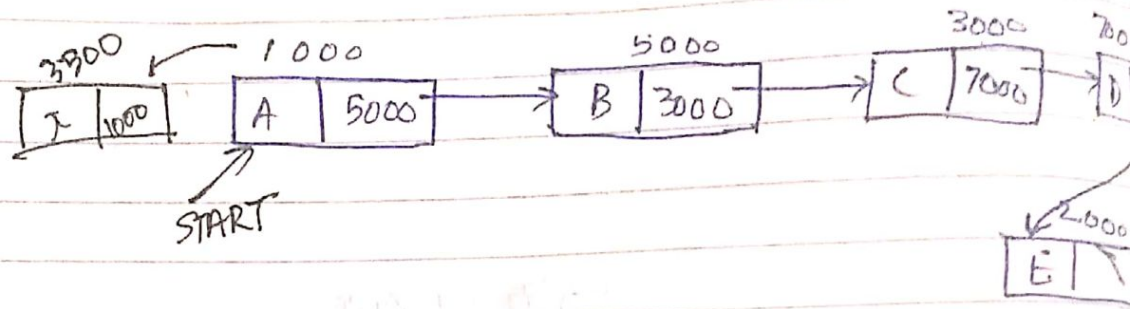
Sagar
Date:
Page:

Link List

Linked List is a data structure which consist of nodes. Every node has atleast 2 fields One of which contains the data and the other contains the address of next node.

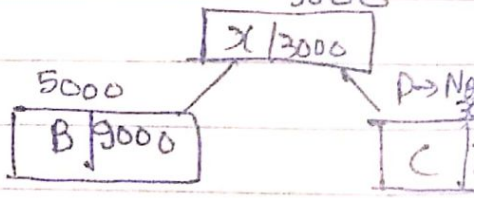


ALGORITHM Ins Beg (START, x)
BEGIN:

struct node *P;
P = GetNode()
P → info = x
P → Next = START
START = P

END;

Time Comp = O(1)
Space = 3
P, Out
2



ALGORITHM InsAfter(P, x)
BEGIN:

Q = GetNode()
Q → Info = x
Q → Next = P → Next
P → Next = Q

END;

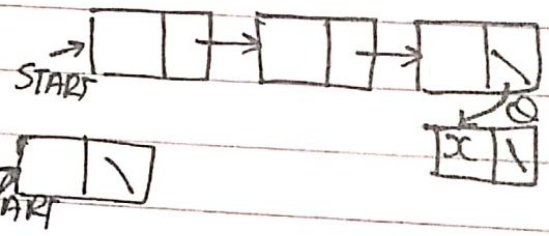
Ex
Time Comp = 4 O
Space = 3 O(

Q is adress Q→Next = adress
P is adress P→Next = adress

ALGORITHM Traverse(START)
BEGIN:
P=START
WHILE P!=NULL DO
WRITE (P→info)
P=P→Next
END;

Time = 2N+1 = O(N)
Space = O(1)

ALGORITHM InsEnd (START, x)
BEGIN: IF START==NULL THEN
InsBeg (START, x)
ELSE
P=START
WHILE P→Next!=NULL DO
P=P→Next
Q=GetNode()
Q→Info=x
P→Next=Q
Q→Next=NULL



Same InsAfter(P,x)

Time complexity
N-1+3 = O(N)
Space
1+3 = 4 = O(1)

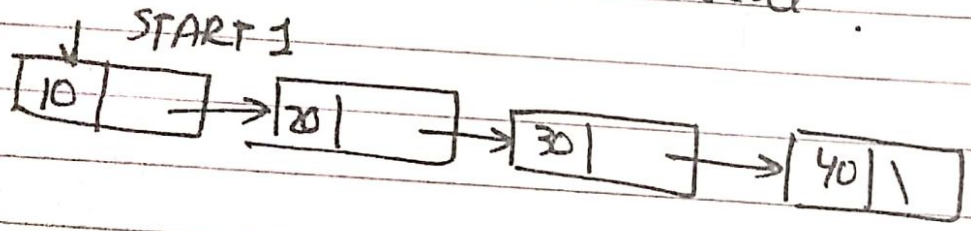
Priority Queue using Array, Stack, Queue.

