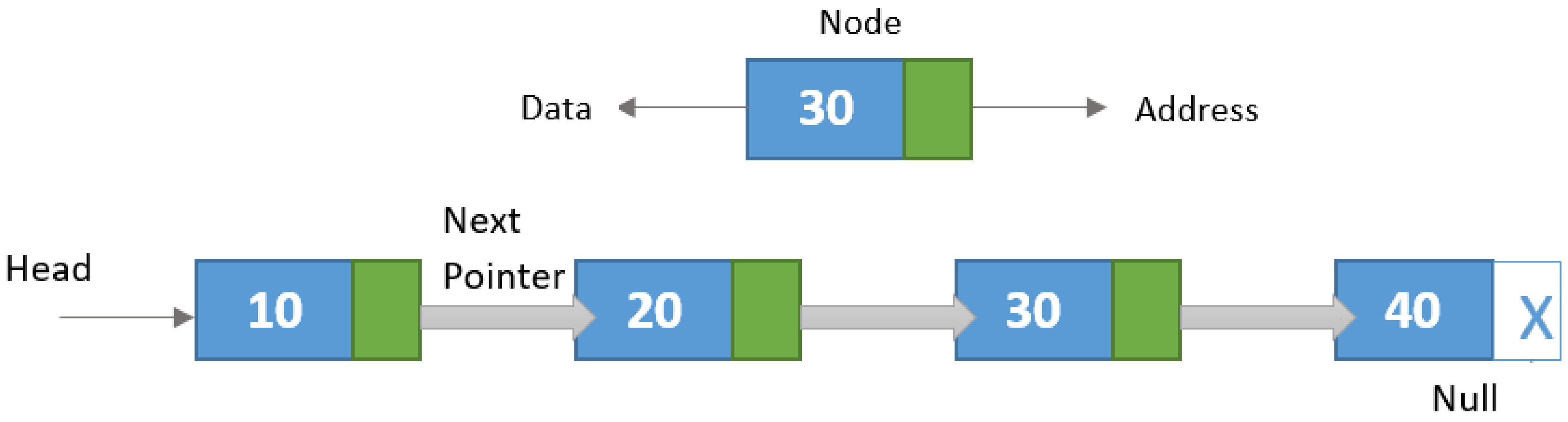
ABES Engineering College
College Code-032
Estd. 2000

Data Structure Training

Linked List



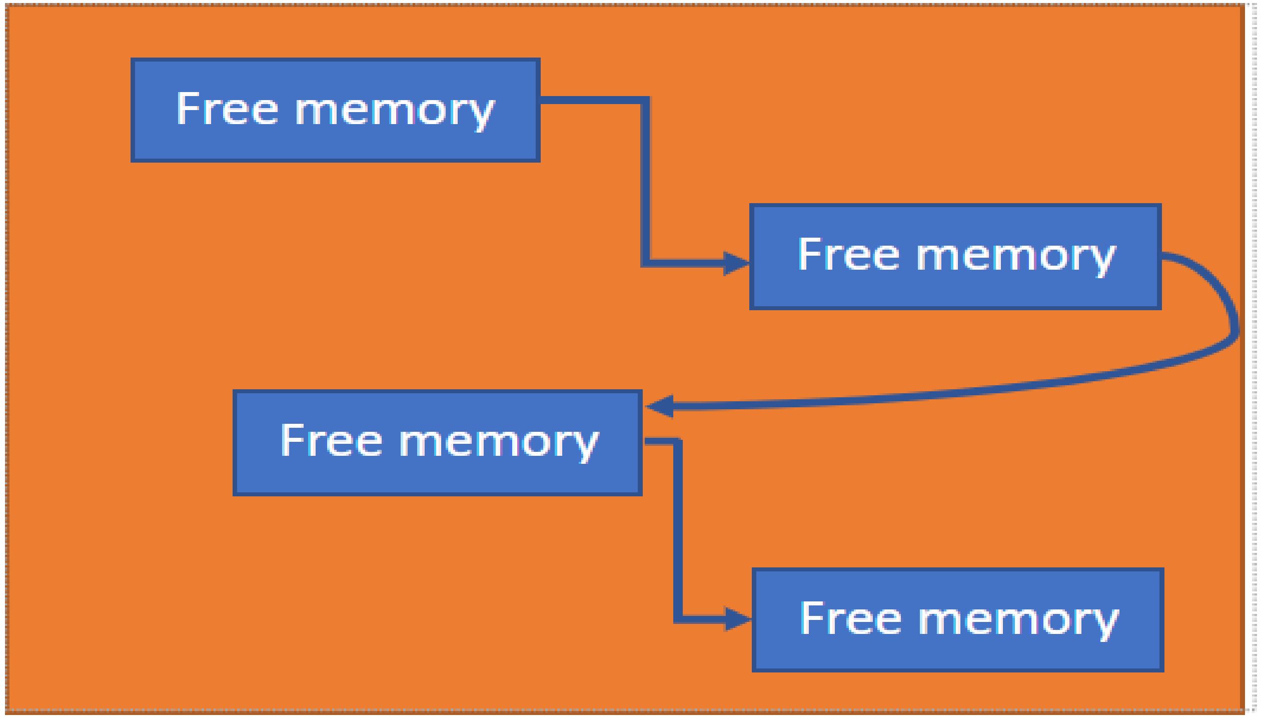
Single/Linear Linked List

Linked List can be best understood with the concept of Treasure hunt. In this, we had to find some items with the help of multiple clues. In this hunt, we have a clue and this clue gives address of next place, thereby we have a starting position and we can search the given item with the help of multiple clues in between.

Single/Linear Linked List

Why Link List ?

- When Free memory is not available contiguously.
- It provides efficient use of memory.
- It's a Dynamic Data Structure, which can change its size during the program execution.
- Insert and delete operations are easier and efficient to design.



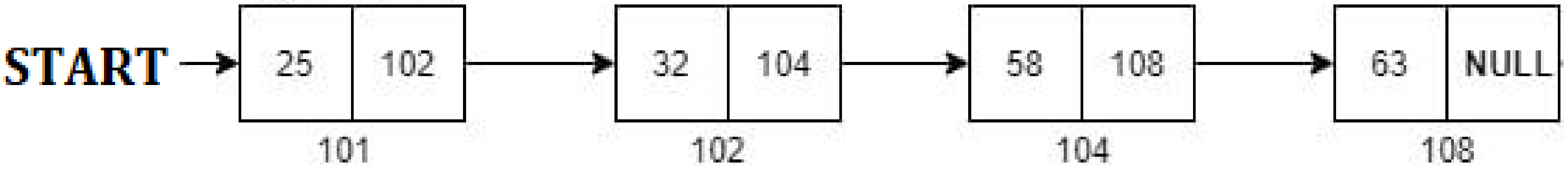
Single/Linear Linked List

Definition-

1-A Linked List is a data structure which consists of nodes. Every node contains at least two fields. One of which contains the information and other one contains address of next node.

2-Like an array, linked list is homogeneous collection of element, but, these elements are not contiguous and hence not indexed. Because they are not contiguous , next element ca not be reached by just incrementing the address.
Each element of the list holds a link to next element(in addition to data or info) and we move forward in the list by using that link.

This combination of data and link is called node. A linked is also known as self referential data structure because in next field it holds address of another node of its same type.



