Program = (Data structure + Algorithm)

Data Structure

a method of organizing data in order to faciliate access and modification

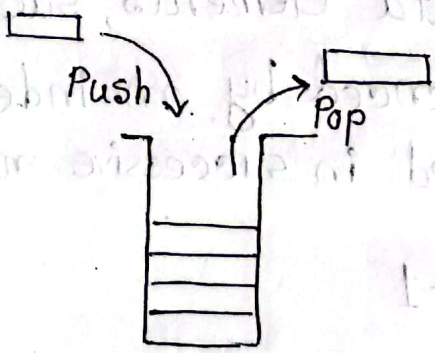
Types of data Structure:

① Array

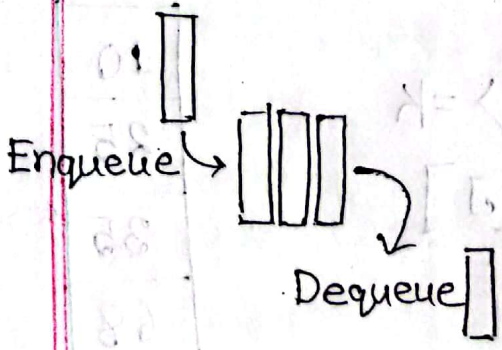
23	4	6	15	7
0	1	2	3	4

Index:

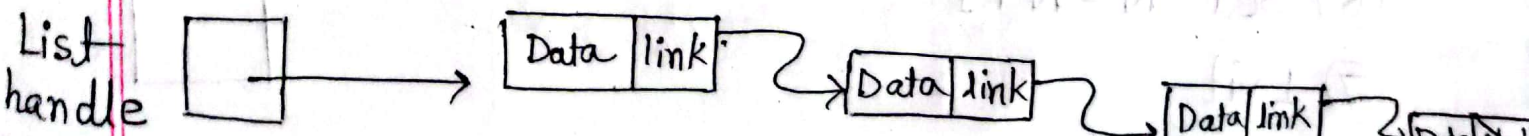
② Stack



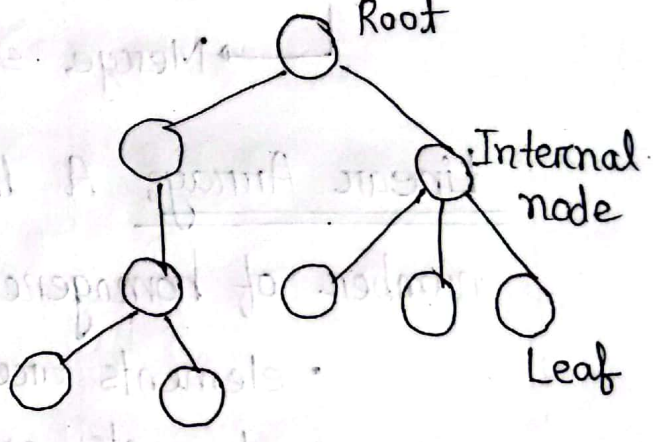
③ Queues



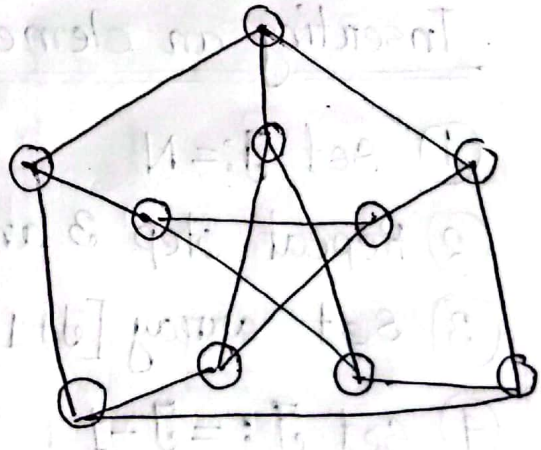
④ Linked lists:



④ Trees



⑤ Graphs



- **Sorting (4)**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort

- **Searching (2)**
  - Linear search
  - Binary search (sorted)

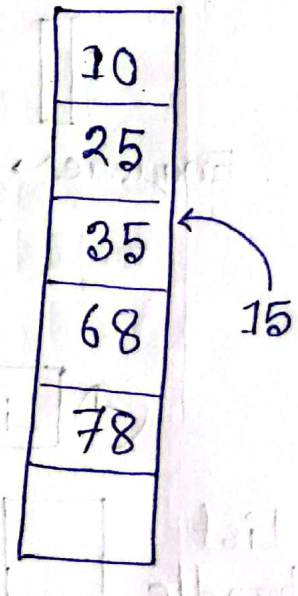
Linear Array: A linear Array is a list of a finite number of homogeneous data elements, such that →

- elements are referenced by an index set
- elements are stored in successive memory locations

Length of array = UB − LB + 1

Inserting an element: K = 3, N = 5

1. Set J := N
2. Repeat step 3 and 4, while J >= K
3. Set array[J+1] := array[J]
4. Set J := J − 1
5. Set array[K] := ITEM
6. Set N = N + 1
7. Exit.



array subscripting = array indexing

Deleting an element:

Initial, K = 3, N = 6

10
25
35
68
78
15

X

1. Set ITEM = array [K]
2. Repeat for J = K, to N-1, Set array[J] = array[J+1]
3. Set N = N-1
4. Exit

Searching Algorithm:

□ Linear Search

1. K := 1 and LOC = 0
2. Repeat step 3 and 4 while K <= N and LOC = 0
3. If ITEM = array [K], then LOC = K
4. Set K = K + 1
5. If LOC = 0, Then write :- "ITEM is not Found"
6. Else, Write:- ITEM found at LOC
7. Exit

~~Linear~~ Complexity of linear search:

Worst case : $C(n) = n$

Average case : $1\frac{1}{n} + 2\frac{1}{n} + 3\frac{1}{n} \cdots + n\frac{1}{n}$

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

$= \frac{n(n+1)}{2}\frac{1}{n} = \frac{n+1}{2}$ (half of the element)

Bubble Sort:

Pass - 1

9 6 2 12 11 9 3

6 9 2 12 11 9 3

6 2 9 12 11 9 3

6 2 9 12 11 9 3

6 2 9 11 12 9 3

6 2 9 11 9 12 3

6 2 9 11 9 3 12

Pass - 2

6 2 9 11 9 3 12

2 6 9 11 9 3 12

2 6 9 11 9 3 12

2 6 9 11 9 3 12

2 6 9 9 11 3 12

2 6 9 9 3 11 12

2 6 9 9 3 11 12

Algorithm:

1. Repeat step 2 and 3 for I = 1 . . . . N
2. Repeat 3 for J = 1 . . . N
3. If array[J] > arr[J+1] then swap(arr[J], arr[J+1])
4. Exit.

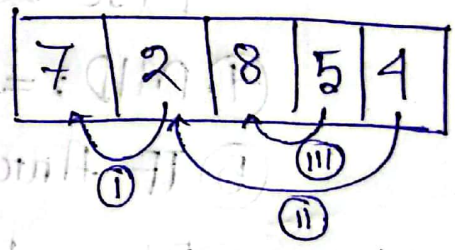
Code:

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

Selection Sort:

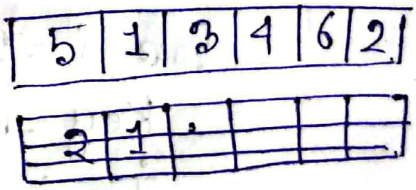
```
for (i = 0; i < arr_size - 1; ++i) {
    min = i;
    for (j = min+1; j < arr_size - 1; j++) {
        if (a[min] > a[j])
            min = j;
    }
    temp = a[i];
    a[i] = a[min];
    a[min] = temp;
}
```