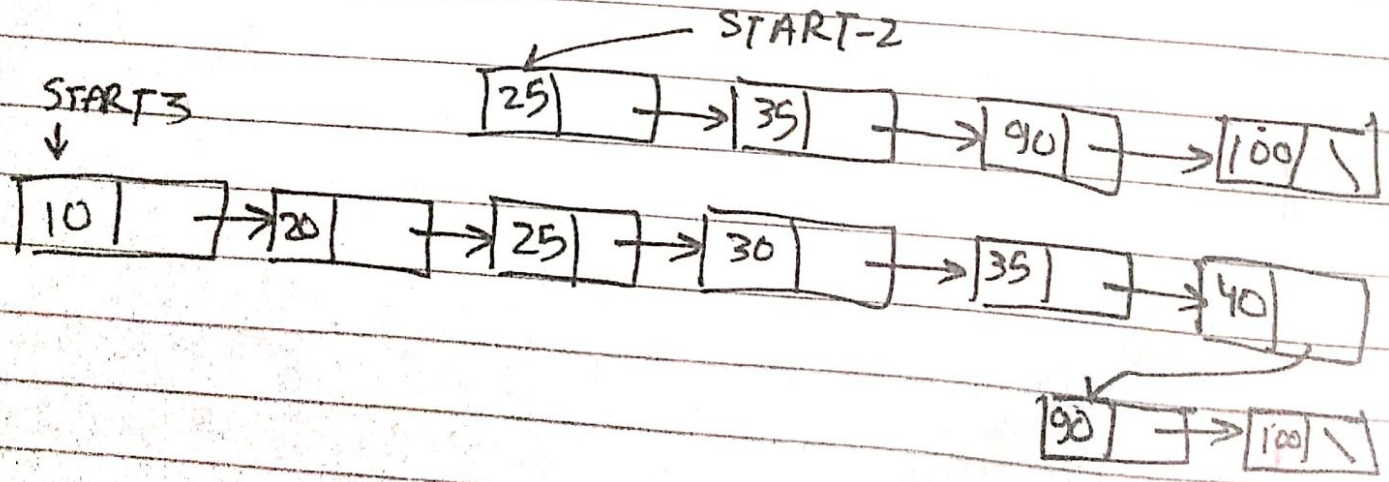
Ordered Insertion (START, x)

```
Q=NULL
P = START
WHILE   x >=P -> info AND P!=NULL
        Q=P
        P= P -> Next
IF Q!=NULL   THEN
        InsAft(Q,x)
ELSE
        InsBeg(START, x)
```

```
Algo  Merge Array( A[], M, B[], N)
BEGIN:
        C[M+N]
        i=1 , j=1, k=1
        WHILE i<=M AND j<=N Do
              IF A[i]<B[j]
                    C[k]=A[i]
                     k++ , i++ ;
              ELSE
                    C[k]=B[j]
                    k++, j++;
        WHILE i<=M DO
                    C[k]=A[i]
                         k++;
                         i++;
        WHILE j<=N DO
                    C[k]=B[j]
                         k++
                         j++;
        RETURN C
END;
```

```
ALGORITHM MergeLL(START1, START2)
BEGIN
        P=START1
        Q=START2, START3=NULL
    WHILE P!=NULL AND Q!=NULL DO
        IF P→Info < Q→Info THEN
            InsEnd(START3, P→info)
            P=P→Next
        ELSE
            InsEnd(START3, Q→info)
            Q=Q→Next
    WHILE P!=NULL DO
        InsEnd(START3, P→info)
        P=P→Next
    WHILE Q!=NULL DO
        InsEnd(START3, Q→info)
        Q=Q→Next
```





Sagar
Date:
Page:

ALGORITHM UNION LL(START1, START2)
BEGIN:

```
START3 = NULL
P = START1
Q = START2
WHILE P != NULL AND Q != NULL DO
    IF(P->Info == Q->Info) THEN
        InsEnd (START3, P->Info)
        P = P->Next
        Q = Q->Next

    ELSE
        IF(P->Info < Q->Info) THEN
            InsEnd (START3, P->Info)
            P = P->Next

        ELSE
            InsEnd (START3, Q->Info
            Q = Q->Next
```

Same

ALGORITHM Intersection (START1, START2)

```
START3 = NULL
P = START1
Q = START2

WHILE P != NULL AND Q != NULL DO
    IF(P->Info == Q->Info) THEN
        InsEnd (START3, P->Info
        P = P->Next
        Q = Q->Next
    ELSE IF (P->Info < Q->Info ) THEN
        P = P->Next
    ELSE
        Q = Q->Next
```

Sagar
Date:
Page:

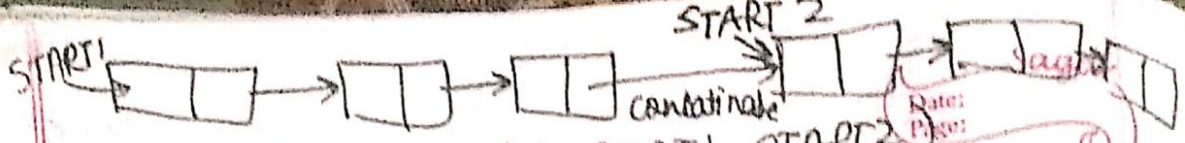
WRITE ALGO for following operation on Linked List
1) A-B (set difference) (considering link list as sets)
2) Count no of nodes in linked list
3) Concatinated 2 linked list
4) Count nodes having even or odd info
5) Print the info of Linked List in reverse order

Sagar
Date:
Page:

Count Nodes

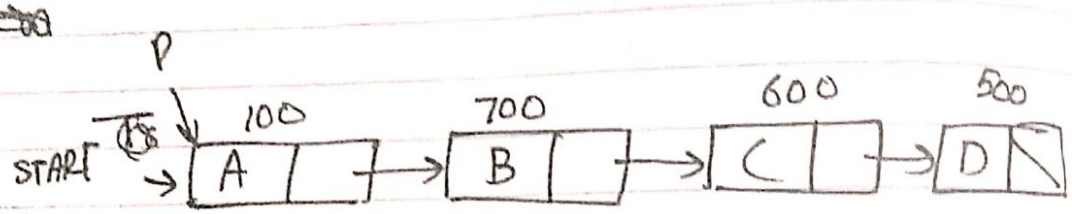
```
ALGORITHM CountNodes(START)
BEGIN:
        P=START
        count=0
        WHILE P!=NULL Do
            Count++
            P=P→Next
        RETURN count
END;
```

```
ALGORITHM CountEXO (START)
BEGIN:
        P=START
        Even=0
        Odd=0
        WHILE P != NULL DO
            IF P→Info % 2==0 THEN
                Even++
            ELSE
                Odd++
            P=P→Next
        Print Even
        Print Odd.
```



ALGORITHM Concatenate (~~Start~~START1 , START2)
BEGIN:
IF START1 == NULL THEN
RETURN START2
ELSE
P = START1
WHILE P→Next != NULL DO
P = P→Next
p→Next = START2
RETURN START1
END.

