Basic Data Structures

1. Array
2. String
3. Stack
4. Queue
5. Linked List

BY: Md Sadman Hafiz
SWE, SUST

Sub :

Day

Time : Date : / /

Array, Records, Pointers

AI-1 Array Traversing

LA is - Linear Array, Lower Bound LB, up. b. UB

1. Initialize Set K = LB
2. Repeat step 3 and 4 while $K \leq UB$
3. Apply Process to LA[K]
4. Set K = K+1
   End of Step 2 loop
5. Exit.

Sub :

Day

Time : Date : / /

Insert And Delete

Al-2 Insert (LA, N, K, ITEM)
↑ Length
1 ≤ K ≤ N position of item.

1. Set J = N.
2. Repeat Step 3 and 4 while J >= K.
3. [Move Jth element] Set LA[J+1] = LA[J]
4. Set J = J-1
5. Insert element LA[K] = Item.
6. Set N = N+1
7. Exit

Time complexity

Sub :

Day

Time : Date : / /

AL-3 Delete (LA, N, k, ~~Item~~)

1. Set Item = LA[k]
2. Repeat for J = k to N-1 ;
   Set LA[J] = LA[J+1]
   End of loop.
3. Set N = N-1
4. Exit.

Sub :

Day

Time : Date : / /

Linked list

→ Dynamically Allocated
→ Size can be vary at run time.

Diff between Array and Linked list

Li. Array	Linked List
→ Deletion and insertion are difficult	→ easy.
→ insertion requires data movement	→ don't
→ space wasted	→ Not
→ can not be reduced to requirement	→ Can
→ To avail/reach element same time is required.	→ different time
→ consequtive memory location	→ may or not
→ We can reach there directly	→ to reach particular node, you need to go through all those nodes that comes before that node

Sub :

Day

Time : Date : / /

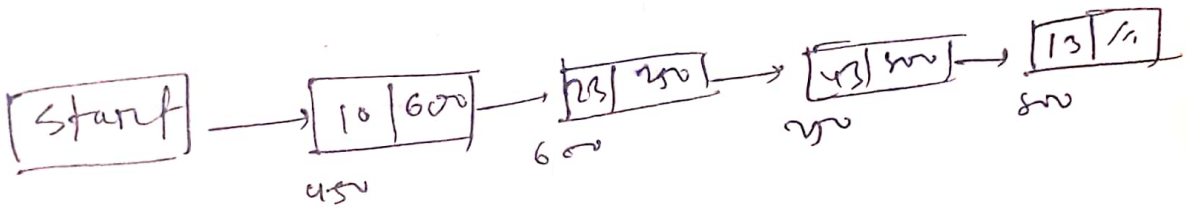
1. Why linked list is better?

- It is relatively expensive to insert and delete elements in an array. Since an array usually occupies a block of memory space, one cannot simply double or tripple the size of an array when additional space is required. On the other hand, in a linked list, in memory each element contains a link or pointer which contains the address of the next element. Thus successive elements in the list need not occupy adjacent space in memory. This will make it easier to insert or delete elements in the list.

Sub :