- Find min and positi
- Swap it with 1st positi



Algorithm:

1. Repeat step 2 to 6 for I = 1 . . . . N
2. Set min := I
3. Repeat steps 4 and 5 for J = I+1 . . . . N
4. If arr[J] < ~~list~~ arr [Min] then
5. Set Min := J
6. swap (arr[I], arr[min])
7. Exit



Binary Search:

① Set LOC = 0 ; MID := INT (BEG + END)/2

② Repeat steps 3 and 4 while BEG <= END and Array [MID] != ITEM

③ If ITEM > Array [MID]
Set BEG := MID + 1
Else Set END := MID - 1

④ MID := INT (BEG + END)/2

⑤ IF Array [MID] = ITEM set LOC := MID
Else LOC := NULL;

⑥ Exit

Insertion sort: Most common sort used by card players. It has two parts — sorted, unsorted

Code:

10	20	30	40

```
for i = 2 ... n {
    Key = A [i]
    j = i - 1
while (j > 0) and (A[j] > key) {
        A[j+1] = A[j]
        j = j - 1
}
A[j+1] = key
```

Stack

- It is a linear data structure where all additions and deletations are made at one end, the top.
- Stacks are known as LIFO (Last in First out) data structure.
- Stack has only two operation – push and pop
- Two stack errors –
  1. Underflow : pop an empty stack
  2. Overflow : push onto an already full stack
- Uses of Stack
  1. Any sort of nesting
  2. Evaluting Arithmetic expressions
  3. Implementing function or method calls
  4. Keeping track of previous choices

Push

1. If Top = MaxSTK then Print Overflow and return
2. Set Top := Top+1
3. set Stack[Top] := Item
4. Return

Pop

1. If Top = -1 then print: Underflow and Return.
2. set Item := Stack[Top]
3. set top := top - 1
4. Return.

polish notation ↑ reverse polish notation ↑

~~Infix~~ Prefix	Infix	Postfix
+AB	A+B	AB+
*AB	A*B	AB*
+2*35= =+2 15 = 17	2+(3*5) = 2+15 = 17	235*+= =2 15+ = 17
*+235= =*55 = 25	(2+3)*5 = ~~7~~5*5 = ~~25~~ 25	2 3+5*= =55* =25
~~paren~~ no parentheses needed	parentheses needed	no parentheses needed

Infix to Prefix :- ((A+B)*(C+D))

= (+ A B * (C+D))

~~= *+AB *(c+CD~~

= *+ AB (C+D)

= *+ AB +CD

Infix to Postfix :- (((A+B)*C)-((D+E)/F))

= AB + C * DE+F/ -

Infix → Postfix

(((A+B)*(C-E))/(F+G))

Symbol	stack	Output
	(	
(	((	
(	(((	
(	((((	
A	((((	A
+	((((+	A
B	((((+	A B
)	(((	AB+
*	(((*	AB+
C	(((* (	AB+
(	(((* (	AB+C
-	(((* (-	AB+C
E	(((* (-	AB+CE
)	(((*	AB+CE-
)	((	AB+CE-*
/	((/	AB+CE-*
(	((/ (	AB+CE-*
F	((/ (	AB+CE-*F
+	((/ (+	AB+CE-*F
G	((/ (+	AB+CE-*FG
)	((	AB+CE-*FG+
)	(	AB+CE-*FG+/

P.T.O

)		AB+CE-*FG+/

Ⓒ 4*(5+3) - 24/6

Infix (Q)	stack	Postfix (P)
	(	
4	(	4
*	(*	4
(	(*(	4
5	(*(	4, 5
+	(*(+	4, 5
3	(*(+	4, 5, 3
)	(*	4, 5, 3, +
-	(-	4, 5, 3, +, *
24	(-	4, 5, 3, +, *, 24
/	(-/	4, 5, 3, +, *, 24
6	(-/	4, 5, 3, +, *, 24, 6
)		4, 5, 3, +, *, 24, 6/-

