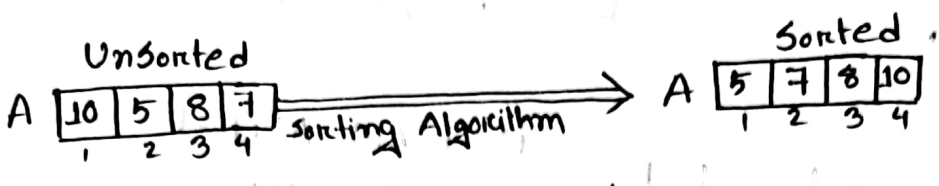
Lecture 1

Date: 19.06.2022

Sorting Algorithms:



- ✓ Insertion Sort
- Bubble Sort
- ✓ Heap Sort → Will read after Mid
- ✓ Merge Sort
- ✓ Quick Sort
- Bucket Sort
- Selection Sort
- Radix Sort
- ✓ Counting Sort

Sorting are in two ways : ① Ascending order (ছোট থেকে বড়)

② Descending order (বড় থেকে ছোট)

Insertion Sort :

A [10 | 5 | 8 | 7] (1 2 3 4) No. of elements. no = 04

$j=2$, $t=5$, $i=j-1=2-1=1$

$i>=1$ AND $A[i]>t$
$\Rightarrow 1>=1$ AND $A[1]>5$
$\Rightarrow$ True AND $\underbrace{10>5}_{\text{True}}$

A [| 10 | 8 | 7] (1 2 3 4)

$i=0$

$i>=1$ AND $A[i]>t$
$\Rightarrow 0>=1$ AND $A[0]>5$
$\Rightarrow$ false AND false

$i+1=0+1=1$

A [5 | 10 | 8 | 7] (1 2 3 4)

$j=3$, $t=8$, $i=j-1=3-1=2$

$i>=1$ AND $A[i]>t$
$\Rightarrow 2>=1$ AND $A[2]>8$
$\Rightarrow$ True AND $\underbrace{10>8}_{\text{True}}$ True

A [5 | | 10 | 7] (1 2 3 4)

$i=1$

$i>=1$ AND $A[i]>t$
$\Rightarrow 1>=1$ AND $A[1]>8$
$\Rightarrow$ True AND $\underbrace{5>8}_{\text{False}}$ False

A [5 | 8 | 10 | 7] (1 2 3 4)

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

$i>=1$ AND $A[i]>t$
$\Rightarrow 3>=1$ AND $A[3]>7$
$\Rightarrow$ True AND $\underbrace{10>7}_{\text{True}}$ True

A [5 | 8 | | 10] (1 2 3 4)

$i=2$

$i>=1$ AND $A[i]>t$
$\Rightarrow 2>1$ AND $A[2]>7$
$\Rightarrow$ True AND $\underbrace{8>7}_{\text{True}}$ True

A [5 | | 8 | 10] (1 2 3 4)

i = 1

i >= 1 AND A[i] > t

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

=> True AND (5 > 7) False (False)

A: 5 | 7 | 8 | 10 (1 2 3 4)

i + 1 = 1 + 1 = 2

Time Complexity of Insertion Sort:

A: 10 | 5 | 9 (1 2 3) No. of elements n = 3

j = 2, t = 5, i = j - 1 = 2 - 1 = 1

i >= 1 AND A[i] > t

=> 1 >= 1 AND A[1] > 5

=> True AND (10 > 5) True (True)

A: | 10 | 9 (1 2 3)

i = 0

i >= 1 AND A[i] > t

=> 0 >= 1 AND A[0] > 5

=> False AND — False

i + 1 = 0 + 1 = 1

A: 5 | 10 | 9 (1 2 3)

j = 3, t = 9, i = j - 1 = 3 - 1 = 2

i >= 1 AND A[i] > t

=> 2 >= 1 AND A[2] > 9

=> True AND (10 > 9) True (True)

A: 5 | | 10 (1 2 3)

i = 1

i >= 1 AND A[i] > t

=> 1 >= 1 AND A[1] > 9

=> True AND (5 > 9) False (False)

A: 5 | 9 | 10 (1 2 3)

i + 1 = 2 + 1 = 2

Insertion Sort Algorithm :

```
L1: for j = 2 to n
L2:     t = A[j]
L3:     i = j - 1
L4:     while ((i >= 1) AND (A[i] > t))
L5:         A[i+1] = A[i]
L6:         i = i - 1
        end while
L7:     A[i+1] = t
    end for
```

C program for Insertion Sort :

```
for(j=1 ; j<=n-1 ; j++){
    t = A[j];
    i = j-1;
    while((i>=0) && (A[i]>t)){
        A[i+1] = A[i];
        i = i-1;
    }
    A[i+1] = t;
}
```

Time Complexity Analysis :

(i) Best case ; [Like A | 10 | 20 | 30 |]

(ii) Worst case ; [Like A | 30 | 20 | 10 |]

(iii) Average case ; [Like A | 10 | 30 | 20 |]

Best case Time Complexity of Insertion Sort :

Line No.	Iteration	Time/Iteration	Time
L1	$(n-2+1)+1 = n$	c_1	$c_1 n$
L2	$n-1$	c_2	$c_2(n-1)$
L3	$n-1$	c_3	$c_3(n-1)$
L4	$n-1$	c_4	$c_4(n-1)$
L5	–	–	–
L6	–	–	–
L7	$n-1$	c_7	$c_7(n-1)$

Total running Time, $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$

$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 n - c_4 + c_7 n - c_7$

$= (c_1 + c_2 + c_3 + c_4 + c_7) n + (-c_2 - c_3 - c_4 - c_7)$

$= An + B$

$= O(n) \rightarrow$ Order of n

Worst Case Time Complexity of Insertion Sort:

A | 30 | 20 | 10 |

```
L1: for j = 2 to n
L2:     t = A[j]
L3:     i = j-1
L4:     while((i>=1) AND (A[i]>t))
L5:         A[i+1] = A[i]
L6:         i = i-1
        end while
L7:     A[i+1] = t
end for
```

Line no.	Iteration	Time/Iteration	Time
L1	$(n-2+1)+1 = n$	c_1	$c_1 n$
L2	$n-1$	c_2	$c_2(n-1)$
L3	$n-1$	c_3	$c_3(n-1)$
L4	$\sum_{j=2}^{n} j$	c_4	$c_4 \sum_{j=2}^{n} j$
L5	$\sum_{j=2}^{n}(j-1)$	c_5	$c_5\left(\sum_{j=2}^{n}(j-1)\right)$
L6	$\sum_{j=2}^{n}(j-1)$	c_6	$c_6\left(\sum_{j=2}^{n}(j-1)\right)$
L7	$n-1$	c_7	$c_7(n-1)$