Single/Linear Linked List

Properties-

- A Linked List is identified through the address of the first node
- To access a node we need to reach to that node

<u>Linked List Applications:</u>

1. Memory Organization
2. Web pages (URL Linking)
3. Linux File System: Handling big files using the concept of I-Nodes
4. Implementation of other Data Structures like Stack, Queue, Tree and
5. Polynomial Arithmetic
6. Arithmetic Computation
7. Maintaining directory of names
8. Representation of Sparse Matrices
9. Maintaining Hash Tables (Chaining)
10. Image Viewer (Use of next and previous buttons for watching images stored in a folder)

Single/Linear Linked List

Analogy:

1. Train arrangement

2. Music Player Playlist

3. DNA molecules

4. Roller chain of bicycle

5. Packet based message delivery on network

6. Giving travel directions

Single/Linear Linked List

Linked List Vs Array

There are some pros and cons with each of the data structure, so too the Array. Array offers the indexed access (fast) but they have fixed size. Static array cannot grow or shrink as per the requirement. Array size may fall short or remain under-utilized according the problem in hand. The Dynamic Arrays, e.g. Python List, provide the variable length but the concept is based on reserving surplus memory. Insertion of the data item at the beginning of Static or Dynamic array is costlier than inserting the data at the tail.

Linked List, on the other hand, allocates the memory on demand and insertion and deletions are easier as compared to Arrays.

Single/Linear Linked List

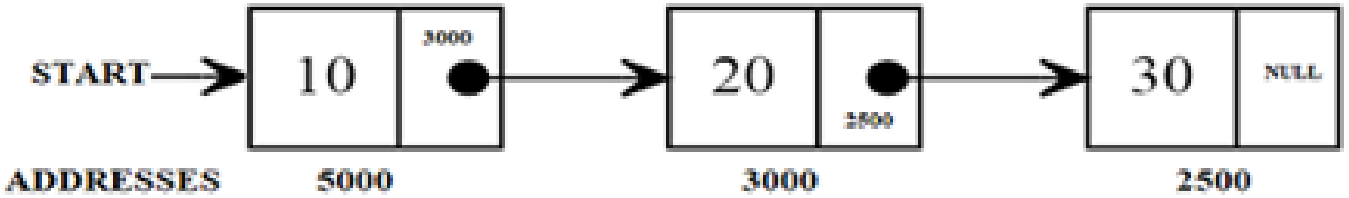
Types of Linked List:

- Linear/Singly Linked List
- Circular Linked List
- Doubly Linked List
- Circular Doubly Linked List

Single/Linear Linked List

Single Link List-

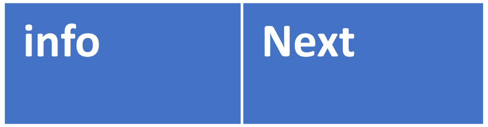
The diagram below shows linear Linked List where each node contains information and the address of the next node. The address field of last node contains no address (NULL).



Single/Linear Linked List

Notations Used In Link List-

For a node address having P, field are accessed as-



P→info

P→Next

- **GetNode( )** is used of allocation of memory for new node
- **START** is used for keeping the address of First node. In case the Linked list is empty, START keeps a NULL.
- **Item** is used to refer to the element to be inserted.

Single/Linear Linked List

1) Consider the following link list

a→ b→ c→ d→ e→ f→ g

and perform the following statements-

1. Struct node *p;

2. P = s→next→next→next;

3. P→next→next→next→next = s→next→next;

4. s→next→next = P→next→next→next;

5. printf(P→next→next→next→next→next→next→next→info);

What would be the output ?

A. a **B.** c

C. g **D.** f

Single/Linear Linked List

1) Consider the following link list

a→ b→ c→ d→ e→ f→ g

and perform the following statements-

1. Struct node *p;

2. P = s→next→next→next;

3. P→next→next→next→next = s→next→next;

4. s→next→next = P→next→next→next;

5. printf(P→next→next→next→next→next→next→next→info);

What would be the output ?

A. a	B. c
C. g	[✓ D. f]

Single/Linear Linked List

2） Consider the following link list

d → c → b → a → g → f → e

and perform the following statements-

1. Struct node *p;

2. P = s→next→next→next→next;

3. s→next→next→next = p→next→next;

4. p = s→next→next→next;

5. s→next→next→next→next = s→next→next;

6. printf ("%c", s→next→next→next→next→next→next→next→info);

What would be the output ?

A. a **B.** e

C.g **D.** f

Single/Linear Linked List

2） Consider the following link list

d → c → b → a → g → f → e

and perform the following statements-

1. Struct node *p;

2. P = s→next→next→next→next;

3. s→next→next→next = p→next→next;

4. p = s→next→next→next;

5. s→next→next→next→next = s→next→next;

6. printf ("%c", s→next→next→next→next→next→next→next→info);

What would be the output ?

A.a **B.** e

C.g **D.** f

Single/Linear Linked List

Primitive Operations on Linked List Insertion

Insertion at the beginning-

Insertion after the given node-

Insertion at the end-

Single/Linear Linked List

3) Insertion at the beginning-

In this case a new node is added at the beginning of the linked list and this new node becomes the starting point of the linked list.

```
ALGORITHM InsBeg(START, item)
BEGIN:
1) P = GetNode()
2) P → Info = item
3) ____________________
4) START = P
END;
```

Complete the Line No 3? Choose the correct option.

A. P→Next = START   **B.** P→Next = NULL

C. P→Next = P   **D.** None

Single/Linear Linked List

3) Insertion at the beginning-

In this case a new node is added at the beginning of the linked list and this new node becomes the starting point of the linked list.

```
ALGORITHM InsBeg(START, item)
BEGIN:
1) P = GetNode()
2) P → Info = item
3) ____________________
4) START = P
END;
```

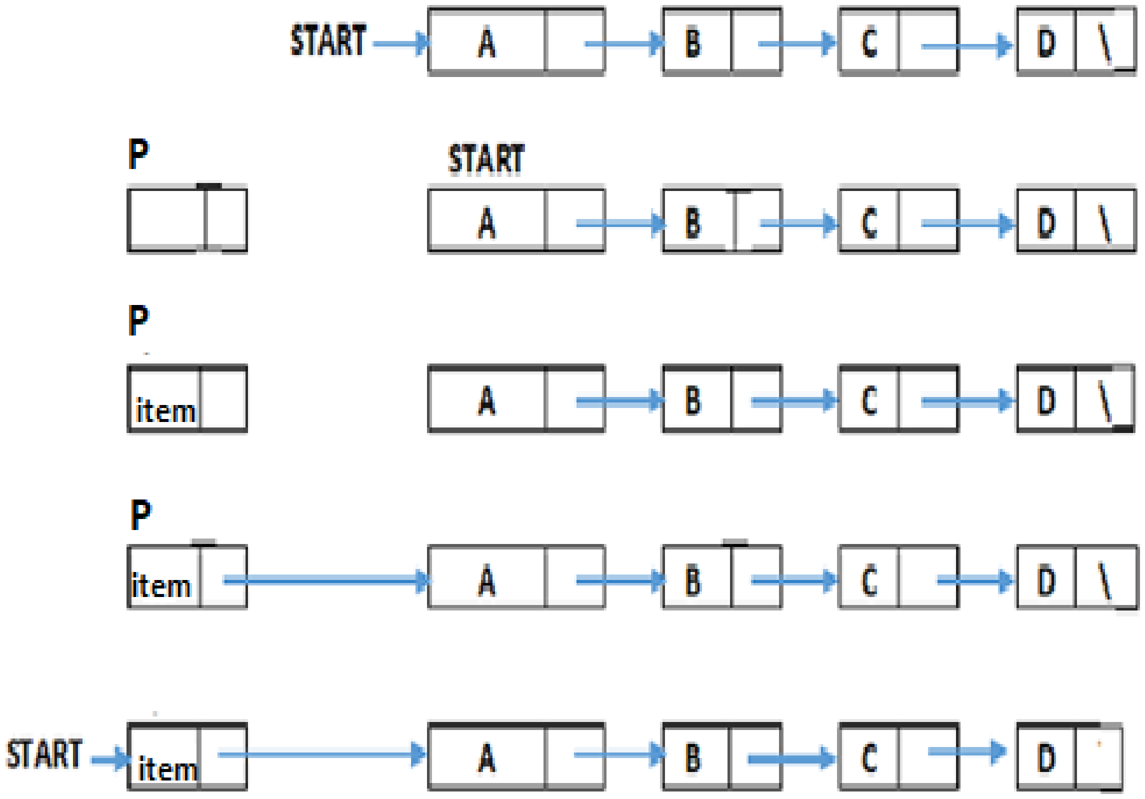
Complete the Line No 3? Choose the correct option.