~~Traverse~~

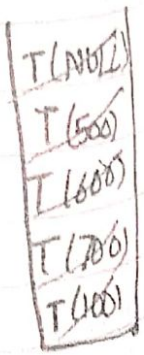


ALGORITHM Traverse (START)
BEGIN:
IF START != NULL THEN
WRITE (START→info)
Traverse(START→Next)
END;

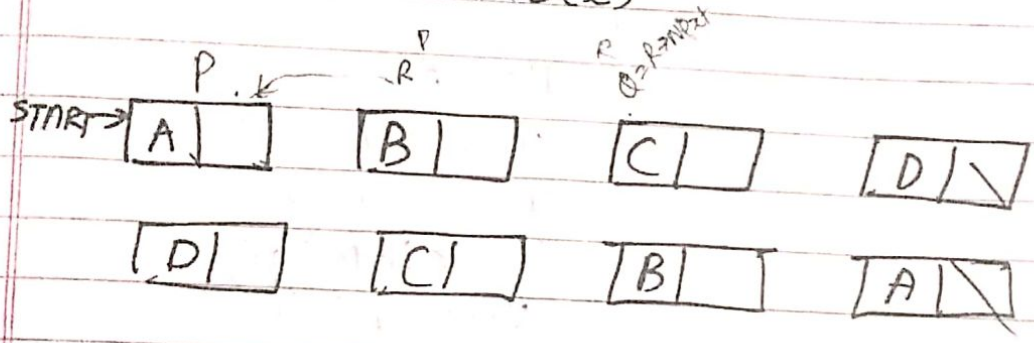
Output
A B C D

ReverseTraverse
ALGORITHM ~~STACK~~ (START)
BEGIN:
IF START != NULL THEN
Traverse (START→Next)
WRITE (STAR→info)
END;

Output
D C B A



```
RevTrav (START)
    STACKS
    Initialize(S)
    P = START
    WHILE P != NULL DO
        Push(S, P->info)
        P = P->Next
    WHILE !Empty(S) DO
        x = Pop(S)
        WRITE(x)
```

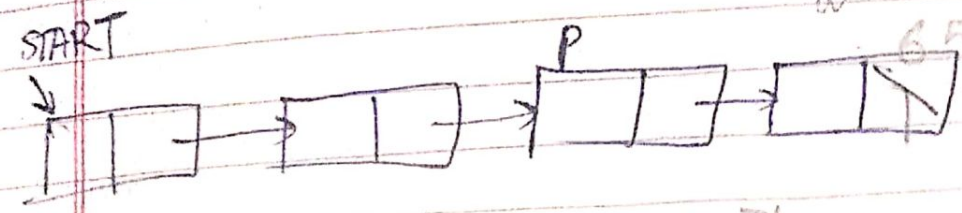


P = P->Next
P->Next = START
WRITE(P->Info)

```
Reverse (START)
P = START
R = START -> Next
WHILE R != NULL
    Q = R -> Next
    R -> Next = P
    P = R
    R = Q
START -> Next = NULL
START = P
```

-74

74



6 5 4 3 2 1 0
& 2 2 2 2
1 1

74 = 0 1 0 0 1 0 1 0
1 0 1 1 0 1 1 0

ALGORITHM DelBeg (START)
BEGIN:

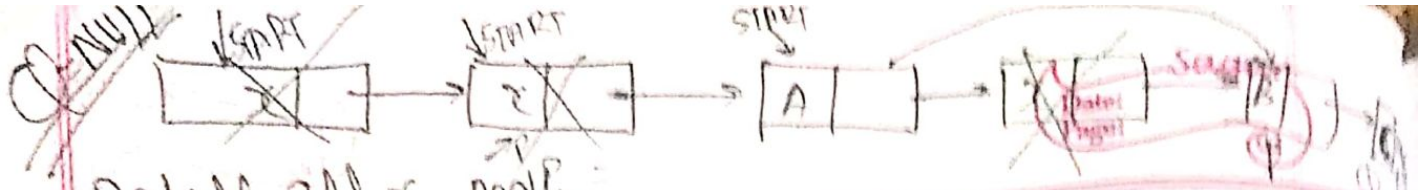

```
IF START == NULL THEN
    WRITE("Void deletion")
    EXIT(1)
ELSE
    P = START
    START = P→Next
    x = P→info
    FreeNode(p)
    Return(x)
END ;
```

Time Complexity - O(1)
Space Complexity - O(1)

```
ALGORITHM DelRear(P)
BEGIN:
    IF (P==NULL OR P→Next == NULL)
        WRITE("Void Deletion")
        EXIT(1)
    ELSE
        q = P→Next
        P→Next = q→Next
        x = q→Info
        FreeNode(q)
        Return(x)
END;
```