Total running time, $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\sum_{j=2}^{n} j\right) + c_5\left(\sum_{j=2}^{n} (j-1)\right) + c_6\left(\sum_{j=2}^{n} (j-1)\right) + c_7(n-1)$

$= c_1 n + c_2(n-1) + c_3(n-1) + c_4(2+3+4+\ldots+n) + c_5(1+2+3+\ldots+(n-1)) + c_6(1+2+3+\ldots+(n-1)) + c_7(n-1)$

$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\{1+2+3+4+\ldots+n-1\} + c_5(1+2+3+\ldots+(n-1)) + c_6(1+2+3+\ldots+(n-1)) + c_7(n-1)$

$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left\{\frac{n(n+1)}{2} - 1\right\} + c_5\left\{\frac{(n-1)n}{2}\right\} + c_6\left\{\frac{(n-1)n}{2}\right\} + c_7(n-1)$

$= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left\{\frac{n^2}{2} + \frac{n}{2} - 1\right\} + c_5\left\{\frac{n^2}{2} - \frac{n}{2}\right\} + c_6\left\{\frac{n^2}{2} - \frac{n}{2}\right\} + c_7(n-1)$

$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + \frac{c_4 n^2}{2} + \frac{c_4 n}{2} - c_4 + \frac{c_5 n^2}{2} - \frac{c_5 n}{2} + \frac{c_6 n^2}{2} + \frac{c_6 n}{2} + c_7 n - c_7$

$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} + \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n + (-c_2 - c_3 - c_4 - c_7)$

$= An^2 + Bn + C$

$= An^2$

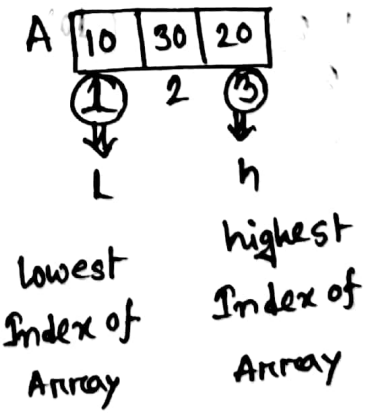
$= O(n^2)$

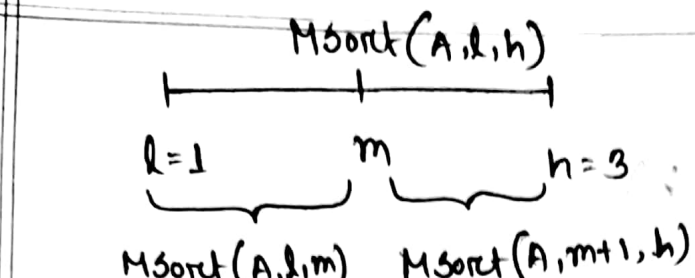
Line	Best case	Worst case	Average case
			$\frac{\text{Best + worst}}{2} = \frac{1+j}{2}$
L4	1	j	
L5	x	j-1	$\frac{1+j}{2} - 1$
L6	x	j-1	$\frac{1+j}{2} - 1$

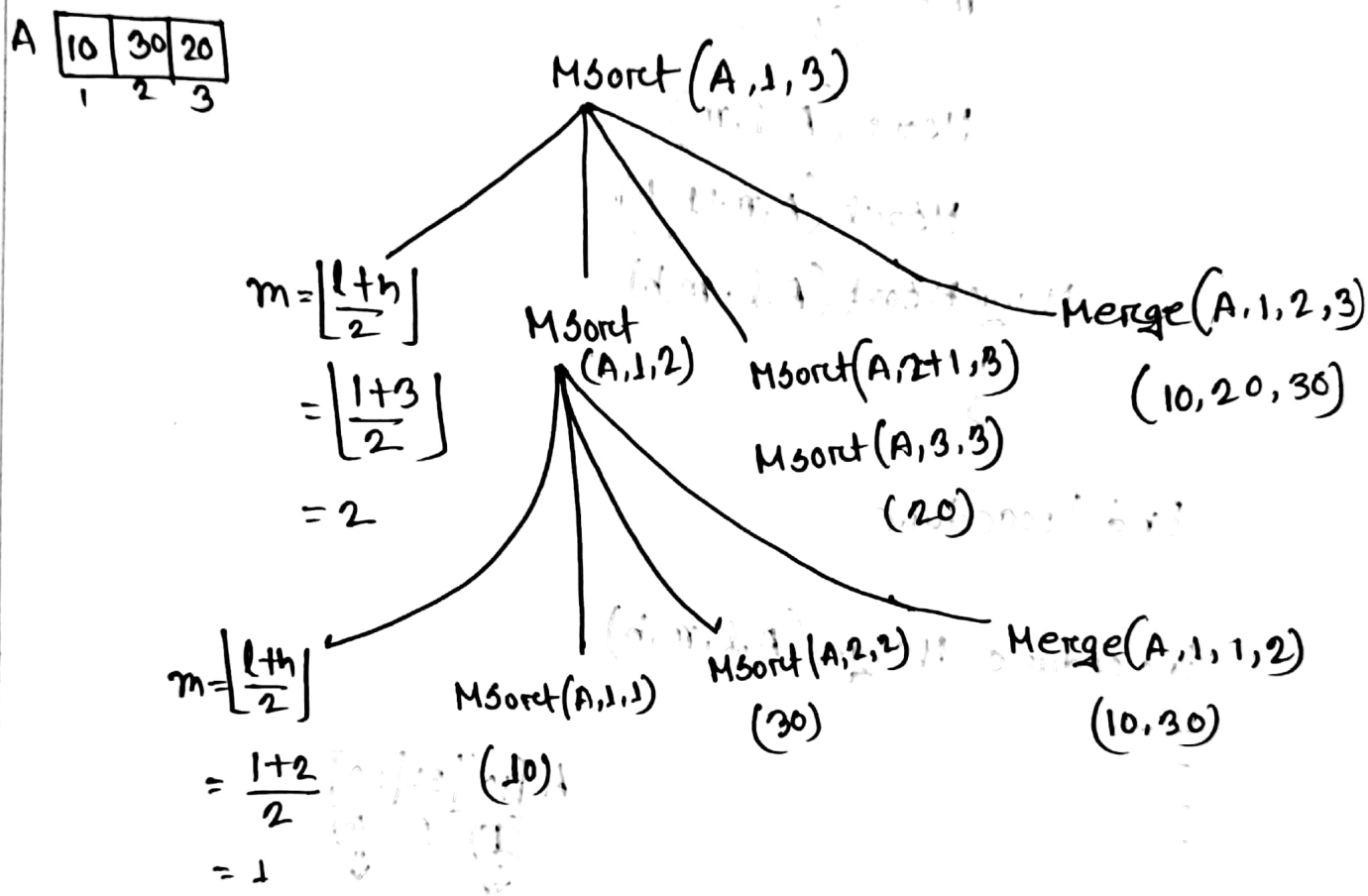
Merge Sort :

```
Procedure MSort (A, l, h)
    if (h > l) {
        m = ⌊(l+h)/2⌋
        MSort (A, l, m)
        MSort (A, m+1, h)
        Merge ~~MSort~~ (A, l, m, h)
    }
End Procedure

Procedure Merge (A, l, m, h)
    ---------
    ---------
    ---------
    ---------
    ---------
End Procedure
```