[✓ A. P→Next = START] B. P→Next = NULL

C. P→Next = P D. None

Single/Linear Linked List

Insertion at the beginning(explanation)



P = GetNode()

P → Info = item

P → Next = START

START = P

Single/Linear Linked List

Insertion at the beginning(complexity)

No. of Address adjustment = 2

4) What would be the time complexity to insert the node at beginning in SLL? Choose the correct option.

A. O(1)	**B.** O(N)
C. O(N^2)	**D.** None

5) What would be the space complexity to insert the node at beginning in SLL? Choose the correct option.

A. O(N)	**B.** O(1)
C. O(N^2)	**D.** None

Single/Linear Linked List

Insertion at the beginning(complexity)

No. of Address adjustment = 2

4) What would be the time complexity to insert the node at beginning in SLL? Choose the correct option.

[✓ A. O(1)] B. O(N)

C. O(N^2) D. None

5) What would be the space complexity to insert the node at beginning in SLL? Choose the correct option.

A. O(N) [✓ B. O(1)]

C. O(N^2) D. None

Single/Linear Linked List

6) Insertion after given node-

In this case, we need to insert a node after a given specific number of nodes. Here, we need to skip the desired numbers of node in list to move the pointer at the point after which the new node will be inserted.

```
ALGORITHM InsAft(Q, item)
BEGIN:
1) IF Q == NULL THEN
2)       WRITE("Void Insertion")
3)       RETURN
4) P =GetNode()
5) P → Info = item
6) ____________________
7) Q → Next = P
END:
```

Complete the Line No 6? Choose the correct option.

A. P→Next = Q

B. Q→Next = P→Next

C. P→Next = Q→Next

D. None

Single/Linear Linked List

6) Insertion after given node-

In this case, we need to insert a node after a given specific number of nodes. Here, we need to skip the desired numbers of node in list to move the pointer at the point after which the new node will be inserted.

ALGORITHM InsAft(Q, item)

```
BEGIN:
1) IF Q == NULL THEN
2)        WRITE(“Void Insertion”)
3)        RETURN
4) P =GetNode()
5) P → Info = item
6) ____________________
7) Q → Next = P
END;
```

Complete the Line No 6? Choose the correct option.

A. P→Next = Q

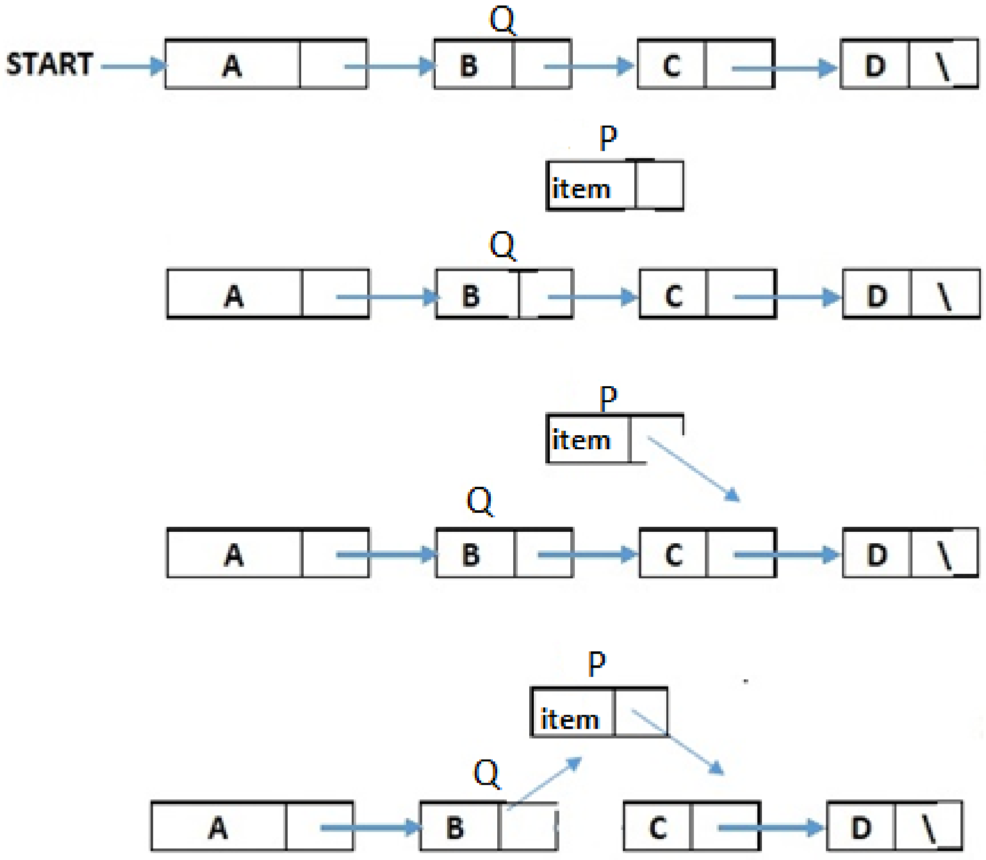
B. Q→Next = P→Next

[✓ **C.** P→Next = Q→Next]

D. None

Single/Linear Linked List

Insertion after given node(explanation)-



Initial Linked List

P =GetNode()

P → Info = item

P → Next = Q → Next

Q → Next = P

Single/Linear Linked List

Insertion after given node(complexity)

No of link adjustment = 2

Complexity of Operation

Time Complexity = O(1)

Here already provided address of node after that insertion takes place so takes O(1) time but What would be the time complexity if we have only address of start node?

7) Choose the correct option.

A. O(1)

B. O(N)

C. O(N^2)

D. None

Space Complexity = O(1)

Single/Linear Linked List

Insertion after given node(complexity)

No of link adjustment = 2

Complexity of Operation

Time Complexity = O(1)

Here already provided address of node after that insertion takes place so takes O(1) time but What would be the time complexity if we have only address of start node?

7) Choose the correct option.

[✓ A. O(1)]
B. O(N)

C. O(N^2)
D. None

Space Complexity = O(1)

Single/Linear Linked List

Insertion at the end-

In this case a new node is added at the end of list. For this, we need to traverse the entire list and then change next of last node to the new node.

ALGORITHM InsEnd(START, item)

```
BEGIN:
1) P =GetNode()
2) P → Info = item
3) P → Next = NULL
4) IF ____________________THEN
5)          START = P
6) ELSE
7)         Q=START
8)          WHILE ____________ DO
9)                Q = Q → Next
10)         Q → Next = P
END;
```

8) Complete the Line No 4? Choose the correct option.