Day
Time : Date : / /

Single Linked List



#implementation of single linked list

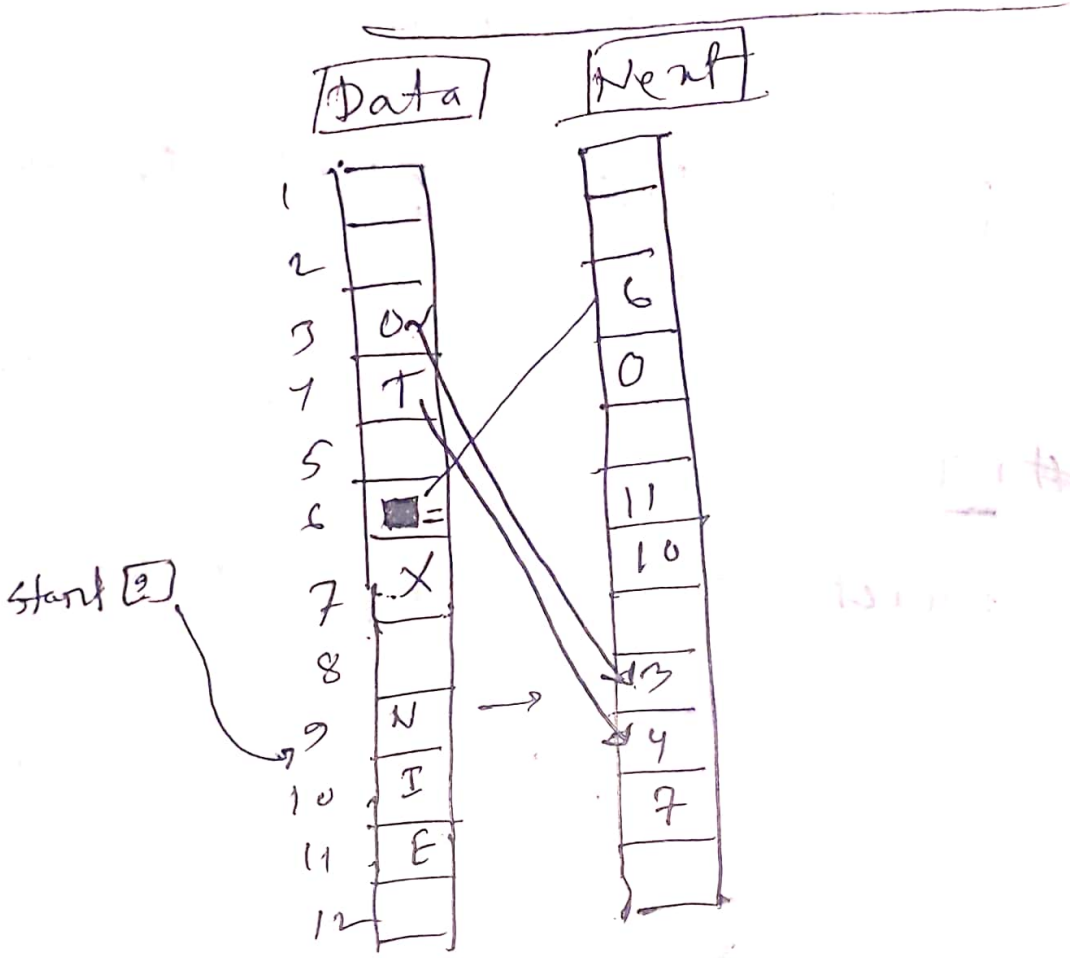
```
struct node {   type 1 member 1;
                type 2 member 2;
                . . . .
                struct node *link;
            }
```

Sub :

Day

Time : Date : / /

Representation of linked list in a memory



Here, Start = 9; Data [9] = 'N'; is the 1st element

Next [9] = 3 Data [3] = 'O' " 2nd "

Next [3] = 6 Data [6] = '=' " 3rd "

Next [6] = 11 Data [11] = 'E' " 4th "

Next [11] = 7 Data [7] = 'X' " 5th "

Next [7] = 10 Data [10] = 'I' " 6th "

Next [10] = 4 Data [4] = 'T' " 7th "

Next [4] = 0, Null value, list has ended.

Sub :

Page :

Time :

Date : / /

Traversing a Linked List

1. Set PTR = START.
2. while PTR ≠ NULL
   - (a) Apply Process to Info[PTR]
   - (b) Set PTR = Link[PTR] (PTR now points to next)

   End of loop
3. Exit.

Print

1. Set PTR = START.
2. while PTR ≠ NULL
   - (a) Write Info[PTR]
   - (b) Set PTR = Link[PTR]

   End of loop.
3. Exit.