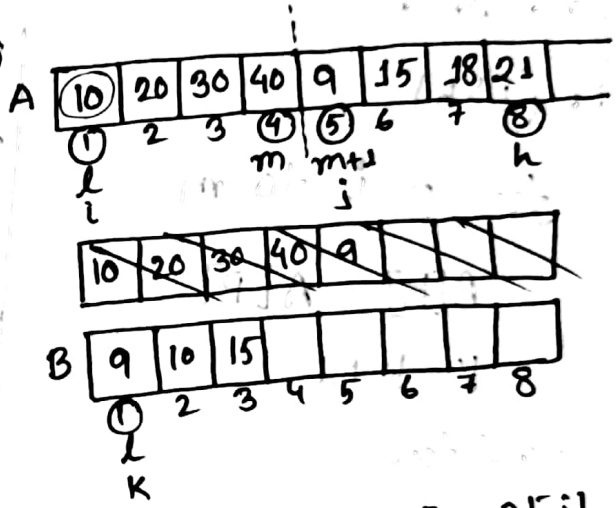

Msort(A,l,h)
l=1
m
h=3
Msort(A,l,m)
Msort(A,m+1,h)


A
10
30
20
1
2
3
Msort(A,1,3)
m=⌊(l+h)/2⌋
=⌊(1+3)/2⌋
=2
Msort (A,1,2)
Msort(A,2+1,3)
Msort(A,3,3)
(20)
Merge(A,1,2,3)
(10,20,30)
m=⌊(l+h)/2⌋
=(1+2)/2
=1
Msort(A,1,1)
(10)
Msort(A,2,2)
(30)
Merge(A,1,1,2)
(10,30)

```
Produce MSort (A, l, h)
    if (h > l) {
        m = ⌊(l+h)/2⌋
        MSort (A, l, m)
        MSort (A, m+1, h)
        Merge (A, l, m, h)
    }
End Produce
```

```
Produce Merge (A, l, m, h)
=========== }
            }
End Produce
```

i = l ; j = m+1 ; K = l



A[i] > A[j] ; B[K] = A[j]

⇒ 10 > 9 True ; B[1] = 9

j = j+1 = 5+1 = 6 ; K = K+1 = 1+1 = 2

i = i+1 = 1+1 = 2 ; K = K+1 = 2+1 = 3

A[i] > A[j] ; B[K] = A[j]

⇒ 10 > 15 False ; ⇒ B[2] = 10

j = j+1 = 6+1 = 7 ; K = K+1 = 3+1 = 4

A[i] > A[j] ; B[K] = A[j]

⇒ 20 > 15 True ⇒ B[3] = 15

j = j+1 = 7+1 = 8 ; K = K+1 = 4+1 = 5

A[i] > A[j] ; B[K] = A[j]

⇒ 20 > 18 True ; ⇒ B[4] = 18

```
i = l ; j = m+1 , k = l ;
while((i<=m) AND (j<=h))
 if (A[i] > A[j])
    B[k] = A[j] ; j = j+1 ; k = k+1 ;
 else B[k] = A[i] ; i = i+1 ; k = k+1 ;
end while
if (j>h)
  for p = i to m
     B[k] = A[p]
     k = k+1
  end for
else
    for p = j to h
       B[k] = A[p]
       k = k+1
    end for
for p = l to h
    A[p] = B[p]
end for
```

if j>h i---m A array to B

if i>m j---h A array to B

Copy all data of ~~Binar~~ B array to A array

Time Complexity Analysis:

$$T(n) = \begin{cases} p & n = 1 \\ 2T\left(\frac{n}{2}\right) + Cn + D & n > 1 \text{ and } n = 2^k \end{cases}$$

Time Complexity of C Code:

```
for(i=1; i<=n; i++){
    sum=0;
    for(j=1; j<=i; j++){
        sum = sum+i+j;
    }
    printf("%d", sum);
}
printf("%d", sum);
```

[Termination condition execute for True & False]

[i++ will be executed only for true value of the termination]

[In the outer loop any statement will execute for the true value of the condition]

Statement	Iteration	T/I	Time
$i = 1$	1	c_1	$c_1 * 1$
$i <= n$	True → n-1+1 = n, False → 1 } n+1	c_2	$c_2 * (n+1)$
$i++$	n	c_3	$c_3 * n$
$sum = 0$	n	c_4	$c_4 * n$
$j = 1$	$n * 1 = n$	c_5	$c_5 * n$
$j <= i$	$\frac{(n+1)(n+2)}{2} - 1$	c_6	$c_6 * \frac{(n+1)(n-2)}{2} - 1$
$j++$	$\frac{n(n+1)}{2}$	c_7	$c_7 * \frac{n(n+1)}{2}$
$sum = sum + i + j$	$\frac{n(n+1)}{2}$	c_8	$c_8 * \frac{n(n+1)}{2}$
printf("%d", sum)	n	c_9	$c_9 * n$
printf("%d", sum)	1	c_{10}	$c_{10} * 1$

Total running Time, $T(n) = c_1 * 1 + c_2 * (n+1) + c_3 n + c_4 n + c_5 n + c_6 \frac{(n+1)(n+2)}{2} - 1 + c_7 \frac{n(n+1)}{2} + c_8 \frac{n(n+1)}{2} + c_9 * n + c_{10}$