Postfix calculation : slide

Postfix calculation: 5, 6, 2, +, *, 12, 4, /, -,)

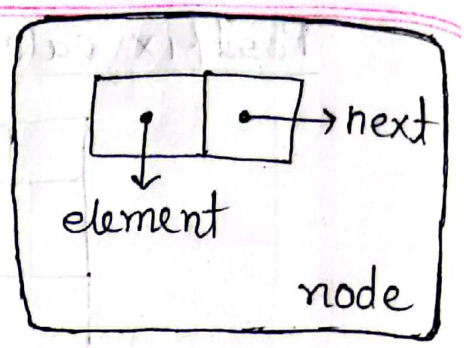
Postfix	Stack
5	5
6	5,6
2	5,6,2
+	5,8
*	40
12	40,12
4	40,12,4
/	40,3
-	37
)	37

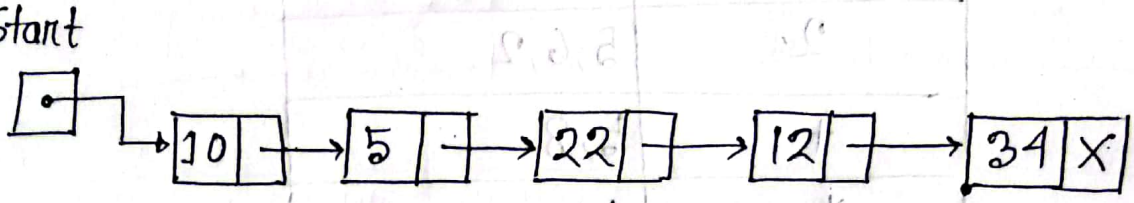
Infix to postfix → ((a+b-c)*d-(e+f))

(	(	
(	((	
(	(((	
a	(((	a
+	(((+	a
b	(((+	ab
-	(((+-	ab+
c	(((-	ab+c
)	((	ab+c-
*	((*	ab+c-
d	((*	ab+c-d
-	((-	ab+c-d*
(	((-(	ab+c-d*
e	((-(	ab+c-d*e
+	((-(+	ab+c-d*e
f	((-(+	ab+c-d*ef
)	((-	ab+c-d*ef+
)	(	ab+c-d*ef+-
)		ab+c-d*ef+

Linked List (One-way linked list)

Definition: A linear collection of data elements represented by node which has an information part and link part.





- LIST - name of the linked list
- START - points to the first node of the linked list
- PTR - points to the node that is currently being process
- INFO[PTR] - value of the current node
- PTR:-LINK[PTR] - moves the pointer to next node

Operation on Linked List

1. Insert
   - at 1st position
   - at last position
   - into ordered list
2. Delete
3. Traverse
4. Copy linked list

Types of Linked List

- Singly linked list
- Circular linked list
- Doubly linked list

efficient operations: insert, delete, split, join

inefficient: search (time consuming)

C structure to represent a node

```
struct node
{
    int info
    struct node *link;
}
```

Traversing, searching → slide

1D – Array	Linked List
① Fixed size	① Dynamic size
② Insertion and Deletions are inefficient; usually shifting needed	② Insertion and Deletions are efficient; No shifting needed.
③ Random access that is efficient indexing	③ No random access, only sequential access to the elements
④ No memory waste if array is full. otherwise may result much memory waste	④ Extra storage needed for reference; uses exactly as much memory as it needs

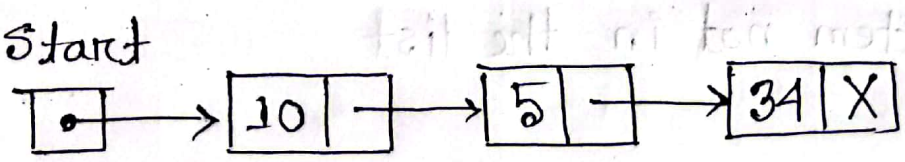
Linked list and its memory representation

(i) A linked list can be maintained in memory using two linear arrays → Info, Link

Info [K] – represents information part of a node

Link [K] – " the next pointer field of a node

(ii) The beginning of the node list denoted by the variable Start.