Sub :

Day

Time : Date : / /

Searching

Unsorted List

1. Set PTR = START.
2. while PTR ≠ NULL
   - (a) If item = INFO[PTR], then set LOC = PTR, and exit.
   - (b) Else, set PTR = LINK[PTR] (next)

   End of loop.
3. [Search is unsuccessful] Set LOC = NULL

4. Exit.

Overflow and Underflow.

→ Overflow: when AVAIL = NULL and there is an insertion

→ Underflow: when START = NULL, there is a deletion.

Sub :

Day

Time : Date : / /

Sorted List

1. Set PTR = START
2. While PTR ≠ NULL
   If Item = Info[PTR],
   Set Loc = PTR, and exit.
   Else. If ITEM < Info[PTR].
   Set PTR = Link[PTR]
   Else Set Loc = NULL and exit.
   End of loop.
3. Set Loc = NULL.
4. Exit.

Day

Sub : Time : Date : / /

Insertion

T.C = O(1)

- Insertion in the beginning
- empty list
- at the end
- between the list nodes.

1. Inserting at the beginning of the list

~~Struct~~

INSFIRST (INFO, LINK, START, AVAIL, ITEM)

1. If AVAIL = NULL, write: overflow and Exit
2. [Remove first node from AVAIL]
3. ~~Set INFO[NEW] =~~ Set NEW = AVAIL and AVAIL = LINK(AVAIL)
3. Set INFO[NEW] = ~~Set~~ ITEM
4. Set LINK[NEW] = START
5. Set START = NEW.
6. Exit.

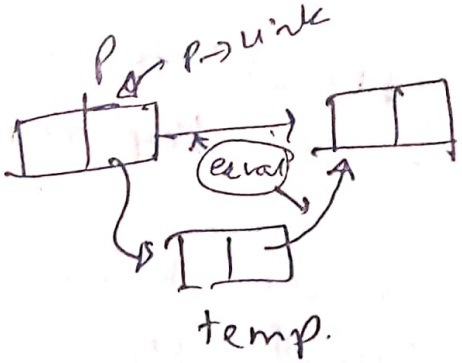
Sub :

Day

Time : Date : / /

2 Inserting after a given node

1. If avail = NULL, write : overflow, exit.
2. (Remove first node from avail list)
   Set New = Avail, Avail = Link [Avail]
3. Set Info [New] = Item.
4. If loc = Null.
   (a) Set Link [New] = Start
   (b) Set Start = New
   else,
   (a) Set Link [New] = Link [loc]
   (b) Link [loc] = New.
5. Exit.



temp->link = P->link

P->link = temp

Sub :

Day

Time : Date : / /

inserting into a sorted linked list.

$2 \rightarrow 5 \rightarrow 7 \rightarrow 10 \rightarrow 15$

insert 9 = $2 \rightarrow 5 \rightarrow 7 \rightarrow 9 \rightarrow 10 \rightarrow 15$

Algorithm:

(1) If linked list is empty then make the node as head and return it

(2) If the value of the node to be inserted is smaller than the value of the head node, then insert the node at the start and make it head.

(3) In a loop, find the appropriate node after which the input node (9) is to be inserted. To find the appropriate node start from the head, keep moving until you reach a node GN (10 in the diagram). The node before GN is (7).

(4) insert the node (9) after the appropriate node (7).

Sub :

Day

Time : Date : / /

Deleting

To delete a node from the linked list, we need to do following steps.

(1) Find the previous node of the node to be deleted.