Time - O(1)
Space - O(1)

```
ALGORITHM DelEnd(START)
BEGIN:
    IF START == NULL THEN          → When no node
        WRITE ("Valid Deletion");
        EXIT(1)
    ELSE
        q = NULL
        P = START                  When only 1 node
        WHILE P→Next != NULL
            q = P
            P = P→Next
    ①   IF q != NULL THEN
            q→Next = NULL
        ELSE
            START = NULL
        x = P→info
        FreeNode(P)
END:
```



Delete all x nodes

IF P → Info == x

If (P = START)

DelBeg

ELSE IF

Q = P

Q = NULL

ALGORITHM Delete specific (START, x)

BEGIN:

P = START

Q = NULL

WHILE P != NULL DO

IF P → info == x THEN

P = P → Next

IF Q == NULL THEN

DelBeg (START)

ELSE

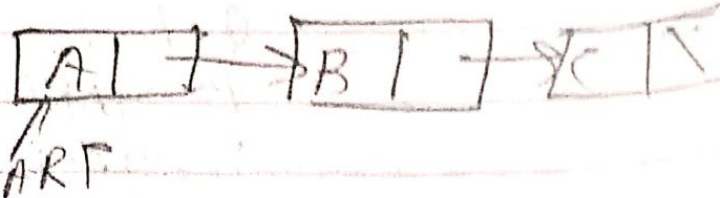
DelAft(Q)

ELSE

Q = P

P = P → Next

END;



Q-1 Delete every alternate Node in Linked List.

```
ALGO DeleteAlternate (START)
BEGIN:
        P = START
        Q = NULL
        i = 1
        WHILE (P != NULL) DO
            IF (i%2 == 0) THEN
                Del After(Q)   P = P->Next
                               Del After(Q)
            ELSE
                Q = P
                P = P-> Next
            i++;
END.
```

Q-2

```
ALGO DeleteOddNode (START)
BEGIN:
        P = START
        Q = NULL
        i = 0
        WHILE (P != NULL) DO
            IF (i%2 == 0) THEN
                P = P->Next
                IF (Q == NULL) THEN
                    Del Beg(Q)
                ELSE
                    Del After(Q)
            ELSE
                Q = P
END:
```

```
ALGORITHM Delete Odd (START)
BEGIN:
        P=START
        Q=NULL
        i=1
        WHILE (P!=NULL) DO
              IF (i%2 != 0)
                  P=P→Next
                  IF Q==NULL THEN
                        DelBeg(START)
                  ELSE
                        DelAfter(Q)
              ELSE
                  Q=P
                  P=P→Next
              i=i+1;
END;
```

Q-3 Delete All Nodes containing Odd Data item

```
ALGORITHM DeleteNodeoddData (START)
BEGIN:
        P=START
        Q=NULL
        WHILE(P!=NULL) DO
              IF (P→Info != 0)
```

Sagar

Q. ALGO for pair wise swap of nodes

Q. Split the Given Link list from very mid

Q. To sort the contents of linked list nodes.

ALGO SwapPairWise (START)

BEGIN:

P=START

Q=NULL

i=1

WHILE (P!= NULL) DO

IF (i%2 == 0)

t = P->Info

P->Info = Q->Info

Q->Info = t

P = P->Next

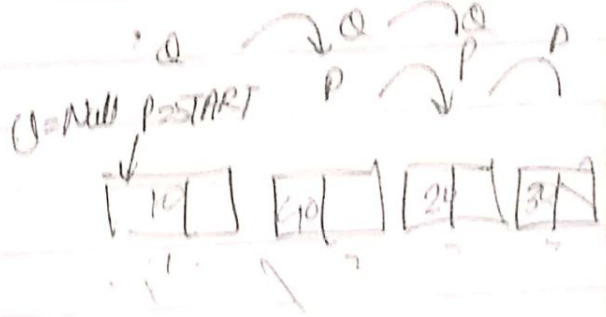
ELSE

Q = P

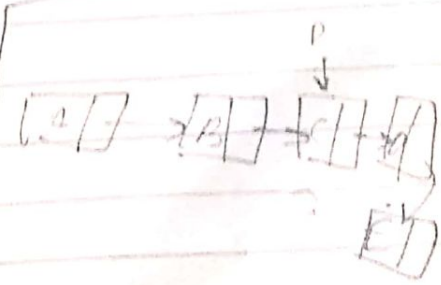
P = P->Next

i++

END;



Info badli address sam



P=START

C=0

WHILE(P!=NULL DO)

P=P->Next

C++

P=START Q=START

mid = [(1+C)/2]

i=1

WHILE P!=NULL DO

IF (i == mid)

Q = P->Next

P->Next = NULL

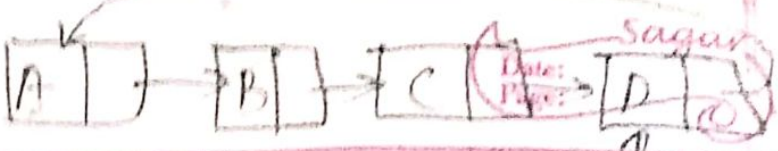
START2 = Q

ELSE P = P->Next

Sagar
Date:
Page:

	Insertion			Deletion			Traversal
	Beginning	Mid	End	Beg	Mid	End	
Array	O(N)	O(N)	O(1)	O(N)	O(N)	O(1)	O(N)
Linked list Simply Linked List	O(1)	O(1)	O(N)	O(1)	O(1)	O(N)	O(N)

Circular



Ins Beg (CSTART, x)

