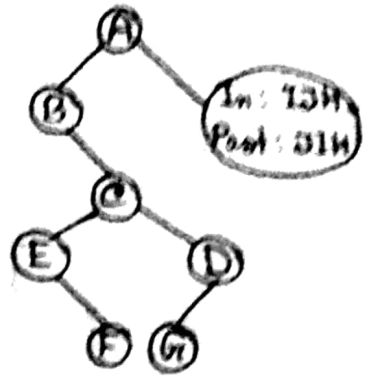
Post order : FEGDCBJIHA

root	left	right	Tree
A	In: BEFCGD Post: FEGDCB	In : IJH Post : JIH	
B	In : - Post: -	In : EFCGD Post : FEGDC	
C	In : EF Post : FE	In : GD Post : GD	

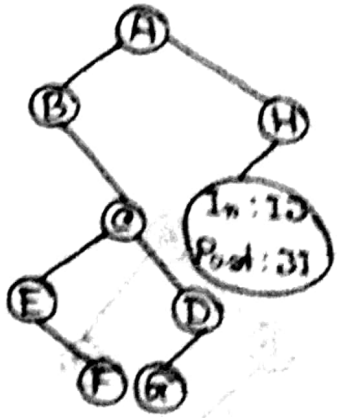
E In: - Post: - In: F Post: F



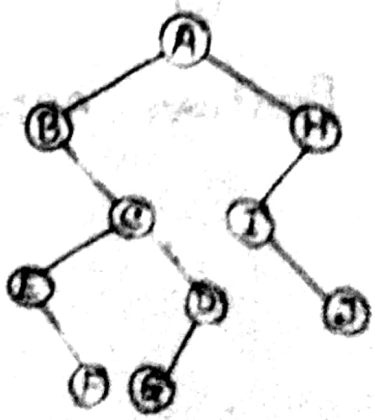
D In: G Post: G In: - Post: -



H In: IJ Post: JI In: - In: -

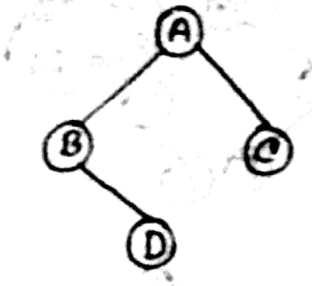


I In: - Post: - In: J Post: J

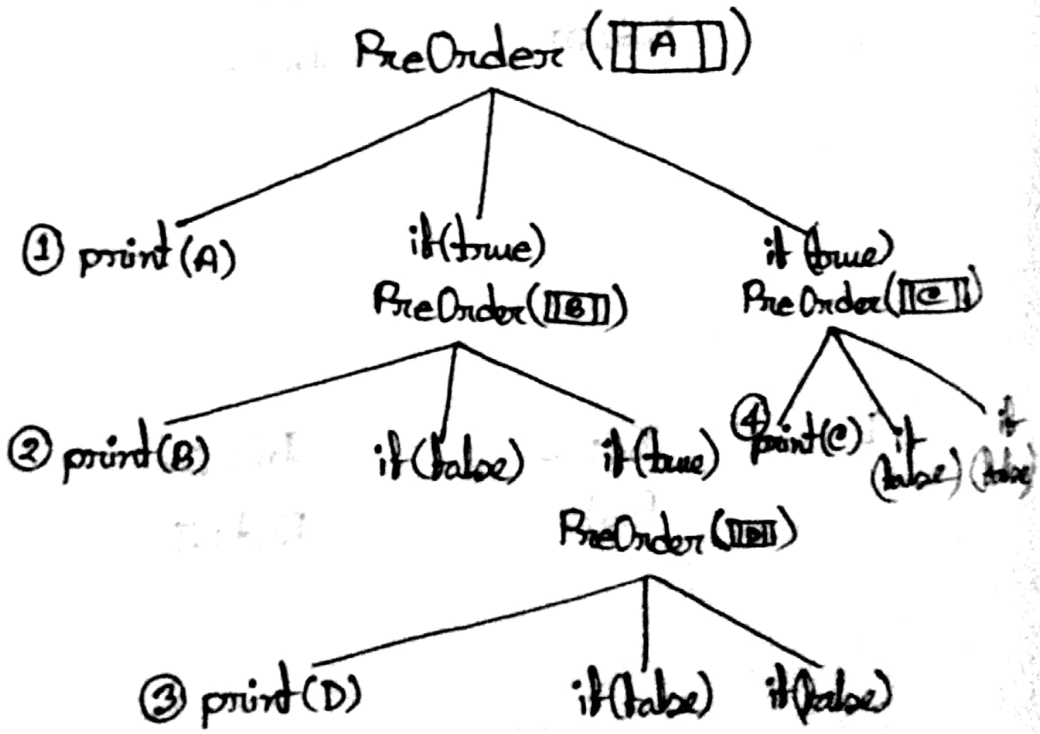


PreOrder Algorithm :

```
void PreOrder (node *root)
{
    if (root != NULL)
    {
        printf ("%c", root->data);
        if (root->left != NULL)
            PreOrder (root->left);
        if (root->right != NULL)
            PreOrder (root->right);
    }
}
```