(2) Change the next of the previous node and set prev_node → next = nodeto_delete → next

(3) Free the memory of the node to be deleted.

time complexity $O(n)$

Sub :

Day

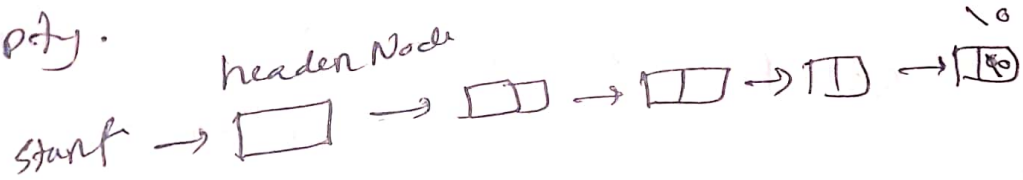
Time : Date : / /

Header linked list

① Grounded Header linked list

It is a list whose last node contains the Null pointer. In the header linked list, the start pointer always points to the header node.

start → next = NULL Indicates that the grounded header linked list is empty.



② circular

Last node b points back to the header node.

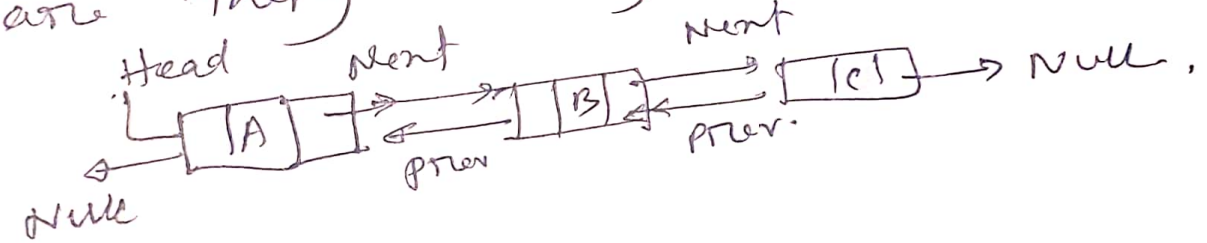
Sub :

Day

Time : Date : / /

Two way linked list

A ~~dok~~ doubly linked list contains an extra point, typically called previous pointer, together with next pointer and data which are there in singly linked list.



Sub :

Day

Time : Date : / /

Mainly is inputted on Laptop and it is also removed → undo button

back button

Data

Stacks, Queue, Recursion

→ Stacks follow LIFO (last in first out)

→ Abstract datatypes containing element in linear order.

→ operations are allowed only at the ends.

→ Insertion, deletion allowed only one end.

→ Last in first out.

Sub :

Day

Time : Date : / /

push operation

push (Stack, TOP, MAXSTK, ITEM)

① If TOP = SIZE, print: overflow and return

② Set TOP = TOP+1

③ Set stack[TOP] = ITEM

④ Return.

Pop (Stack, TOP, Item)

① If top = 0, print underflow and return

② Set ITEM = STACK[TOP]

③ Set TOP = TOP-1

④ Return.

Sub :

Day

Time : Date : / /

Use of Stack

Polish notation

1924, Polish Mathematician Jan Lukasiewicz

* Infix $P+Q \wedge R + S * T / (U-V)$

infix → prefix ± $P + Q \wedge R + S * T / [-UV]$

$= P + [\wedge QR] + S * T / [-UV]$

$= P + [\wedge QR] + [* ST] / [-UV]$.

$= + [P \wedge QR] + [/ * ST - UV]$.

$= + + P \wedge QR / * ST - UV$.

Day

Sub : Time : Date : / /

→ postfix $P + Q \wedge R + S * T / (U - V)$

$= P + [QR\wedge] + [ST*] / [UV-]$

$= [PQR\wedge +] + [ST*UV-/]$.

$= PQR\wedge + ST*UV-/+$.

postfix → Infix

$PQR\wedge + ST*UV-/+$

$= P\ Q\wedge R + (S*T)(U-V)/+$

$= P + Q\wedge R \quad S*T/U-V +$

$= P + Q\wedge R + S*T/U-V$.

prefix → Infix $++P\wedge QR/*ST-UV$.

$= ++P\ Q\wedge R / (S*T)\ (U-V)$

$= +\ P + Q\wedge R \quad S*T/U-V$.