$= c_1 + c_2(n+1) + c_3 n + c_4 n + c_5 n + c_6\left(\frac{n^2}{2} + \frac{2n}{2} + \frac{n}{2} + 1 - 1\right) + c_7\left(\frac{n^2}{2} + \frac{n}{2}\right) + c_8\left(\frac{n^2}{2} + \frac{n}{2}\right) + c_9 n + c_{10}$

$$= c_1 + c_2 n + c_2 + c_3 n + c_4 n + c_5 n + c_6 \frac{n^2}{2} + c_6 \frac{2n}{2} + c_6 \frac{n}{2} + c_7 \frac{n^2}{2} + c_7 \frac{n}{2} + c_8 \frac{n^2}{2} + c_8 \frac{n}{2} + c_9 n + c_{10}$$

$$= c_6 \frac{n^2}{2} + c_7 \frac{n^2}{2} + c_8 \frac{n^2}{2} + c_2 n + c_3 n + c_4 n + c_5 n + c_6 n + c_6 \frac{n}{2} + c_7 \frac{n}{2} + c_8 \frac{n}{2} + c_9 n + c_{10} + c_1$$

$$= n^2 \left(\frac{c_6}{2} + \frac{c_7}{2} + \frac{c_8}{2} \right) + n \left(c_2 + c_3 + c_4 + c_5 + c_6 + \frac{c_6}{2} + \frac{c_7}{2} + \frac{c_8}{2} + c_9 \right) + c_{10} + c_{10}$$

$$= An^2 + Bn + C$$

$$= O(n^2)$$

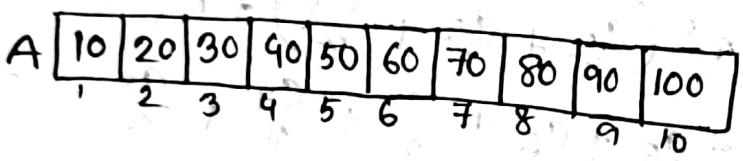
Ans.

Searching :

There are two types of searching

- Binary search
- Linear search

Binary Search:



Key = 70 Found 7
Key = 35 Not Found

low	high	mid = [(low+high)/2]	A[mid]=key	A[mid]>key	A[mid]<
1	10	m = 5	A[5] = 70	A[5] > 70	A[5] < 70
			=> 50 = 70 False	=> 50 > 70 False	=> 50 < 70
6 = mid+1	10	mid = 8	A[8] = 70	A[8] > 70	
			=> 80 = 70 False	=> 80 > 70 True	X
6	7 = mid-1	mid = 6	A[6] = 70	A[6] > 70	A[6] < 70
			=> 60 = 70 False	=> 60 > 70 False	=> 60 < 70 Tru
7 = mid+1	7	mid = 7	A[7] = 70	X	X
			=> 70 = 70 True		
			↳ "Found", mid ↳ 7		

Algorithm :

```
low = 1
high = n
while (low <= high)
    mid = ⌊(low + high) / 2⌋
    if (A[mid] = key)
      print "Found:", mid
      exit loop
    else if (A[mid] > key)
        high = mid - 1
    else if (A[mid] < key)
        low = mid + 1
end while
If (low > high)
print "Not found"
```

Linear Search:

A	10	6	15	9

No. of elements, $n = 4$

key = 15

$i = 1$ A[i] = key

$\Rightarrow A[1] = 15$

$\Rightarrow 10 = 15$ False

$i = 2$ A[i] = key

$\Rightarrow A[2] = 15$

$\Rightarrow 6 = 15$ False

$i = 3$ A[i] = key

$\Rightarrow A[3] = 15$

$\Rightarrow 15 = 15$ True → "Found", i → 3

$i = 4$ ✗

key = 12

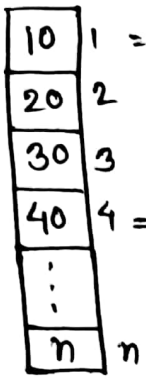
	A[i]=key
$i=1$	False
$i=2$	False
$i=3$	"
$i=4$	"

$i = 5$ → "Not Found" (if(i>n))

Algorithm :

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

Memory Allocation (One dimensional Array):



1 = base $\Rightarrow$ loc(A[1]) = known

$\longrightarrow$ loc(A[4]) = ??

$$loc(A[4]) = loc(A[1]) + 2 + 2 + 2$$
$$= loc(A[1]) + 2*3$$
$$= loc(A[1]) + 2*(4-1)$$

↓

for int, w = 2 bytes

In general,

$$loc(A[i]) = loc(A[1]) + w*(i-1)$$
$$= loc(A[base]) + w*(i-base)$$

$$loc(A[i]) = loc(A[base]) + w*(i-base)$$

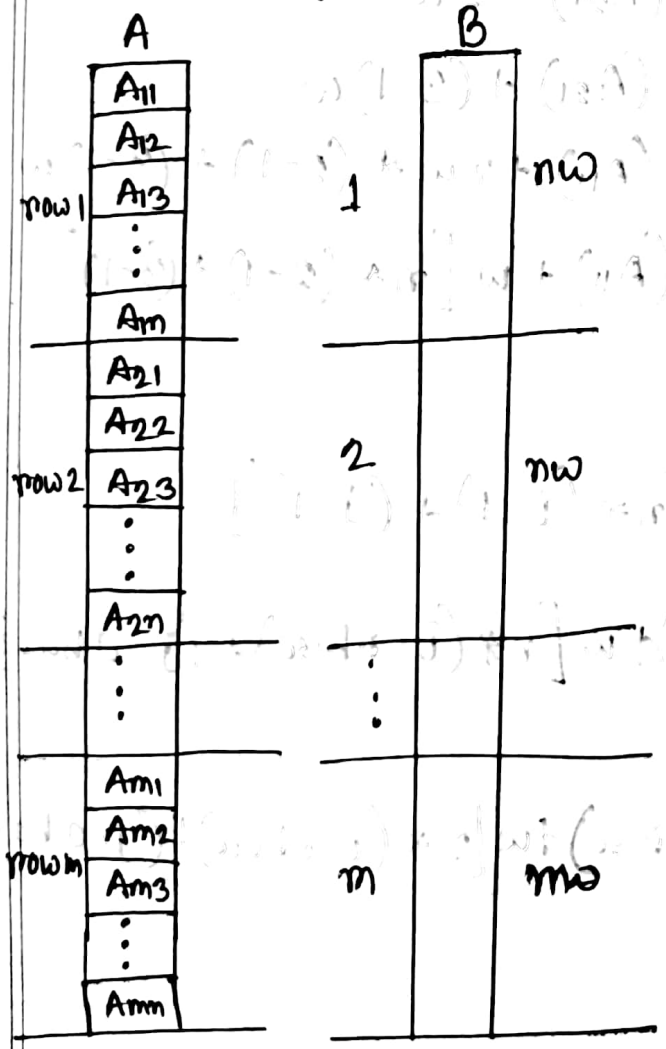
int	2 bytes
Long	4 bytes
float	4 bytes
Char	1 bytes
double	8 bytes

Memory Allocation (Two dimensional Array):

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2n} \\ \vdots & & & & \\ A_{m1} & A_{m2} & A_{m3} & \cdots & A_{mn} \end{bmatrix}$$

← row 1
← row 2
← row m

row-wise memory allocation



[one data width = w
n " " = nw]

[Starting of B[1] is similar of A[11] & Starting of B[2] is similar to A[21]]

From 1 dimensional Array memory allocation

$$loc(A[i]) = loc(A[base]) + w * (i - base)$$

$$\therefore loc(B[2]) = loc(B[1]) + nw * (2-1)$$

$$\Rightarrow loc(A_{21}) = loc(A_{11}) + nw * (2-1)$$

Again, $loc(A_{23}) = loc(A_{21}) + w + w$

$$= loc(A_{21}) + 2w$$

$$= loc(A_{21}) + (3-1)w$$

$$= loc(A_{11}) + nw * (2-1) + (3-1)w$$

$$= loc(A_{11}) + w[n * (2-1) + (3-1)]$$

In general,

$$loc(A_{ij}) = loc(A_{11}) + w[n * (i-1) + (j-1)]$$

$$= loc(A_{rbase, cbase}) + w[n * (i - rbase) + (j - cbase)]$$

$$\therefore loc(A[i][j]) = loc(A_{rbase, cbase}) + w[n * (i - rbase) + (j - cbase)]$$

⊛ Ques:

double A[50][100];

If $loc(A[10][15]) = (5CDFE)_{16}$, find $loc(A[40][50])$.