PreOrder : ABDC



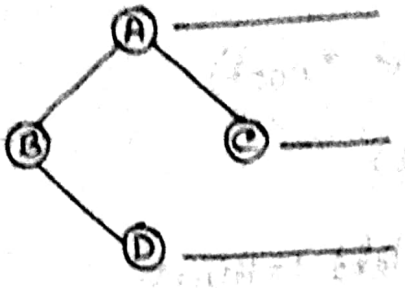
In Order Algorithm :

```
void InOrder (node *root)
{
    if (root != NULL)
    {
        if (root->left != NULL)
            InOrder (root->left);
        printf ("%c", root->data);
        if (root->right != NULL)
            InOrder (root->right);
    }
}
```

Post Order Algorithm :

```
void PostOrder (node *root)
{
    if (root != NULL)
    {
        if (root->left != NULL)
            PostOrder (root->left);
        if (root->right != NULL)
            PostOrder (root->right);
        printf ("%c", root->data);
    }
}
```

Level Order :



v ← Queue	visit v	VisitSeq ← v	l = left(v)	Queue ← l	r = right(v)	Queue ← r
v = 'A'	A	A	l = 'B'	B	r = 'C'	B C
v = 'B'	B	AB	×	×	r = 'D'	C D
v = 'C'	C	ABC	×	×	×	×
v = 'D'	D	ABCD	×	×	×	×

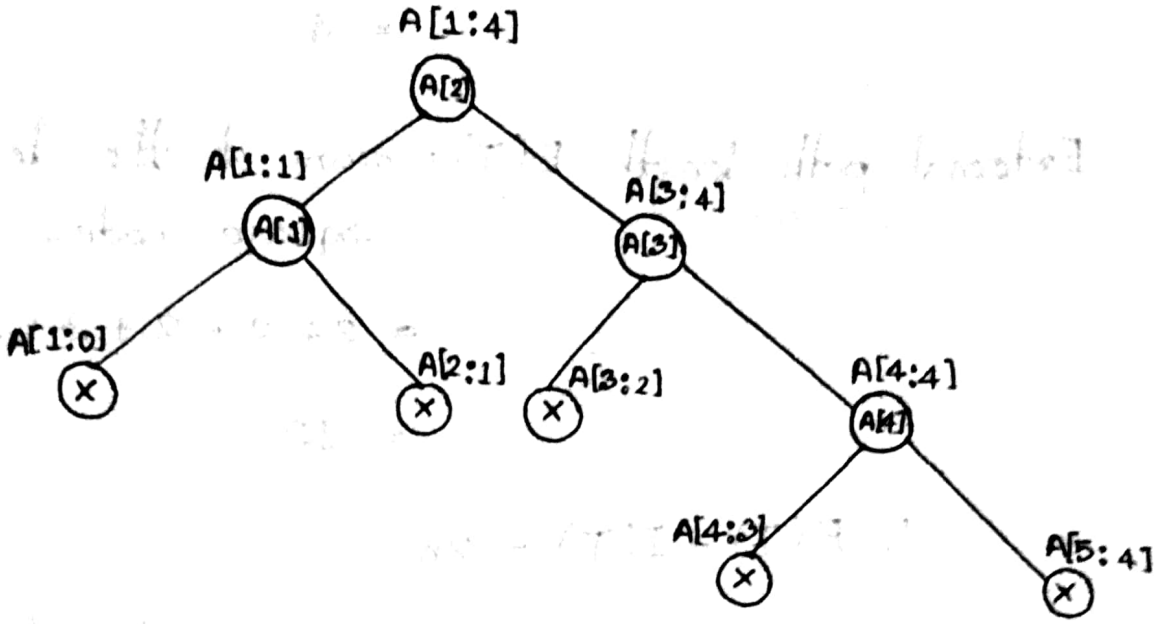
Level Order : ABCD

Level Order Algorithm :