$= P + Q\wedge R + S*T/U-V$.

Ans.

Sub :

Day

Time : Date : / /

Postfix → Prefix

PQR^ + ST * UV- / +

= P^QR + *ST - UV / +

= + P^QR / *ST-UV +.

= + + P^QR / *ST-UV .

prefix → postfix + + P^QR / *ST-UV .

= + + P Q^R^ / S * T * UV-

= + PQR^+ ST* UV- /

= PQR^+ ST* UV-/ +

.

Sub :

Day

Time : Date : / /

Evaluating Postfix Expression using stack

Steps:

1. Create a stack to store operands
2. Scan the given Expression and the
   (a) If the element is a number, push it into the stack
   (b) If the element is an operator, pop operands for the operator from the stack. Evaluate the operation and the push the result back to the stack
3. Number in the stack is answer

Sub :

Day
Time : Date : / /

Example:

Let the given expression be 2 3 1 * + 9 -

(1) Scan '2' → push.

(2) '3' → push, '2 3'

(3) '1' → '2 3 1'

(4) '*' → pop, 3, 1, 1 * 3 again push. 3

stack '2 3'

(5) '+' → pop 2, 3. 2 + 3 = 5 push -

stack '5'

(6) '9' → push, '5 9'

(7) '-' → pop '5, 9' 5 - 9 = - 4.

Ans.

Sub :

Day

Time : Date : / /

Infix to Postfix conversion

Step-0 Tokenize the infix expression, i.e. store each element of an infix expression into a list

Step 1 : If operand (Number) → add it to postfix.

Step-2: If left paran. "(" → push it on the stack.

Step-3 : Right paran ")", pop out and add to postfix all operator till first parenthesis. Discard both left and right paran.

Step-4: If an operator, pop and add to postfix these operators that have higher or equal precedence than the scanned operator.

→ push the scanned symbol on the stack.

Sub :