Ans: Given;

$$loc(A[10][15]) = (5CDFE)_{16}$$

We know,

$$loc(A[i][j]) = loc(A_{rbase, cbase}) + w[n*(i-rbase)+(j-cbase)]$$

$$\Rightarrow loc(A[40][50]) = loc(A[10][15]) + 8[100*(40-10)+(50-15)]$$

$$= (5CDFE)_{16} + 8[100*30+35)]$$

$$= (5CDFE)_{16} + 8[100*65] \quad 8[3000+35]$$

$$= (5CDFE)_{16} + (24280)_{10}$$

$$= 380414 + 24280$$

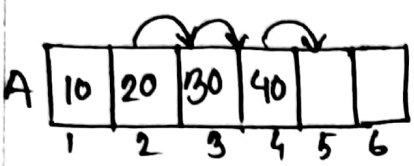
$$= (404694)_{10}$$

$$= (62CD6)_{16}$$

Linked List :

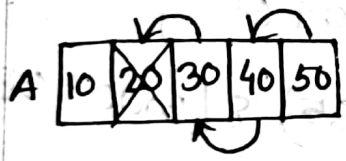
Array

1. Insertion difficult



To add a new item we have to move all the elements.

2. Deletation difficult



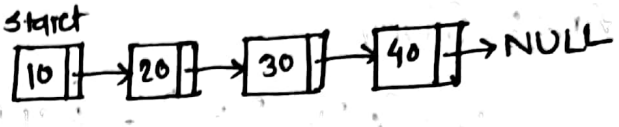
After deletion we have to move data forward

3. Binary search Easy

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

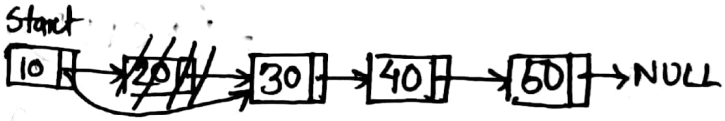
Linked List

1. Insertion Easy



To insert a new item don't need to move other items.

2. Deletion Easy



After deletion don't need to move data. just linked it

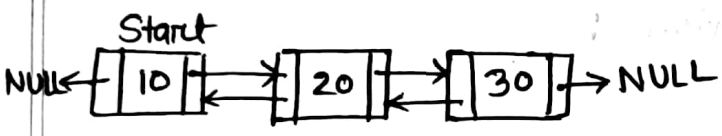
3. Binary search difficult

[No index]

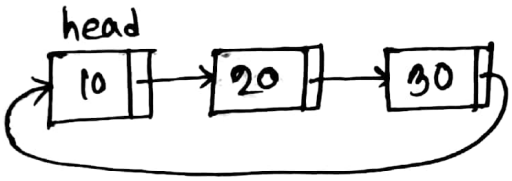
Types of Linked List :

1. Linear or Singular Linked List

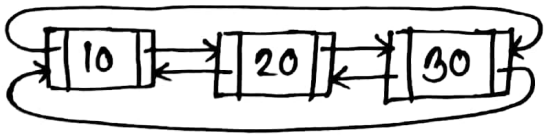
2. Doubly or Two-way Linked List



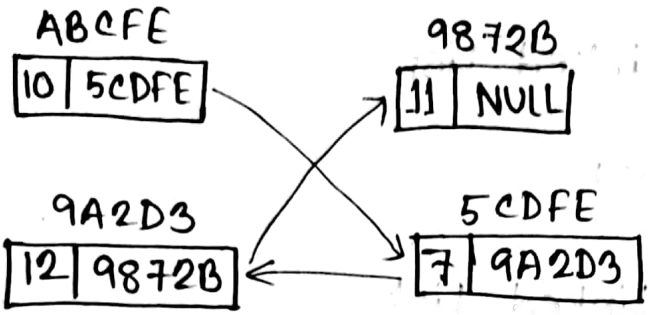
3. Linear circular Linked List



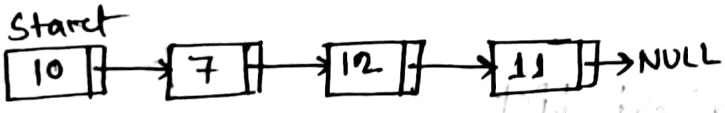
4. Doubly circular Linked List



Actual Memory Representation of Linked List:



Pictorial Representation :



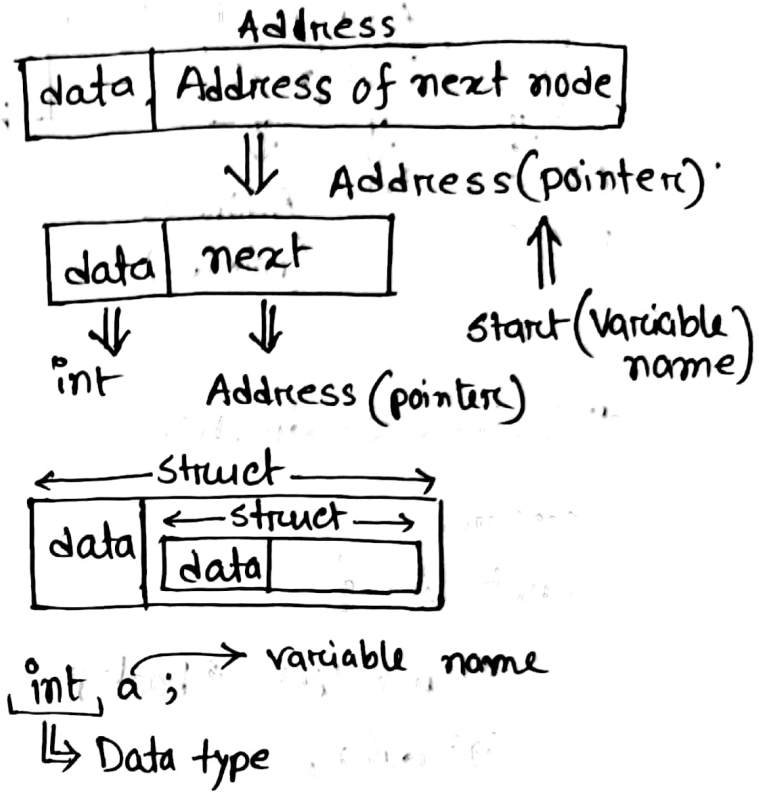
Variable declaration

```
#include <stdio.h>

struct list {
        int data;
        struct list *next;
};
typedef struct list node;

int main()
{
    node *start;
     // code here
    return 0;
}
```