P = GetNode()
P → info = x
P → Next = CSTART → Next
CSTART → Next = P

if ((*CSTART) != NULL)
P → Next = (*CSTART) → Next;
else
CSTART = P
CSTARTNext = P

Ins End (CSTART, x)

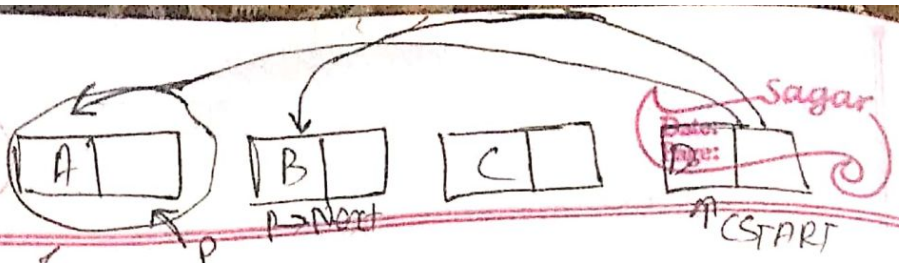
P = GetNode()
P → info = x
P → Next = CSTAR → Next
CSTART → Next = P
CSTART = P

O(1)

Ins Aft (P, x)

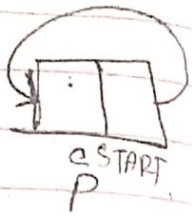
q = GetNode()
q → info = x
q → Next = P → Next
P → Next = q

Char *p
p = (char*) malloc (2 * sizeof(INT)) → It is address
*p = 'A'
*(p+1) = 'B'
*(p+2) = 'C'
*(p+3) = 'D';



ALGORITHM Del Beg (CSTART)

```
IF (CSTART == NULL) THEN
        WRITE ("Void Deletion")
        EXIT(1)
ELSE
        P = CSTART → Next
        IF CSTART → Next != CSTART      $ If not only 1 node
            ← CSTART → Next = P → Next
            x = P → info
            FreeNode (P)
            RETURN(x)
        ELSE
            CSTART = NULL  → for 1 node
            ← x = P → info
            ← FreeNode (P)
            ← RETURN x
```



ALGORITHM Del End (CSTART)

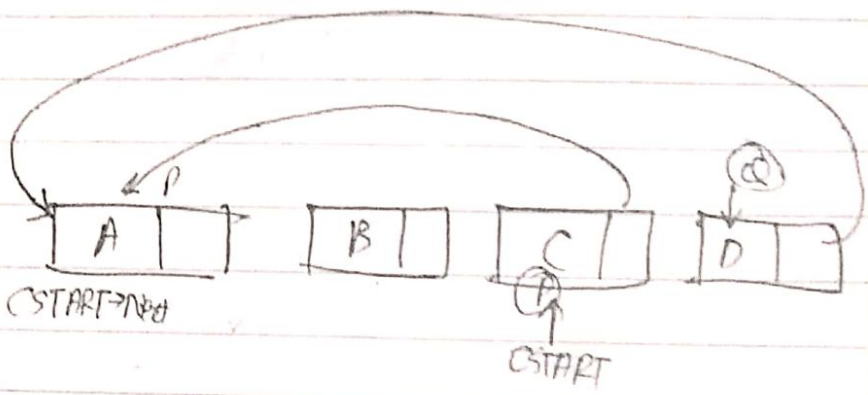
```
BEGIN:
        IF (CSTART == NULL) THEN
                WRITE ("VOID Deletion")
                EXIT (1)
        ELSE
                P = CSTART
                x = P → Info
                P → Next = CSTART
                FreeNode (P)
                Return x
```

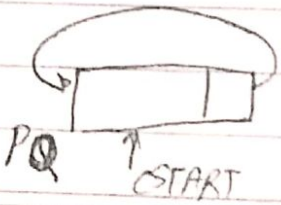
Sagar
Date:
Page:

ALGORITHM DelEnd (CSTART)
BEGIN:

Q = CSTART
P = CSTART → Next
WHILE P → Next != CSTART DO
P = P → Next
P → Next = CSTART → Next
FREE(Q)
x = Q → info
FreeNode(Q)
Return x

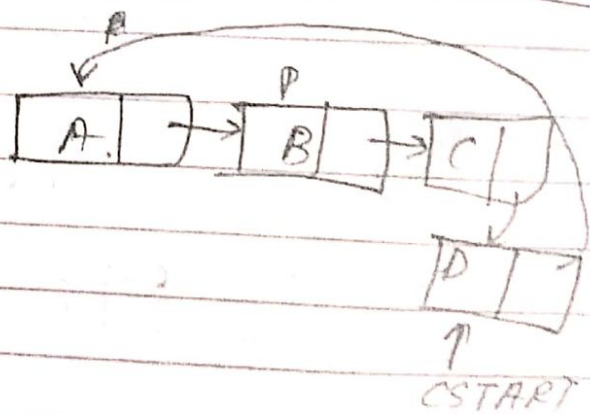
IF CSTART → Next == CSTART
CSTART = NULL
ELSE
CSTART = P