A. START == NULL

B. Q→Next == NULL

C. P == NULL

D. None

Single/Linear Linked List

Insertion at the end-

In this case a new node is added at the end of list. For this, we need to traverse the entire list and then change next of last node to the new node.

```
ALGORITHM InsEnd(START, item)
BEGIN:
1) P =GetNode()
2) P → Info = item
3) P → Next = NULL
4) IF ____________________THEN
5)         START = P
6) ELSE
7)         Q=START
8)         WHILE _____________ DO
9)             Q = Q → Next
10)        Q → Next = P
END;
```

8) Complete the Line No 4? Choose the correct option.

[✓ **A.** START == NULL]

B. Q→Next == NULL

C. P == NULL

D. None

Single/Linear Linked List

Insertion at the end-

In this case a new node is added at the end of list. For this, we need to traverse the entire list and then change next of last node to the new node.

ALGORITHM InsEnd(START, item)

```
BEGIN:
1) P =GetNode()
2) P → Info = item
3) P → Next = NULL
4) IF ______________________THEN
5)          START = P
6) ELSE
7)          Q=START
8)          WHILE ______________ DO
9)                Q = Q → Next
10)         Q → Next = P
END;
```

9) Complete the Line No 8? Choose the correct option.

A. Q != NULL

B. Q→Next == NULL

C. Q → Next != NULL

D. None

Single/Linear Linked List

Insertion at the end-

In this case a new node is added at the end of list. For this, we need to traverse the entire list and then change next of last node to the new node.

ALGORITHM InsEnd(START, item)

```
BEGIN:
1) P =GetNode()
2) P → Info = item
3) P → Next = NULL
4) IF _____________________THEN
5)          START = P
6) ELSE
7)          Q=START
8)          WHILE _____________ DO
9)               Q = Q → Next
10)         Q → Next = P
END;
```

9) Complete the Line No 8? Choose the correct option.

A. Q != NULL

B. Q→Next == NULL

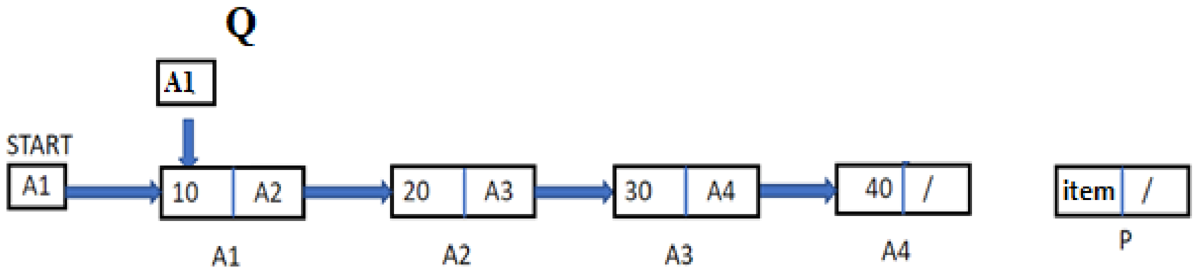
[✓ **C.** Q → Next != NULL]

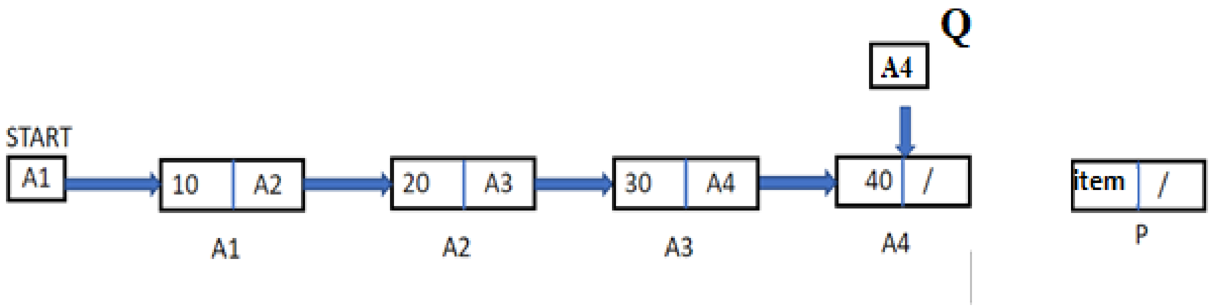
D. None

Single/Linear Linked List

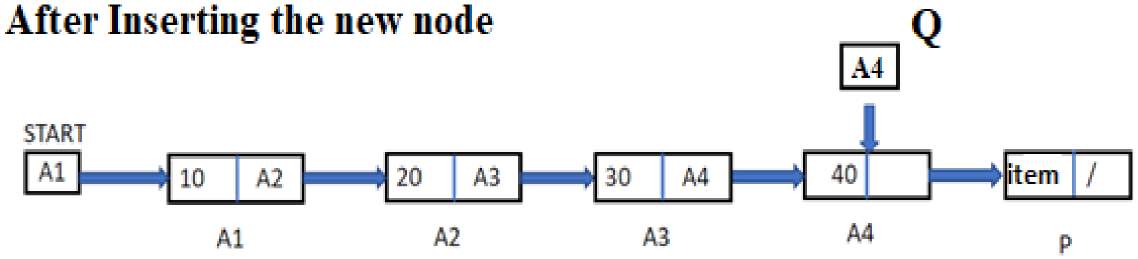
Insertion at the end-

Before insertion





After Inserting the new node



```
Q=START
P =GetNode()
P → Info = item

P → Next = NULL

WHILE Q → Next != NULL DO
        Q = Q → Next

Q → Next = P
```

Single/Linear Linked List

Insertion at the end(complexity)

No. of Address adjustment = 2

10) What would be the time complexity to insert the node at end in SLL? Choose the correct option.

A. O(N) **B.** O(1)

C. O(N^2) **D.** None

11) What would be the space complexity to insert the node at beginning in SLL? Choose the correct option.

A. O(N) **B.** O(1)

C. O(N^2) **D.** None

Single/Linear Linked List

Insertion at the end(complexity)

No. of Address adjustment = 2

10) What would be the time complexity to insert the node at end in SLL? Choose the correct option.

[✓ A. O(N)] B. O(1)

C. O(N^2) D. None

11) What would be the space complexity to insert the node at beginning in SLL? Choose the correct option.

A. O(N) [✓ B. O(1)]

C. O(N^2) D. None

Single/Linear Linked List

Primitive Operations on Linked List

Deletion

Deletion at the beginning-

Deletion after the given node-

Deletion at the end-

Single/Linear Linked List

Deletion at the beginning-

Given a linked list, our task is to remove the first node of the linked list and update the head pointer of the linked list. To remove first node, we need to make the second node as head and delete the memory allocated for first node.

ALGORITHM DelBeg(START, item)

```
BEGIN:
        IF START = = NULL THEN
                WRITE ("Void Deletion")
        ELSE
            P= START
            Item = P → Info
            ___________________
            Free(P)
            RETURN Item
END;
```

12) Complete the Missing line in the given Algorithm? Choose the correct option.

A. START = NULL

B. START = START→Next

C. P = P→Next

D. None

Single/Linear Linked List

Deletion at the beginning-

Given a linked list, our task is to remove the first node of the linked list and update the head pointer of the linked list. To remove first node, we need to make the second node as head and delete the memory allocated for first node.

```
ALGORITHM DelBeg(START, item)
BEGIN:
        IF START = = NULL THEN
                WRITE ("Void Deletion”)
        ELSE
            P= START
            Item = P → Info
            ____________________
            Free(P)
            RETURN Item
END;
```

12) Complete the Missing line in the given Algorithm? Choose the correct option.