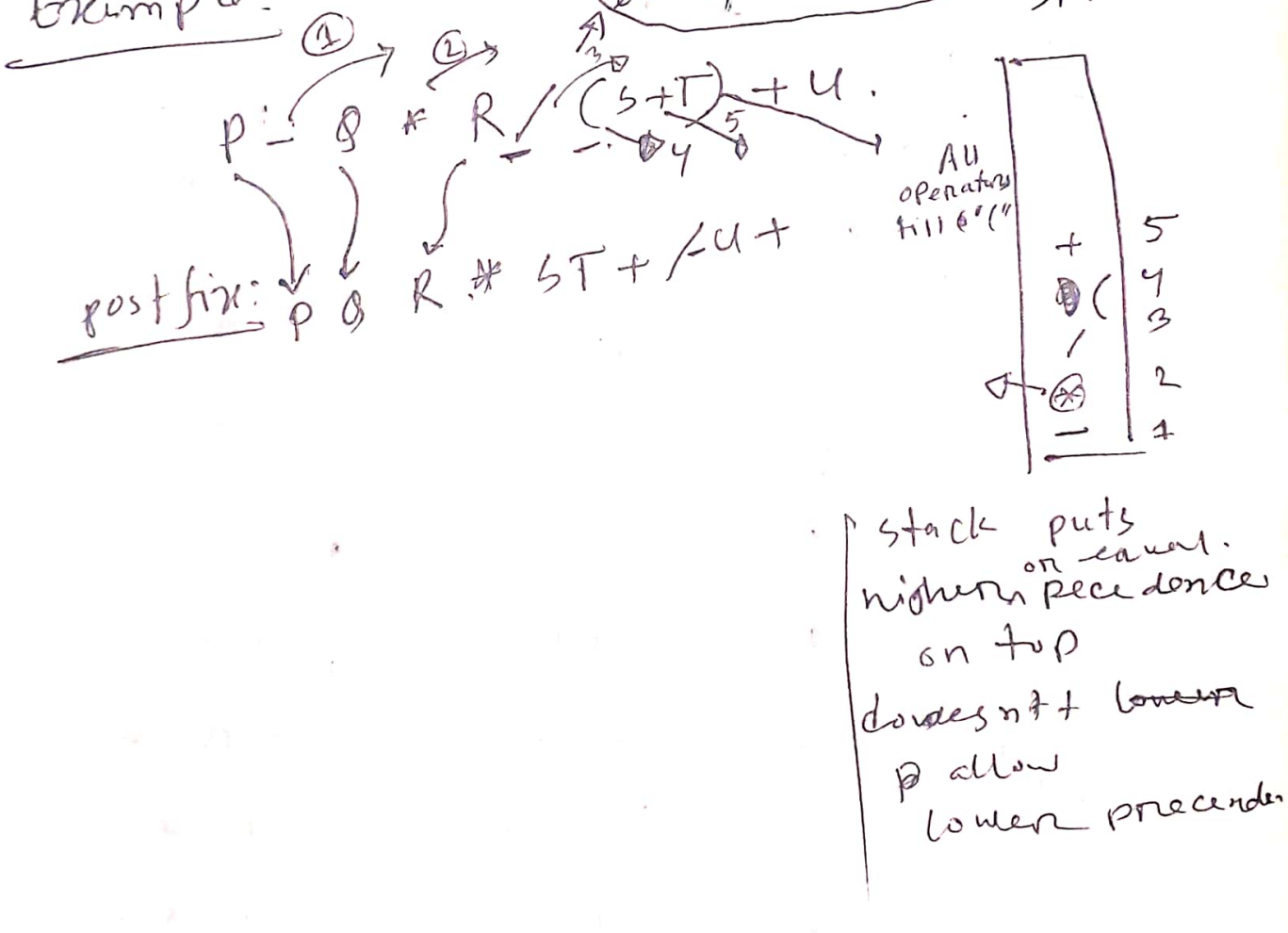
Day

Time : Date : / /

Step-5 ~~If the token is an operant~~

After scanning all the symbol of infix. pop and add to postfix. all the operator remaining on stack

Example:



Sub :

Day

Date :

Example-2

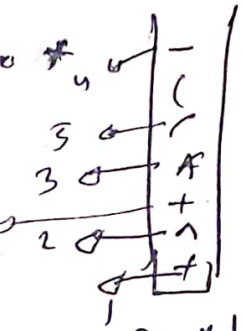
$P + Q \uparrow R + S * T / (U - V)$

lower than ↑ so, ↑ will be popped, again equal to ↑ popped.

equal to *

$P\ Q\ R \uparrow + S\ T * U\ V - / +$

1 2 3 4 5 6



Example-3 $A + (B * C - (D / E \uparrow F) * G) * H +$

popped till (

Infix	stack	postfix
A		A
+	+	A
(	+ (	A
B	+ (	AB
*	+ (*	AB
C	+ (*	ABC
-	+ (-	ABC *
(	+ (- (	ABC *
D	+ (- (	ABC * D
/	+ (- (/	ABC * D

Sub :

Day

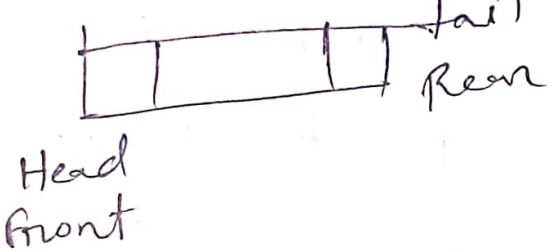
Time : Date : / /

Infix	stack	postfix
E	+(-(/	ABC*DE
^	+(-(/^	ABC*DE
F	+(-(/^	ABC*DEF
)	+(-	ABC*DEF^/
*	+(-*	ABC*DEF^/
G	+(-*	ABC*DEF^/G.
)	+	ABC*DEF^/G*-
*	+*	ABC*DEF^/G*
H	+*	ABC*DEF^/G*-H
		ABC*DEF^/G*-H*+

Ans.