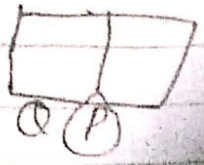


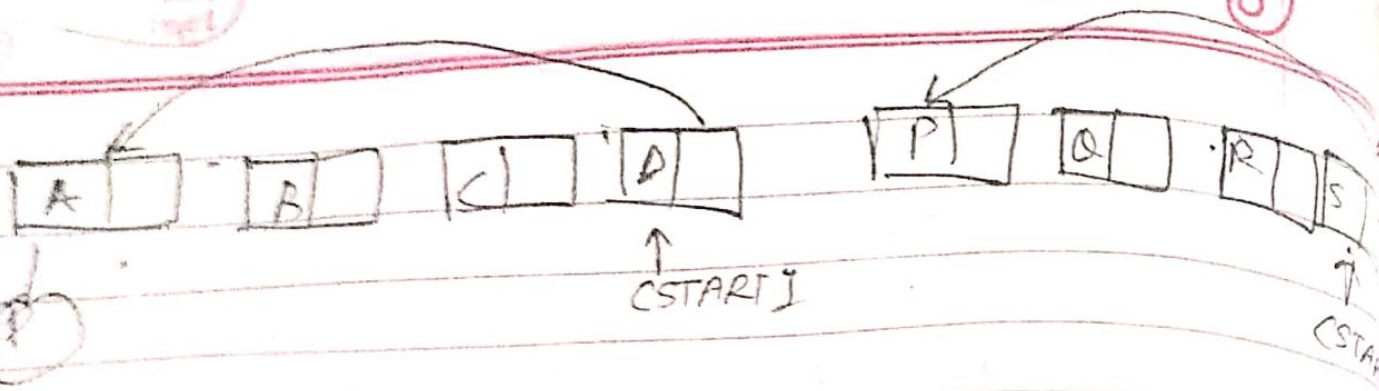


ALGORITHM DelAfter(P)
BEGIN:

Q = P → Next
IF P → Next == P THEN
~~P → Next = Q → Next~~
P = NULL
ELSE
P → Next = Q → Next
x = P → info
FreeNode(Q)
RETURN x