A. START = NULL

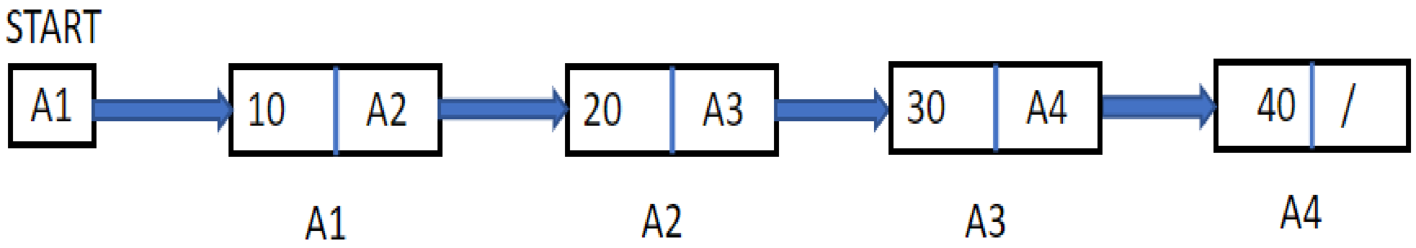
[✓ B. START = START→Next]

C. P = P→Next

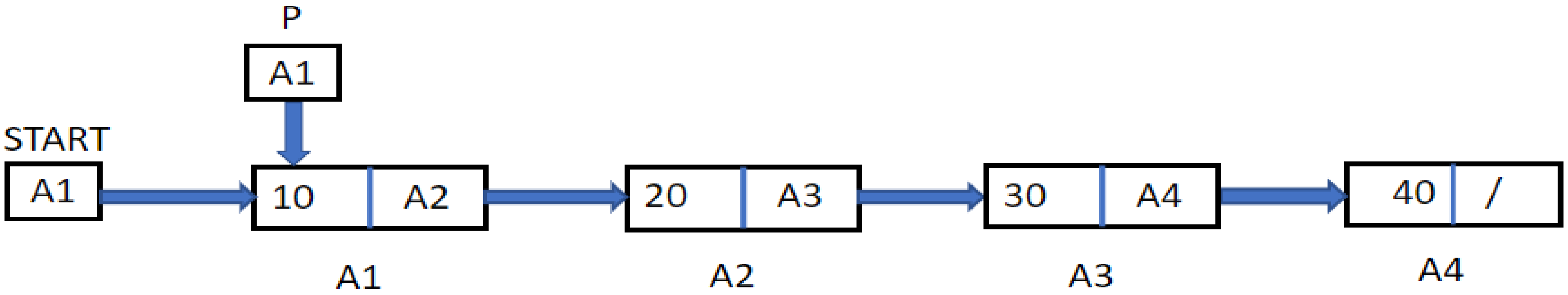
D. None

Single/Linear Linked List

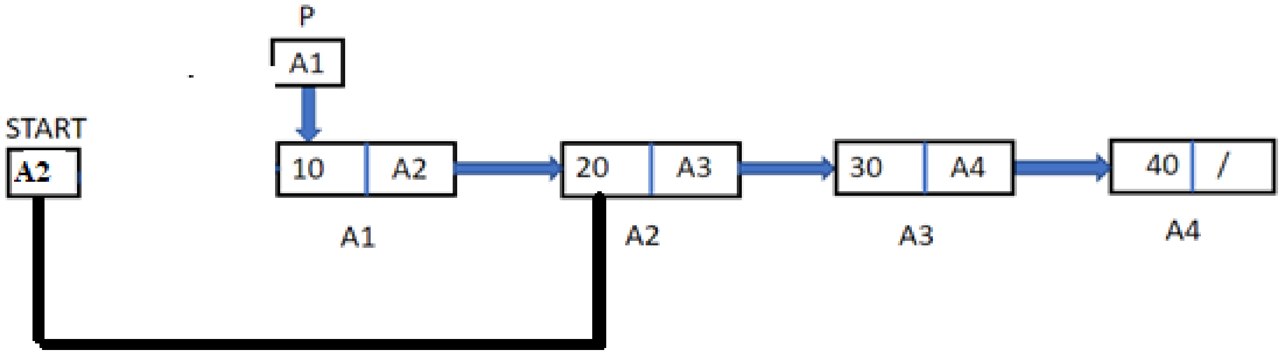
Deletion at the beginning-



Before Deletion



P = START



START = START->Next

Single/Linear Linked List

Deletion at the beginning-

No. of Address adjustment = 1

Complexity of Operation

Time Complexity = O(1)

here add the node in beginning so there is no traversal, hence takes O(1) time.

Space Complexity = O(1)

Single/Linear Linked List

Deletion at the end-Given a linked list, our task is to remove the last node of this list and set the next part of the second last node as null. To remove last node, we need to traverse up to second last node and set the next part of this node to be null.

```
ALGORITHM  DelEnd(START, item)
BEGIN:
    IF START == NULL THEN
        WRITE (“Void Deletion”)
    ELSE
        P=START
    A) IF __________________THEN
                START = START⟶ Next
        ELSE                                //General
                WHILE  P⟶ Next != NULL  DO
                    B)
                    P = P⟶ Next
                Q⟶ Next = NULL
                Item = P⟶ Info
                Free(P)
    RETURN Item
```

13) Complete the Line No A? Choose the correct option.

A. START == NULL

B. START→Next == NULL

C. START != NULL

D. None

Single/Linear Linked List

Deletion at the end-Given a linked list, our task is to remove the last node of this list and set the next part of the second last node as null. To remove last node, we need to traverse up to second last node and set the next part of this node to be null.

```
ALGORITHM  DelEnd(START, item)
BEGIN:
    IF START == NULL THEN
        WRITE ("Void Deletion")
    ELSE
        P=START
    A) IF ___________________THEN
                START = START→ Next
        ELSE                                  //General
                WHILE  P→ Next != NULL  DO
                    B)
                    P = P→ Next
                Q→ Next = NULL
                Item = P→ Info
                Free(P)
    RETURN Item
```

13) Complete the Line No A? Choose the correct option.

A. START == NULL

[✓ **B.** START→Next == NULL]

C. START != NULL

D. None

Single/Linear Linked List

Deletion at the end-Given a linked list, our task is to remove the last node of this list and set the next part of the second last node as null. To remove last node, we need to traverse up to second last node and set the next part of this node to be null.

```
ALGORITHM  DelEnd(START, item)
BEGIN:
    IF START == NULL THEN
        WRITE (“Void Deletion”)
    ELSE
        P=START
    A) IF ________________THEN
                START = START→ Next
        ELSE                                  //General
                WHILE  P→ Next != NULL  DO
                    B)
                    P = P→ Next
                Q→ Next = NULL
                Item = P→ Info
                Free(P)
    RETURN Item
```

14) Complete the Line No B? Choose the correct option.

A. P = Q

B. Q = P

C. Q = P→Next

D. None

Single/Linear Linked List

Deletion at the end-Given a linked list, our task is to remove the last node of this list and set the next part of the second last node as null. To remove last node, we need to traverse up to second last node and set the next part of this node to be null.

```
ALGORITHM  DelEnd(START, item)
BEGIN:
    IF START == NULL THEN
        WRITE ("Void Deletion")
    ELSE
        P=START
    A) IF ___________________THEN
                START = START→ Next
        ELSE                                //General
                WHILE  P→ Next != NULL  DO
                    B)
                    P = P→ Next
                Q→ Next = NULL
                Item = P→ Info
                Free(P)
    RETURN Item
```

14) Complete the Line No B? Choose the correct option.