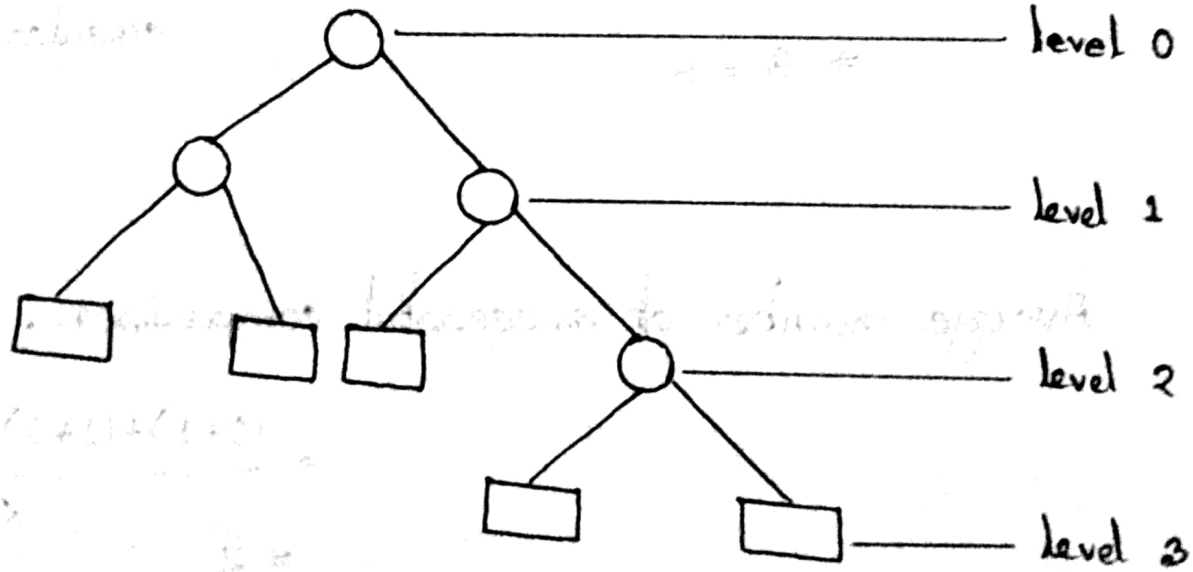
```
Queue ⇐ root
while Queue ≠ empty
{
    v ← Queue
    visit v
    visitedseq ← v
    if left(v) ≠ NULL
        Queue ← left(v)
    if right(v) ≠ NULL
        Queue ← right(v)
}
```

Quantative Aspect of Binary Tree :

Binary Search Tree for [1:4] :