Linear Linked List:



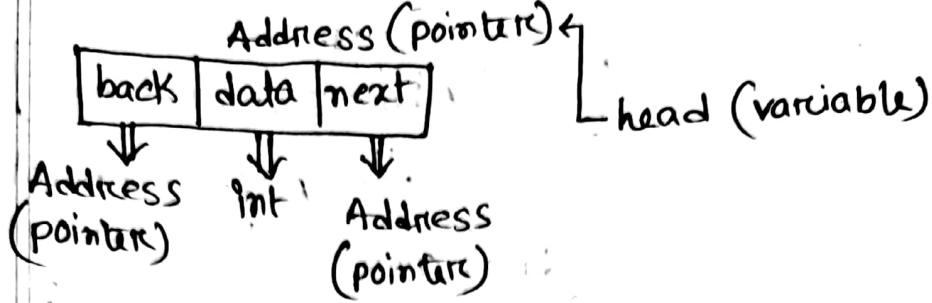
int a; → variable name

↳ Data type

int, float, char, double → Std. data type;

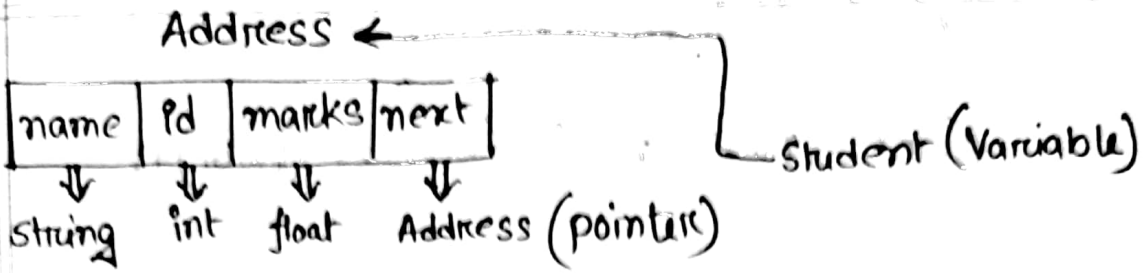
node → User data type.

⊛ Doubly linked list




```
#include <stdio.h>

struct list {
    struct list *back;
    int data;
    struct list *next;
};
typedef struct list node;
int main() {
    node *head;
    // code
    return 0;
}
```

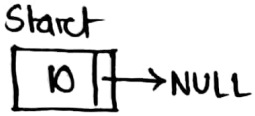



```c
#include <stdio.h>
struct list {
    char name[50];
    int id;
    float marks;
    struct list *next;
};
typedef struct list node;
int main()
{
    node * head; student;
    //code
    return 0;
}
```

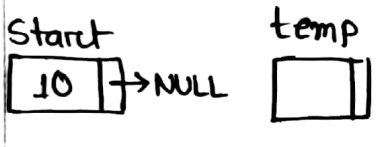
Code for linked list :

start = (node*) malloc(size of (node));

start → data = 10 ;

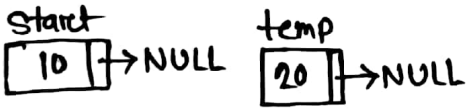


start → next = NULL ;

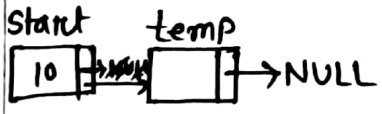


temp = (node*) malloc (size of (node));

temp → data = 20 ;



temp → next = NULL ;



start → next = temp ;

Write effect of each statement :

super = (node*) malloc (size of (node));

temp = "

temp1 = "

super → data = 10;

temp → data = 20;

temp1 → data = 30;

super → next = temp1;

temp1 → next = temp;

temp → next = NULL;

super → next = super → next → next;

free (temp1);

tempsuc = (node*) malloc (size of (node));

tempsuc → data = 35;

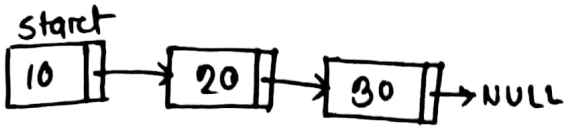
tempsuc → next = super → next;

super → next = tempsuc;

Creation of Linear Linked List :

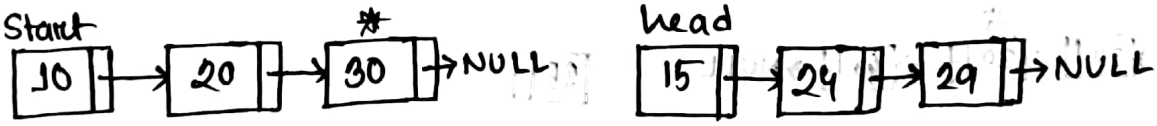
```
#include <stdio.h>
#include <stdlib.h>
struct list{
        int data;
        struct list *next;
};
typedef struct list node;
int main()
{
    node *start, *prev , *temp ;
    int i;
    start = (node*) malloc(size of (node));
    start → data = 10;
    start → next = NULL;
    prev = start;
    for (i = 20 ; i<=30 ; i = i+10) {
        temp = (node*)malloc(size of (node));
        temp → data = i;
        temp → next = NULL ;
        prev → next = temp ;
        prev = temp;
    }
return 0;
}
```

Display a Linear linked list :



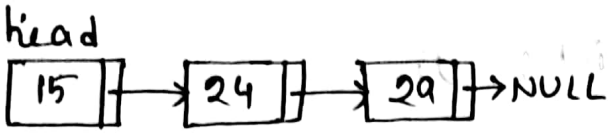

```
printf ("Create linked list: ");
temp = start;
while (temp != NULL) {
    printf ("%d ", temp->data);
    temp = temp->~~next~~; data;
}
```

Concatenation of two linked list :




```
temp = start
while (temp->next != NULL){
    temp = temp->next;
}
temp->next = head ;
```

How to create Linear circular Linked List from Linear link list :




```
temp = head;
while (temp -> next != NULL) {
        temp = temp -> next;
}
temp -> next = head;
```