A. P = Q

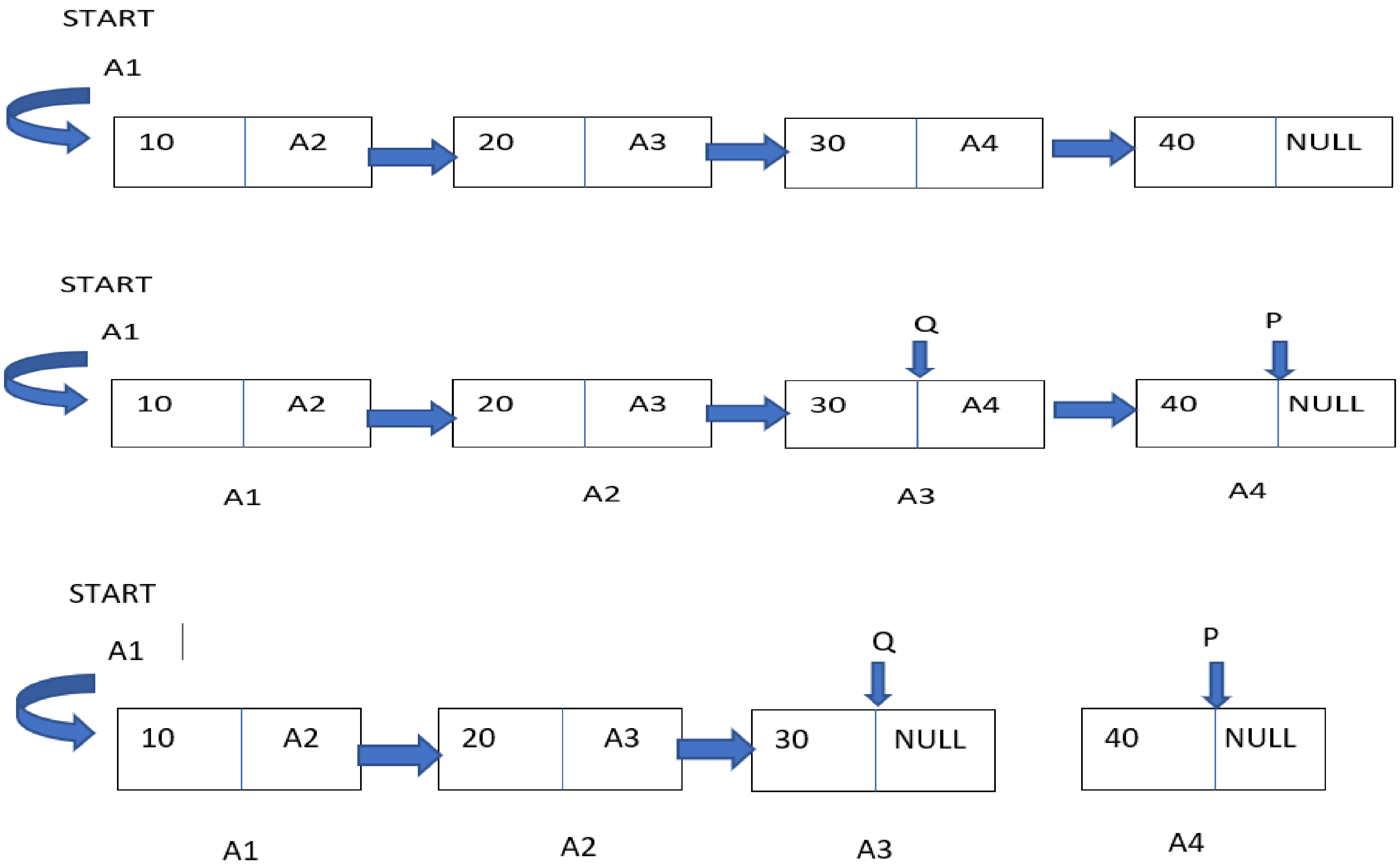
[✓ **B.** Q = P]

C. Q = P→Next

D. None

Single/Linear Linked List

Deletion at the end-



Single/Linear Linked List

Deletion at the end

No of Link adjustment = 1

Complexity of Operation:

Space Complexity = O(1)

15) What would be the time complexity to delete the node at end in SLL? Choose the correct option.

A. O(N)

B. O(1)

C. O(N^2)

D. None

Single/Linear Linked List

Deletion at the end

No of Link adjustment = 1

Complexity of Operation:

Space Complexity = O(1)

15) What would be the time complexity to delete the node at end in SLL? Choose the correct option.

[✓ A. O(N)] **B.** O(1)

C. O(N^2) **D.** None

Single/Linear Linked List

16) Deletion after specific position-

Given a linked list, our task is to remove the specific node from the list. In this case, we need to traverse up to specific count after which the node is to be deleted. To remove, we need to set the address part of this node to the node after the node to be deleted.

```
ALGORITHM  DelAfter(Q)
BEGIN:
        IF Q = = NULL THEN
            WRITE (“Void Deletion”)
        ELSE
            A) ____________________
            Q→Next = P→Next
            Item = P→ Item
            FreeNode(P)
            RETURN Item
END;
```