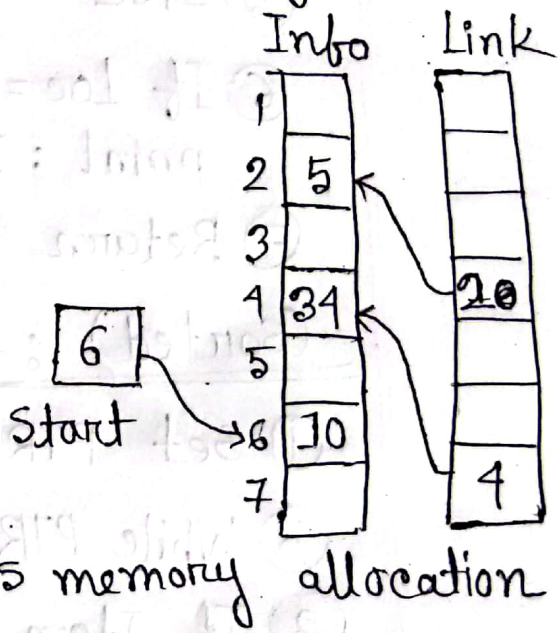


Figure: linked list and its memory allocation

Counting Node List:

1. Set PTR := Start and Num=0
2. While PTR ≠ Null repeat steps 3 and 4
3. Num := Num+1
4. PTR := Link [PTR]
5. Return

Searching :- (Unsorted) :-

1. Set PTR := Start and Loc := Null
2. While PTR ≠ Null do steps 3 to 5
3. If Item = Info [PTR] then
4. Loc = PTR , print : Item found and return
5. Else PTR := Link [PTR]
6. If Loc = Null , then
   print : Item not in the list
7. Return

(Sorted) :-

1. Set PTR := Start and LOC := Null
2. While PTR ≠ Null do steps 3 to 7
3. If Item = Info [PTR]
4. then LOC := PTR , print : Item found and return
5. else if Item > Info [PTR]
6. then PTR := Link [PTR]
7. else Exit.
8. If LOC = NULL then Print : Item is not in the list.
9. Return

Queue

⇒ A queue is a FIFO (First in First out) data structu accomplished by inserting at one end (rear) and deleting from the other (front)

enqueue – insert

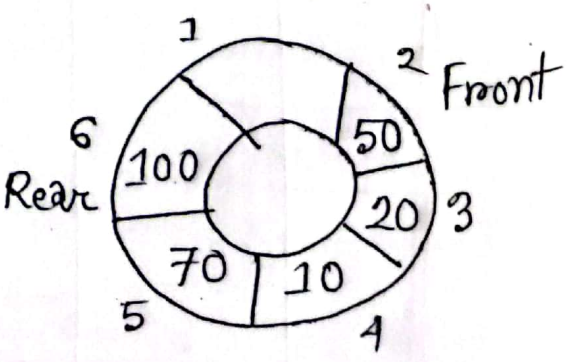
dequeue – delete

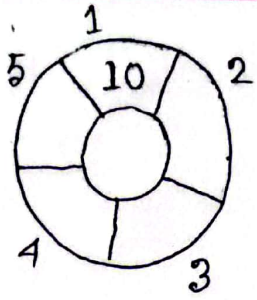
- empty queue
  - rear = -1
  - front = -1
- enqueue
  - rear = rear+1
- dequeue
  - front = front+1