ALGORITHM

~~CSTART1 → CSTART2 → Next~~

P = CSTART1 → Next

CSTART1 → Next = CSTART2 → Next

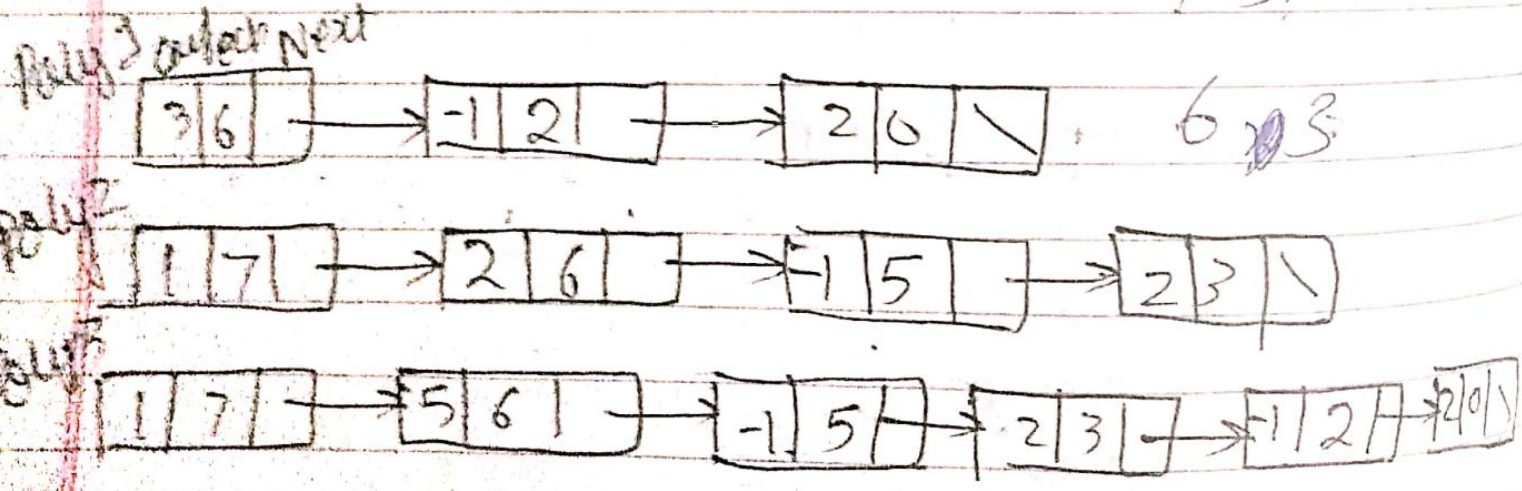
CSTART2 → Next = P

Polynomial Addition

$3x^6 - x^4 + 2x^3 + 11x + 4$

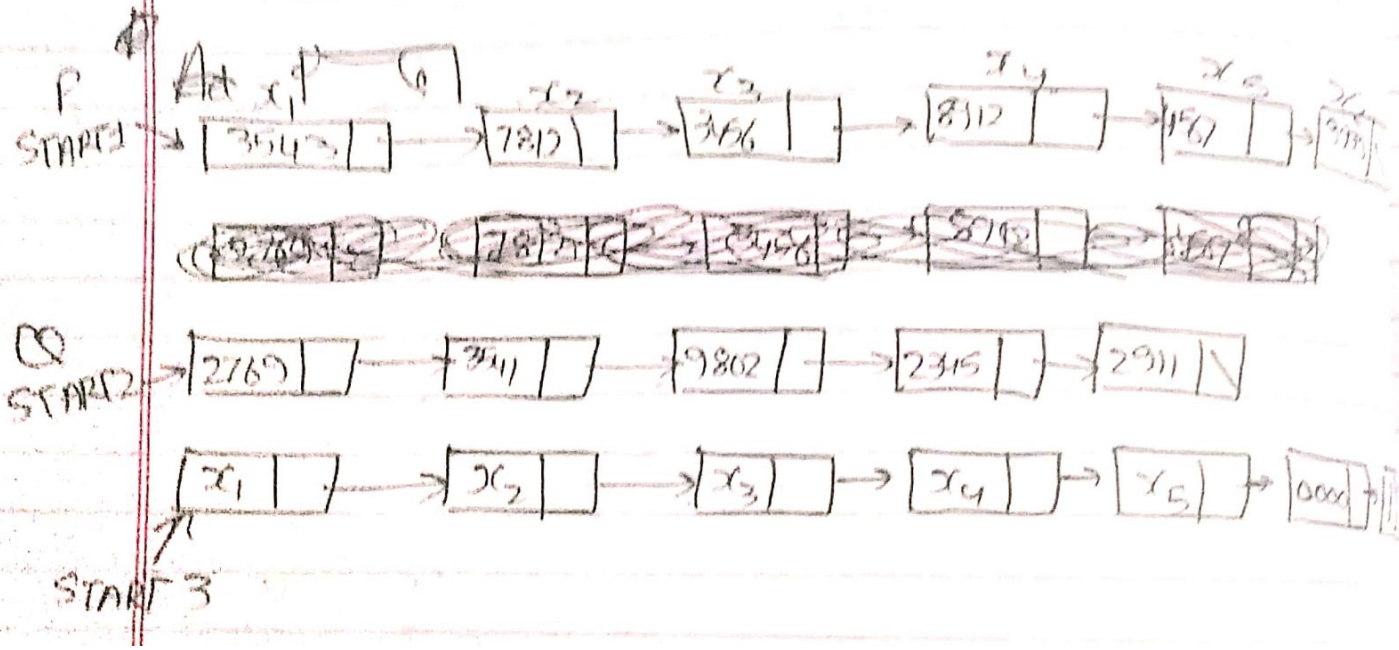
$5x^7 - x^6 + 3x^5 - 3x^3 + 2x^2 - 7$

$5x^7 + 2x^6 + 3x^5 - x^4 - x^3 + 2x^2 + 11x - 3$



```
ALGORITHM Polynomial Addition (Poly1, Poly2)
BEGIN:
    Poly3 = NULL
    P = Poly1
    Q = Poly2
    WHILE P!=NULL AND Q!=NULL DO
      IF P->Exp == Q->Exp THEN
          InsEnd (Poly3, P->Coef + Q->Coef, P->Exp)
            P=P->Next
            Q=Q->Next
      ELSE
          IF P->Exp > Q->Exp THEN
              InsEnd (Poly3, P->Coef, P->Exp)
              P=P->Next
          ELSE
              InsEnd (Poly3, Q->Coef, Q->Exp)
              Q= Q->Next
    WHILE P!=NULL DO
          InsEnd (Poly3, P->Coef, P->Exp)
            P=P->Next
    WHILE Q!=NULL DO
          InsEnd (Poly3, Q->Coef, Q->Exp)
          Q=Q->Next
    RETURN Poly3
END.
```

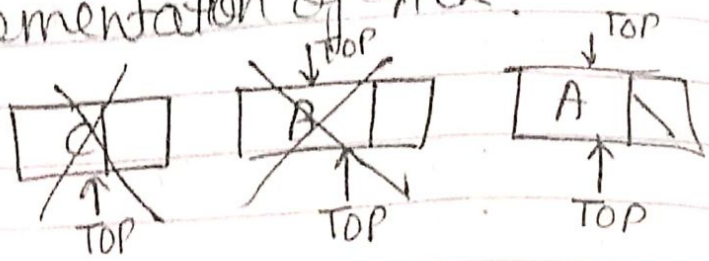
Add 2 very long digits



```
Add LongNumbers(START1, START2)
BEGIN:
    P=START1
    Q=START2
    START3=NULL
    Carry=0
    WHILE P!=NULL AND Q!=NULL DO
        TOTAL = P->Info+Q->Info+Carry
        Carry = Total /. 10000
        Sum = Total %.10000
        InsEnd (START3, Sum)
        P=P->Next, Q=Q->Next
    WHILE P!=NULL DO
        TOTAL=P->Info+Carry
        Carry = Total /.10000
        Sum = Total %.10000
        InsEnd (START3, sum)
        P=P->Next
    WHILE Q!=NULL DO
        TOTAL=Q->Info+Carry
        Carry = TotAL /.10000
        Sum = Total %.10000
        InsEnd (START3, Sum)
        Q=Q->Next
    IF Carry==1 THEN
        InsEnd (START3, 1)
    RETURN START3
END;
```

Sagar
Date:
Page:

Linklist Implementation of STACK.