Single/Linear Linked List

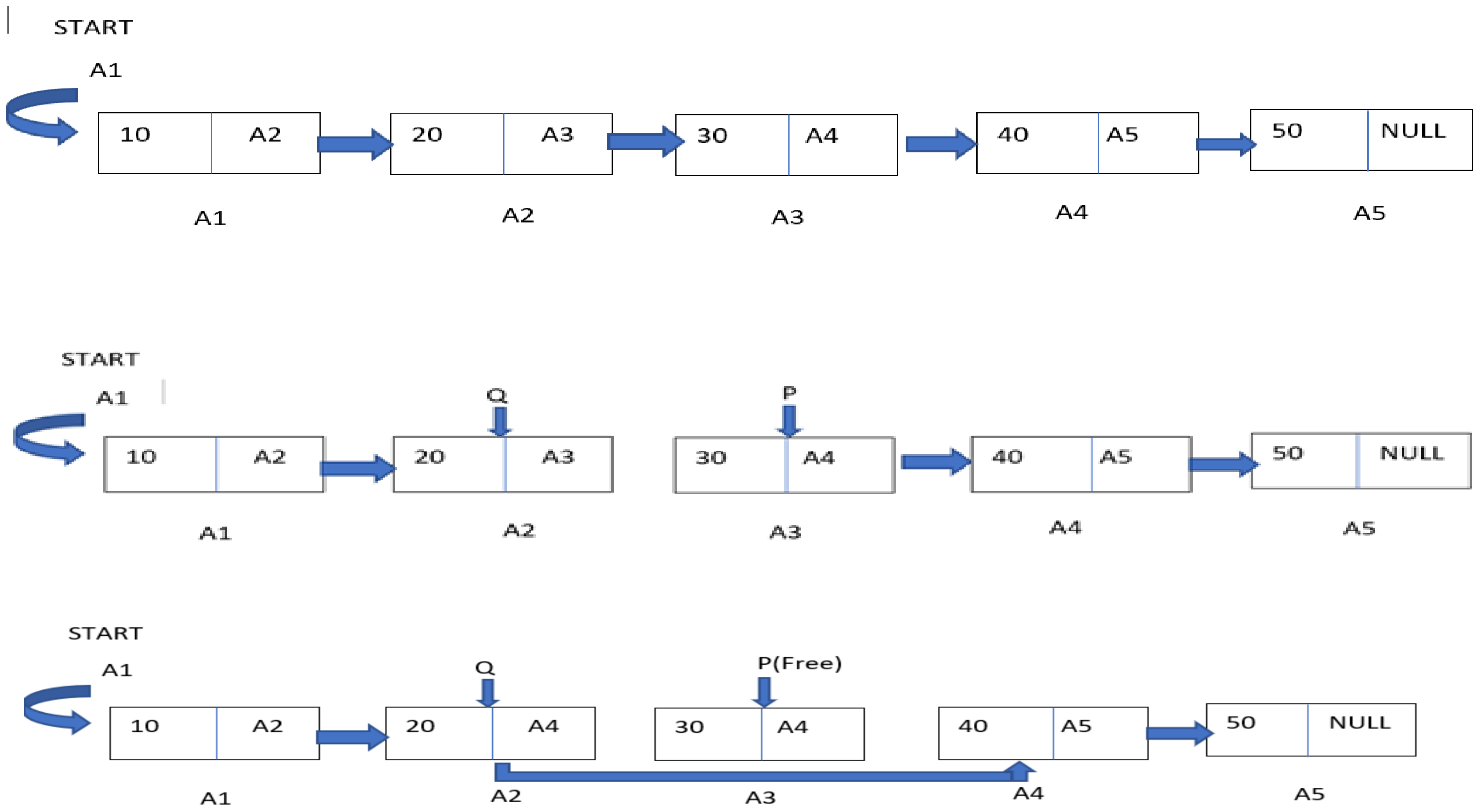
16) Deletion after specific position-

Given a linked list, our task is to remove the specific node from the list. In this case, we need to traverse up to specific count after which the node is to be deleted. To remove, we need to set the address part of this node to the node after the node to be deleted.

```
ALGORITHM  DelAfter(Q)
BEGIN:
        IF Q = = NULL THEN
            WRITE (“Void Deletion”)
        ELSE
            A) ______________________
            Q→Next = P→Next
            Item = P→ Item
            FreeNode(P)
            RETURN Item
END;
```

Single/Linear Linked List

Deletion after specific position-



START
A1
10 A2 | 20 A3 | 30 A4 | 40 A5 | 50 NULL
A1 A2 A3 A4 A5

START
A1
Q P
10 A2 | 20 A3 | 30 A4 | 40 A5 | 50 NULL
A1 A2 A3 A4 A5

START
A1
Q P(Free)
10 A2 | 20 A4 | 30 A4 | 40 A5 | 50 NULL
A1 A2 A3 A4 A5

Single/Linear Linked List

Deletion after specific position-

No of address adjustment = 1

Complexity of Operation:

Time Complexity = O(1)

Space Complexity = O(1)

Single/Linear Linked List

Sum of all elements in Linked List

Given a singly linked list, the task is to find the sum of nodes of the given linked list.

Solution

The algorithm work by calling the function Add, which helps in adding all the value of data that is present in all nodes by traversing each node from start.

Steps :

1. Initialize the node P to start and sum to be 0.
2. Traverse the entire list and add the data values present in all nodes of the in variable sum.
3. Return this sum value to program.

Single/Linear Linked List

Sum of all elements in Linked List

```
ALGORITHM Add(P)
BEGIN:
SUM=0
A) WHILE ____________________DO
        SUM = SUM + P→Info
        P = P → Next
        RETURN SUM
END;
```

Complexity- O(n)

17) Here P contains the address of first node. Complete the Line No A? Choose the correct option.

A. P→Next != NULL

B. P != NULL

C. None

Single/Linear Linked List

Sum of all elements in Linked List

```
ALGORITHM Add(P)
BEGIN:
SUM=0
A) WHILE __________________DO
        SUM = SUM + P→Info
        P = P → Next
        RETURN SUM
END;
```

Complexity- O(n)

17) Here P contains the address of first node. Complete the Line No A? Choose the correct option.

A. P→Next != NULL

[✓ B. P != NULL]

C. None

Single/Linear Linked List

Traversal Algorithm (Recursive)

Traversing in Linked List is determined by visiting all the nodes present in Linked List one by one. In this we visit all the nodes present in linked list one by one and prints its data value.

Solution:

1.Point node P at starting of linked list. P = Start.

2.Visit all the nodes present in linked list, and prints its data part.

ALGORITHM Display(P)

```
BEGIN:
        A)_________________
            WRITE  P→Info DO
            Display(P→Next)
END;
```

18) Here P contains the address of first node. Complete the Line No A? Choose the correct option.

A. WHILE P != NULL DO

B. IF P != NULL THEN

C. IF P→Next != NULL

D. NULL

Single/Linear Linked List

Traversal Algorithm (Recursive)

Traversing in Linked List is determined by visiting all the nodes present in Linked List one by one. In this we visit all the nodes present in linked list one by one and prints its data value.

Solution:

1.Point node P at starting of linked list. P = Start.

2.Visit all the nodes present in linked list, and prints its data part.

```
ALGORITHM Display(P)

BEGIN:

        A)__________________

            WRITE  P→Info DO

            Display(P→Next)

END;
```

18) Here P contains the address of first node. Complete the Line No A? Choose the correct option.

A. WHILE P != NULL DO

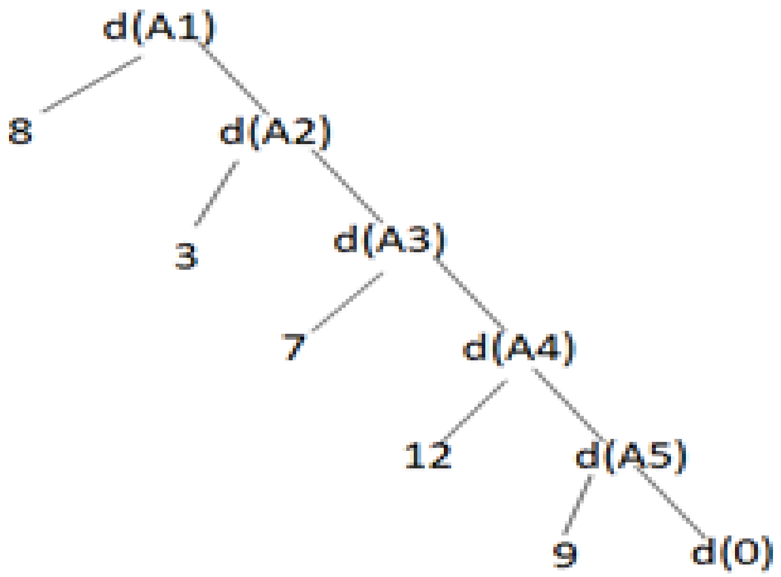
[✓ B. IF P != NULL THEN]

C. IF P→Next != NULL

D. NULL

Single/Linear Linked List

Traversal Algorithm (Recursive)



STACK

P =0

P = A5

P = A4

P = A3

P = A2

P = A1

Single/Linear Linked List

Traversal Algorithm (Recursive)

Complexity of Operation:

19) What would be the time complexity to traverse the SLL in recursive manner? Choose the correct option.

A. O(N) **B.** O(1)

C. O(N^2) **D.** None

20) What would be the space complexity to traverse the SLL in recursive manner? Choose the correct option.

A. O(N) **B.** O(1)

C. O(N^2) **D.** None

Single/Linear Linked List

Traversal Algorithm (Recursive)

Complexity of Operation:

19) What would be the time complexity to traverse the SLL in recursive manner? Choose the correct option.