Add → last (Rear)
Remove → first (front)

Queue:



FIFO
First in
First out

Algorithm: → insert

En Queue: (Que, N, Front, Rear, Item)

1. [Que already filled ?]
   If Front = 1 and Rear = N, or if
   FRONT = REAR + 1, then write overflow
2. [Find of Rear]
   If Front = Null, then set Front = 1, Rear = 1
   Else if Rear = N, then set Rear = 1
   Else Rear = Rear + 1;
   [End of if structure]
3. Set Que [Rear] = Item.
4. Return.

Insertion complexity O(1)

Deletion

Sub :

Day

Time : Date : / /

Q Delete (Queue, N, Front, Rear, Item)

1. [Queue already empty ?]
   If Front = NULL, then : Underflow
2. Set Item = Queue [Front]
3. ~~Front = Front + 1~~ ;
   [Find the new value of Front]
   If Front = Rear,
   Set Front = NULL, Rear = Null.
   Else if Front = N,
   Set Front = 1
   Else, Set Front = Front + 1

deletion complexity

O(n)

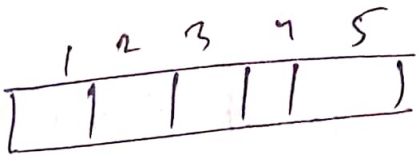
Sub :

Day
Time : Date : / /

Example:

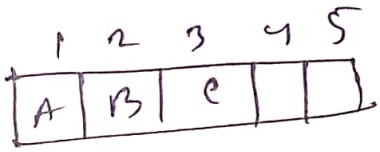
(1) Initially empty:

Front = 0
Rear = 0

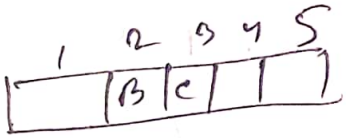


(2) A, B, C inserted

Front 1
Rear 3

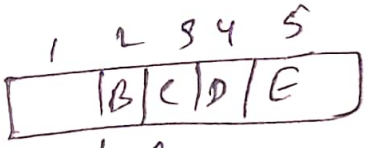


(3) A deleted (dequeue)



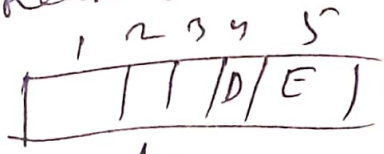
Front 2
Rear 3.

(4) D, E inserted (Enqueue)



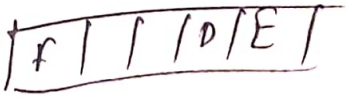
Front 2
Rear 5.

(5) B, C deleted



Front 4,
Rear 5

(6) F inserted



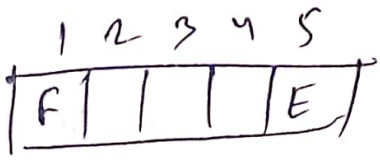
Front 4
Rear 1.

Sub :

Day

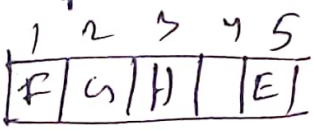
Time : Date : / /

7. D deleted



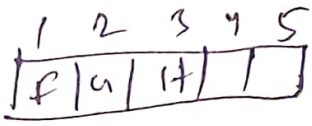
F = 5
R = 1.

8. G. H inserted



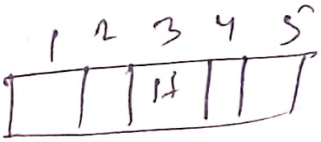
F = 5
R = 3

9. E deleted



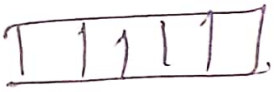
F = 1
R = 3

10. F & G deleted



F = 3
R = 3

(1). D, H "



F = 0
R = 0