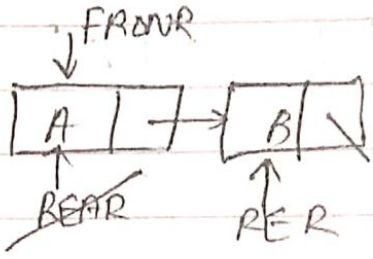

```
ALGORITHM Initialize (TOP)
  BEGIN:
        TOP = NULL
  END;
ALGORITHM PUSH (TOP, x)
BEGIN:
        InsBeg (TOP, x)
END;
ALGORITHM  Empty(TOP)
BEGIN:
      IF TOP == NULL
      RETURN TRUE
      ELSE
        RETURN FALSE
END

ALGORITHM POP(TOP)
BEGIN:
    IF Empty (TOP) THEN
        WRITE ("Stack Underflow")
        EXIT(1)
    ELSE
        x = DelBeg(TOP)
        RETURN x
END
```

```
ALGORITHM StackTop(
BEGIN:
        ReturnTopInfo
END.
```

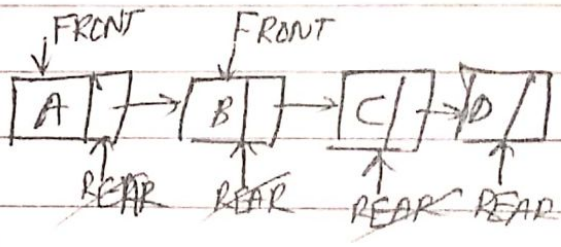
Linklist Implementation of Queue

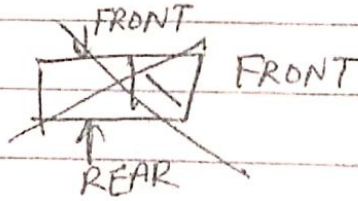
```
ALGORITHM Initialize (FRONT, REAR)
  BEGIN:
        REAR = NULL
        FRONT = NULL
  END
```

```
ALGORITHM EnQueue(FRONT, REAR, x)
BEGIN:
      IF (REAR == NULL) THEN
            InsBeg (REAR)
            FRONT = REAR
      ELSE
            InsAft (REAR, x)
            REAR = REAR -> Next
END
```




```
ALGORITHM DeQue(FRONT, REAR)
BEGIN:
      IF FRONT == NULL THEN
            WRITE ("Queue Underflow")
            EXIT(1)
      ELSE
            x = DelBeg (FRONT)
            IF FRONT == NULL THEN
                  REAR = NULL
            RETURN x
```



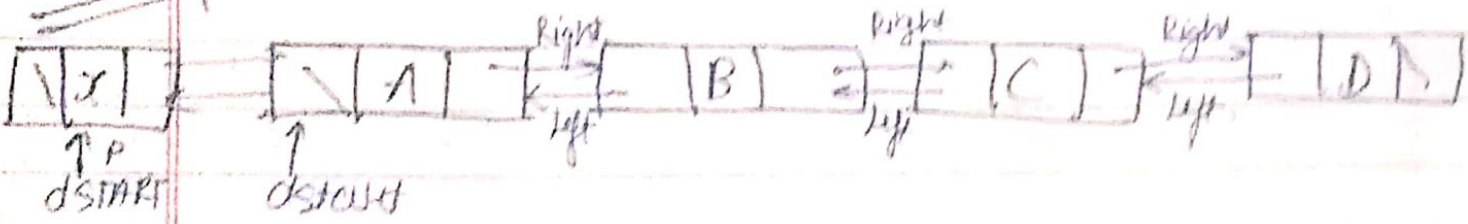



```
ALGORITHM Empty (FRONT)
 BEGIN:
        IF FRONT == NULL
              RETURN TRUE
        ELSE
              RETURN FALSE
 END:
```

ALGORITHM Traverse
BEGIN:

P = FRONT
WHILE P != NULL
  WRITE P -> Info
  P = P -> Next

Doubly LL



P -> info
P -> Left
P -> Right

ALGORITHM InsBeg(dSTART, x)
BEGIN:

P = GetNode()
P -> info = x
P -> Left = NULL
P -> Right = dSTART
IF dSTART != NULL THEN
  dSTART -> Left = P
dSTART = P

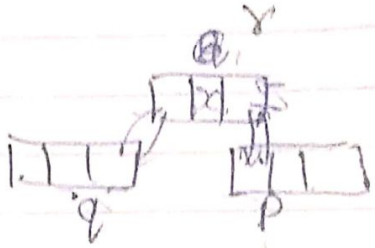
END;

ALGO Traverse(dSTART)
BEGIN:

P = dSTART
WHILE (P != NULL)
  Write(P -> Info)
  P = P -> Right

END;

ALGORITHM InsEnd (dstart, x)
BEGIN: IF (dstart = NULL)
ELSE InsBeg (dstart, x)
p = dstart
WHILE (P → Right != NULL)
P = P → Right
Q = GetNode()
Q → info = x
Q → Right = NULL
P → Right = Q
Q → Left = P

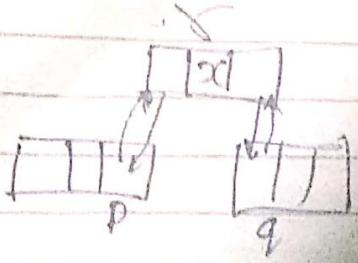


InsLeft (P, x)
BEGIN

q = GetNode()
q → info = x
q → right = p
p → left = q

q = P → left
R = GetNode()
R → info = x
q → Right = R
R → Left = Q
R → Right = P
P → left = R

Ins Right (P, x)
BEGIN:



q = P → Right
r = getNode()
r → info = x
P → Right = r
r → Left = P
q → Left = r
r → right = q