To insert an element (newitem = 35)


```
temp = start;
while((temp -> next != NULL) && (temp -> next -> data <= newitem))
{
    temp = temp -> next;
}
temp1 = (node *) malloc (size of (node));
temp1 -> data = newitem;
temp1 -> next = temp -> next;
temp -> next = temp1;
```

To delete a node from linked list (item = 30)


```
temp = start;
while ((temp->next != NULL) && (temp->next->data != item))
{
    temp = temp->next;
}
temp1 = temp->next;
temp->next = temp1->next;
free (temp1);
```

Algorithm Notation :

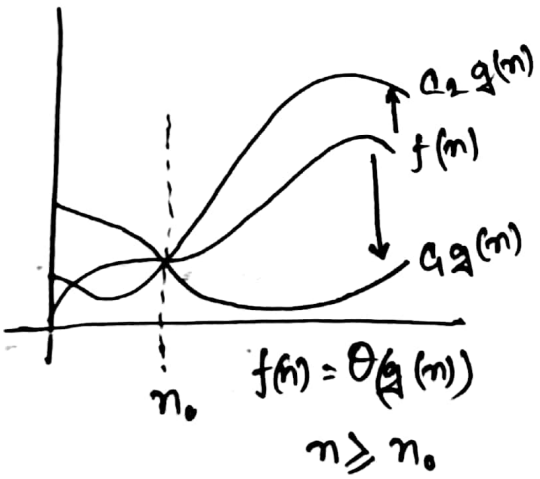
- θ notation $\left(\overbrace{c_1 g(n) \leq f(n)}^{LS} \leq \underbrace{c_2 g(n)}_{RS}\right)$

- O notation $\left(\underbrace{f(n) \leq c g(n)}_{RS}\right)$

- Ω notation $\left(\underbrace{c g(n) \leq f(n)}_{LS}\right)$

θ - notation :

$f(n) = \theta(g(n))$ iff there exists three positive constants c_1, c_2, n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $n \geq n_0$



Average case Analysis

⊛ If $f(n) = 3n-4$, Prove that $f(n) = \Theta(n)$

Ans: Given,

$f(n) = 3n-4$

$g(n) = n$

From the definition of Θ

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\Rightarrow c_1 n \leq 3n-4 \leq c_2 n$

$\Rightarrow c_1 \leq 3 - \frac{4}{n} \leq c_2$ [Divided by n]

Left hand inequality:

$n = 1, \quad 3 - \frac{4}{n} = -1$

$n = 2, \quad 3 - \frac{4}{n} = 1$

$n = 3, \quad 3 - \frac{4}{n} = \frac{5}{3}$

$\vdots$

$n = \infty, \quad 3 - \frac{4}{n} = 3$

$c_1 \leq 3 - \frac{4}{n}$

$\therefore c_1 = 1$ where $n \geq 2$

$1 \leq 3 - \frac{4}{n}$, $\because n = 1$ False

$1 \leq 3 - \frac{4}{n}$; $n = 2$ True

$1 \leq 3 - \frac{4}{n}$; $n = 3$ True

$\vdots$

$1 \leq 3 - \frac{4}{n}$; $n = \infty$ True

Right hand inequality :

$3 - \frac{4}{n} \leq c_2$

$\therefore c_2 = 3$ where $n \geq 1$

$n = 1, \quad 3 - \frac{4}{n} = -1$

$n = 2, \quad 3 - \frac{4}{n} = 1$

$n = 3, \quad 3 - \frac{4}{n} = \frac{5}{3}$

$n = \infty, \quad 3 - \frac{4}{n} = 3$

Now,

$3 - \frac{4}{n} \leq 3 \; ; n = 1$ True

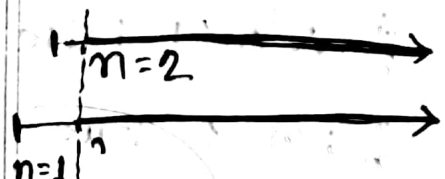
$3 - \frac{4}{n} \leq 3 \; ; n = 2$ True

$3 - \frac{4}{n} \leq 3 \; ; n = 3$ True

$3 - \frac{4}{n} \leq 3 \; ; n = \infty$ True

After combining both inequalities

$c_1 = 1, \quad c_2 = 3$ where



So, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

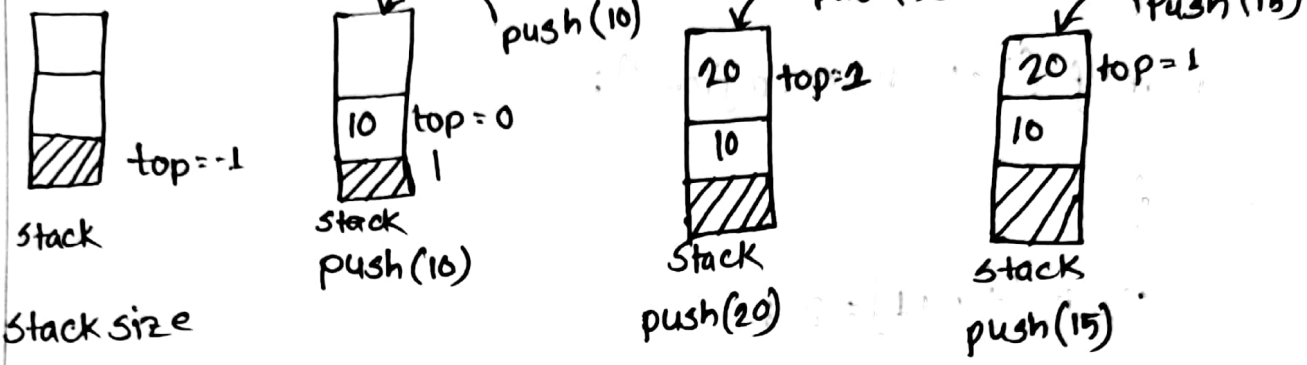
$\Rightarrow 1 g(n) \leq f(n) \leq 3 g(n)$ where $n \geq$

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

[proved]

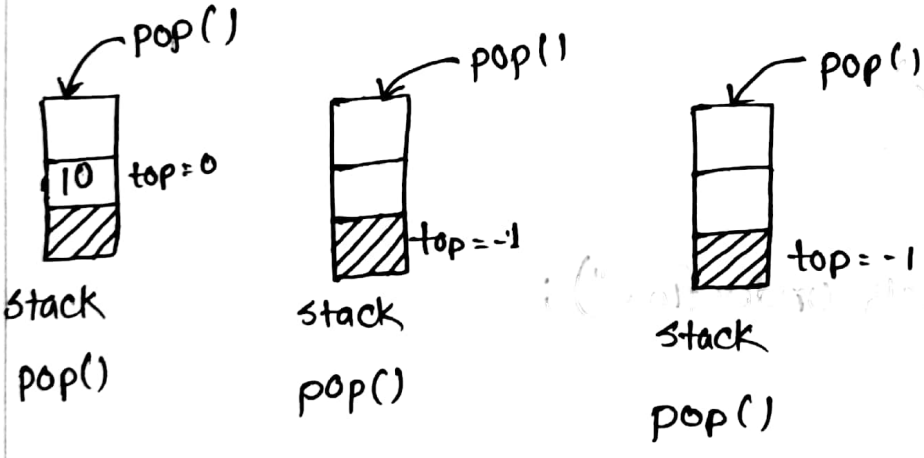
Stack:

A stack is a LIFO (Last in First Out) data structure.