Insert operation page-26	Deletetion operation page-27
① If rear = N, then print: Overflow and Return ② set rear := rear+1 ③ set Queue[rear] := Item ④ Return	① If Front = n+1, then print: Underflow and retur ② set front item := Queue[fro ③ set front := front+1 ④ Return

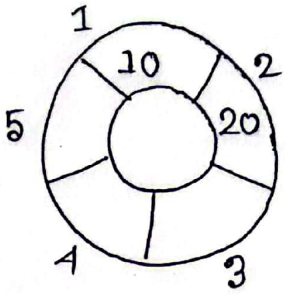
Circular Queue: a non-linear (circular) queue

Insertion: slide page-26
Deletetion: slide page-27
28, 29, 30 page.

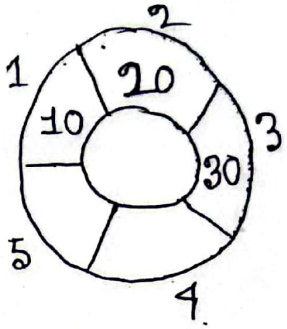




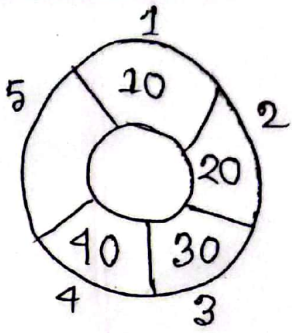
Rear = 1
Front = 1



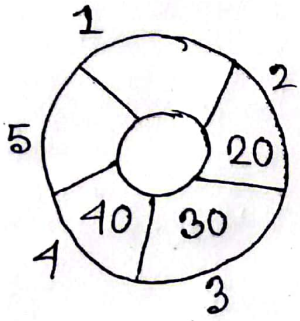
Rear = 2
Front = 1



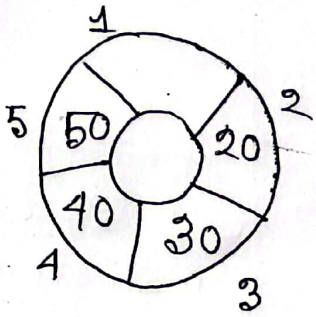
Rear = 3
Front = 1



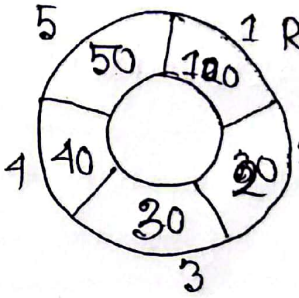
Rear = 4
Front = 1



Rear = 4
Front = 2



Rear = 5
Front = 2

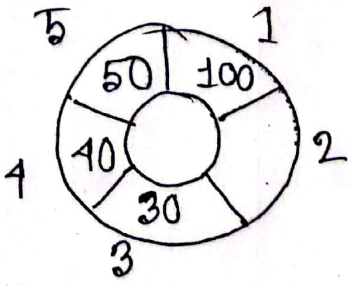


Rear = 1
Front = 2

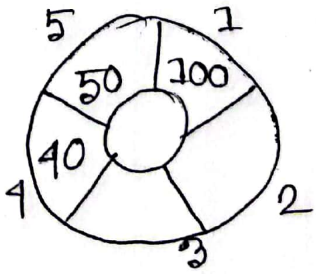
- Insert 10, when Rear=1 and Front = 2

  Ans: As Front = Rear+1, so queue overflow

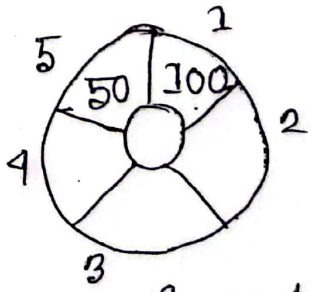
- Delete :



Rear = 1
Front = 3



Rear = 1
Front = 4



Rear = 1
Front = 5