⇒



Internal node (O), $I(n) = 4$

External node (□), $E(n) = 5$

$\therefore E(n) = I(n) + 1$

Internal path length, $I(T)$ = sum of the levels of circular nodes

$= 0 + 1 + 1 + 2$

$= 4$

External path length, $E(T)$ = sum of the levels of square nodes

$= 2 + 2 + 2 + 3 + 3$

$= 12$

$\therefore E(T) - I(T) = 2n$

$\Rightarrow 12 - 4 = 2 * 4$

$\Rightarrow 8 = 8$

n = internal node / circular node

Average number of successful comparison: $S(n) = \frac{\sum(l_i + 1)}{n}$

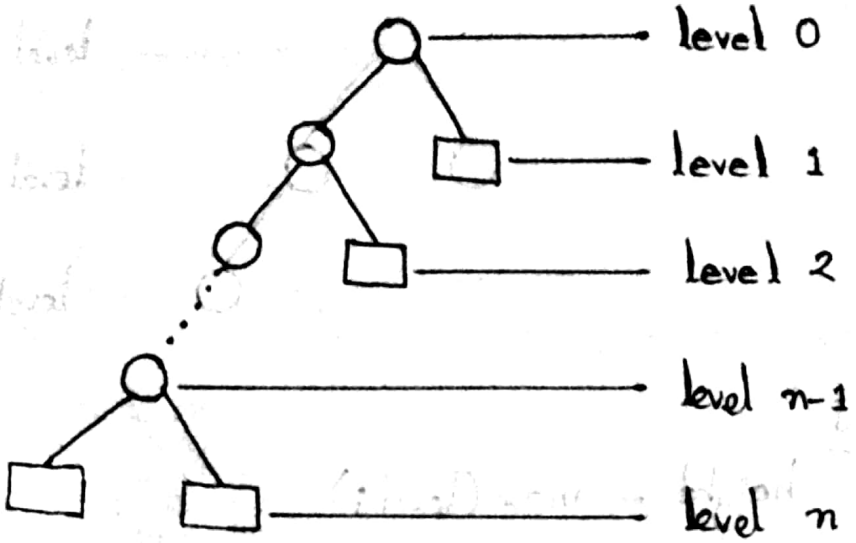
$= \frac{(0+1)+(1+1)+(1+1)+(2+1)}{4}$

$= 2$

Average number of unsuccessful comparison, $U(n) = \frac{\sum l_i}{(n+1)}$

$= \frac{2+2+2+3+3}{4+1}$

$= 2.4$



Internal path length, $I(T) = 0 + 1 + 2 + \ldots\ldots + n-1$

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

External path length, $E(T) = 1 + 2 + \ldots\ldots + n + n$

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

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

$$= \frac{n(n+3)}{2} \quad \ldots\ldots (2)$$

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

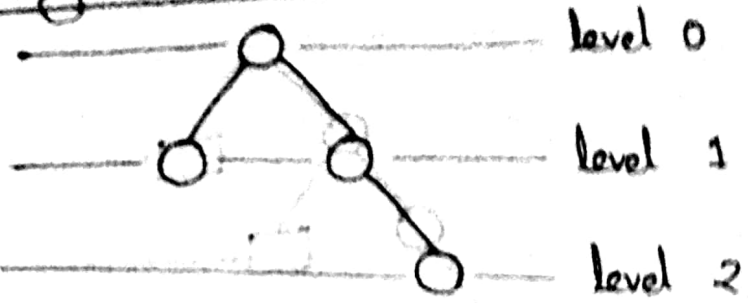
$$= \frac{n^2 + 3n - n^2 + n}{2}$$

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

$$= 2n$$

$\therefore E(T) - I(T) = 2n$

Height measuring methods :



Method I

height = max (level i)

= 2

Method II

height = $\lfloor \log_2^n \rfloor$

= $\lfloor \log_2^4 \rfloor$

= $\lfloor 2 \log_2^2 \rfloor$

= $\lfloor 2 * 1 \rfloor$

= 2

Method III

height (T) = max (height (T_1), height (T_n)) + 1

= max (0, max (height(T_{n1}), height(T_{nn})) + 1) + 1

= max (0, max (−1, 0) + 1) + 1

= max (0, 0 + 1) + 1

= max (0, 1) + 1

= 1 + 1

= 2

```
int height (node *root)
{
    if (root == NULL)
        return -1;
    else if ((root->left == NULL) && (root->right == NULL))
        return 0;
    else
        return max (height(root->left), height(root->right)) +
}

int max (int x, int y)
{
    if (x>y)
        return x;
    else
        return y;
}
```

Hashing :

English Alphabate = 26

Code for each letter = $\lceil \log_2^{26} \rceil$

$= \left\lceil \frac{\log e^{26}}{\log e^{2}} \right\rceil$

$= \left\lceil \frac{\ln 26}{\ln 2} \right\rceil$

$= \lceil 4.7 \rceil$

$= 5$ bit

A = 00000 = 0

B = 00001 = 1

C = 00010 = 2

D = 00011 = 3

⋮

T = 10011 = 19

⋮

Z = 11001 = 25

Hash function generation methods :

→ Extraction

→ Compression

→ Division

→ Multiplication

Extraction :

size, $m = 4$

Words	Bit Representation	Hash Table Index
CAT	00010 00000 10011	$(01)_2 = (1)_{10}$
BAT	00001 00000 10011	$(01)_2 = (1)_{10}$

Hash Table

	0
CAT, BAT	1
	2
	3

Division :

size, $m = 4$

Words	Bit Representation	Hash Table Index
CAT	00010 00000 10011 $= (2067)_{10}$	2067 mod 4 = 3
BAT	00001 00000 10011 $= (1043)_{10}$	1043 mod 4 = 3

Hash Table

	0
	1
	2
CAT, BAT	3

Compression :

size, m = 4

Words	Bit Representation	Hash Table Index
CAT	$00010 \oplus 00000 \oplus 10011 = (10001)_2$	$(10001)_2 \bmod 4 = 17 \bmod 4 = 1$
BAT	$00001 \oplus 00000 \oplus 10011 = (10011)_2$	$(10010)_2 \bmod 4 = 18 \bmod 4 = 2$

```
 00010
 00000
 10011
-------
 10001
```

Hash Table

	0
CAT	1
BAT	2
	3

```
 00001
 00000
 10011
-------
 10010
```

Multiplication :

size, $m = 4$

Words	Bit Representation	Hash Table Index
CAT	00010 00000 10011 $= (2067)_{10}$	0
BAT	00001 00000 10011 $= (1043)_{10}$	3

$h(z) = \lfloor m (z\theta \bmod 1) \rfloor$ where $\theta = 0.6125\ldots$

$h(CAT) = \lfloor 4 * (2067 * 0.6125 \bmod 1) \rfloor$

$= \lfloor 4 * (1266.0375 \bmod 1) \rfloor$

$= \lfloor 4 * 0.0375 \rfloor$

$= \lfloor 0.15 \rfloor$

$= 0$

$h(BAT) = \lfloor 4 * (1043 * 0.6125 \bmod 1) \rfloor$

$= \lfloor 4 * (638.8375 \bmod 1) \rfloor$

$= \lfloor 4 * 0.8375 \rfloor$

$= \lfloor 3.35 \rfloor$

$= 3$

Hash Table

CAT	0
	1
	2
BAT	3