Stack size
$m = 2$

MSG: "Stack Overflow"
OR
"Stack full"



MSG: "Stack underflow"
OR
"Stack empty"

Stack Implementation Using Array :

push (stack ⟸ x)

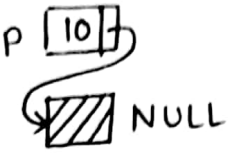
```
if (top+1 ≥ m)
  printf ("stack overflow");
else {
    stack [top+1] = x ;
    top = top+1;
}
```

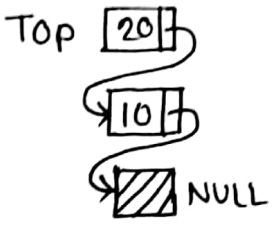
pop (stack ⟹ x)

```
if (top == -1)
  printf ("stack underflow");
else {
      x = stack [top];
      printf ("%d", x);
      stack [top] = NULL;
      top = top - 1;
}
```

Stack using Linked List :

Initial : 

push(10) : 

push(20) : 

pop( ) : 

pop ( ) : 

pop ( ) : MSG: "Stack underflow"

push (stack ⇐ x)

```
temp = (node*) malloc (size of (node));
temp→data = x ;
temp→next = top ;
top = temp;
```

pop (stack ⇒ x)

```
~~if (top == N~~
if (top == NULL)
   printf ("stack underflow")
else {
      temp = top ;
      top = temp→next ;
      free ( temp) ;
}
```

Queue:

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

Operation:

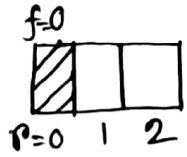
- Insertion (Enqueue) (Queue $\Leftarrow$ x) $\leftarrow$ rear
- Deletion (Dequeue) (Queue $\Rightarrow$ x) $\leftarrow$ front

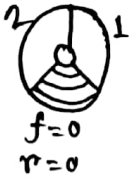
* Ques: show the status of queue of size 2 using array of the following operations:

- Insert(10), Insert(20), Delete(), Insert(30), Insert(25) Delete(), Delete(), Delete().

Ans:

Initial :

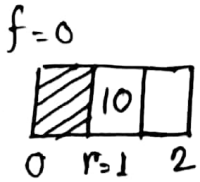




$S = (r+1) \% (m+1)$

$= (0+1) \% (2+1) = 1 \% 3 = 1$

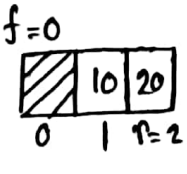
Insert (10) :



$S = (r+1) \% (m+1)$

$= (1+1) \% (2+1) = 2 \% 3 = 2$

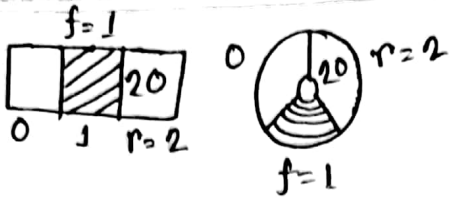
Insert (20) :



$S = (r+1) \% (m+1)$

$= (2+1) \% (2+1) = 3 \% 3 = 0$

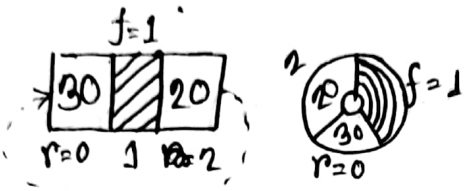
Delete ( ) :



$f = (f+1) \% (m+1)$

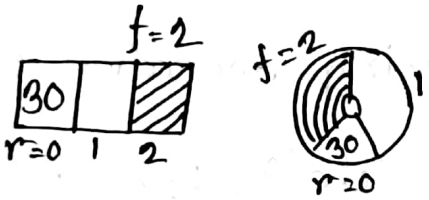
$= (0+1) \% (2+1) = 1 \% 3 = 1$

Insert (30):

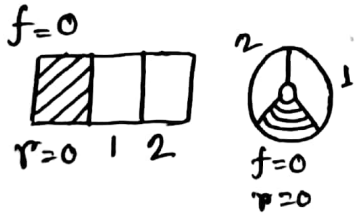


Insert (25): "Queue overflow" $[S = (r+1) \% (m+1) = (0+1) \% (2+1) = 1 \% 3 = 1 = f]$

Delete ( ):



Delete ( ):



Delete ( ): "Queue underflow"

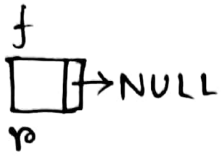
Insert (Queue ⇐ x):

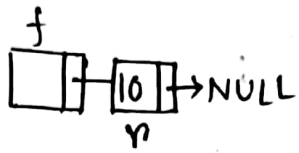
```
S = (r+1) % (m+1)
if (S == f)
    printf ("Queue overflow");
else {
    Queue [S] = x ;
    r = S;
}
```

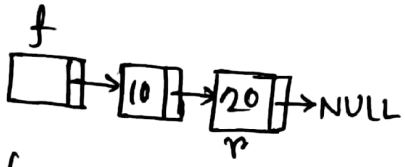
Delete (Queue ⇒ x):

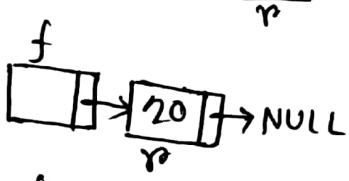
```
if (f == r)
  printf ("Queue Underflow")
else{
    f = (f+1) % (m+1) ;
    Queue [f] = NULL;
}
```

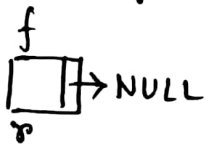
Queue implementation using linked list:

Initial : 

Insert (10): 

Insert (20): 

Delete ( ): 

Delete ( ): 

Delete ( ): "Queue underflow".

Insert (Queue ⇐ x):

```
temp = (node*) malloc (size of (node));
if (temp == NULL)
        printf ("No memory");

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;
}
```