[✓ A. O(N)]   **B.** O(1)

C. O(N^2)   **D.** None

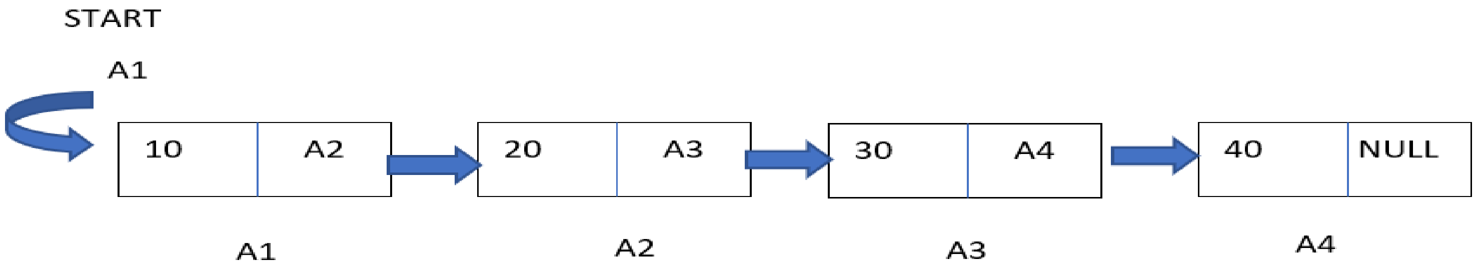
20) What would be the space complexity to traverse the SLL in recursive manner? Choose the correct option.

A. O(1)   [✓ **B.** O(N)]

C. O(N^2)   **D.** None

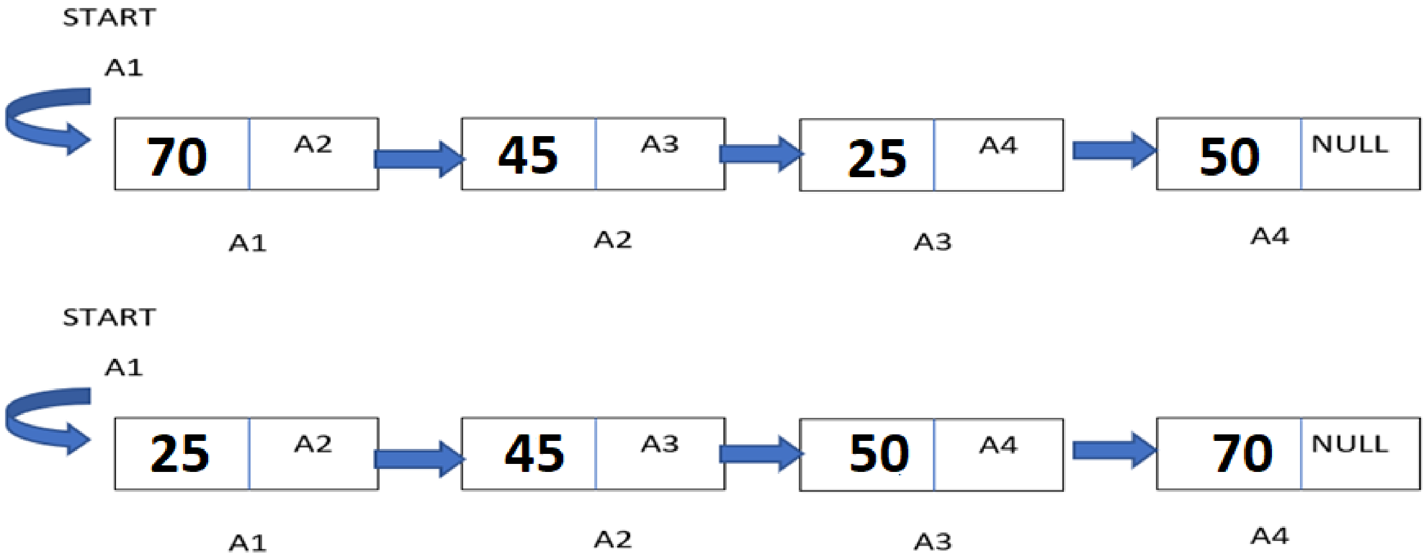
Single/Linear Linked List

Find the Count of nodes in the Linked List



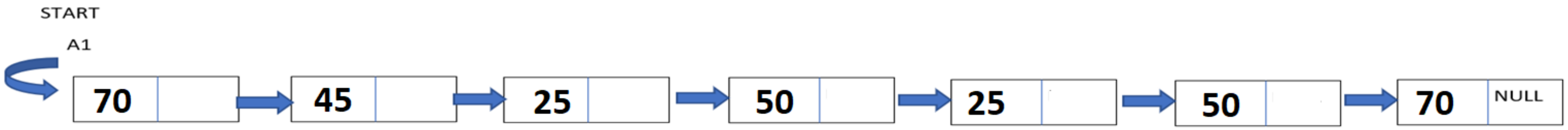
Single/Linear Linked List

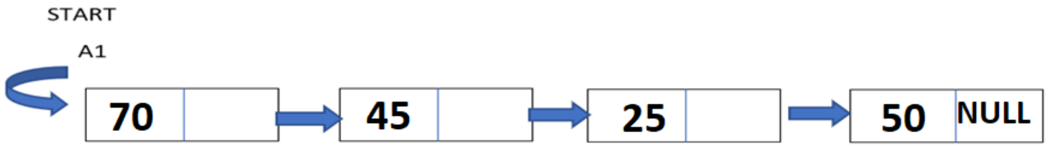
Sort the Given Linked List



Single/Linear Linked List

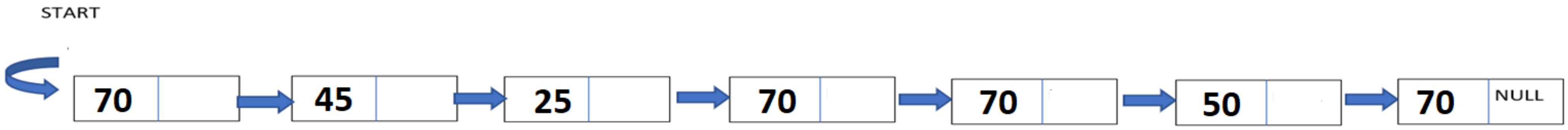
Remove duplicate nodes from the Linked List

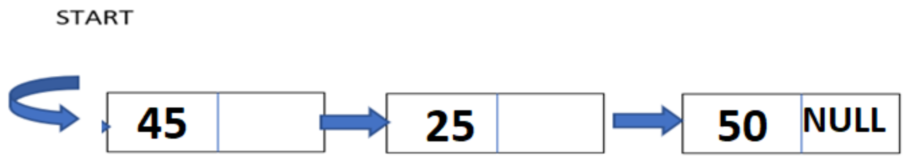




Single/Linear Linked List

Remove all nodes containing specific information from the Linked List





Single/Linear Linked List

Concatenation of two Linked List

Analogy:

This concept can be best understood with the help of two trains A and B. Train A consists of 8 coaches and Train B consists of 6 coaches. During the time of festival because of increase in rush the railway department concatenate these two trains. It can be done simply by joining the last coach of train A with the first coach of Train B.

Two different linked list can be concatenated using the same approach. Address part of ending node of list one is linked to first node of second list. This approach only required traversing entire nodes of list one and when we are at last node than we can simply linked its address part with the first node of second list.

Single/Linear Linked List

21) Concatenation of two Linked List

```
ALGORITHM ConcatLinkList(Start1, Start2)
BEGIN:
    IF Start1 == NULL THEN
        RETURN Start2
    ELSE
        P = Start1
        WHILE P→Next! = NULL
                P = P→ Next
                _______________
    RETURN Start1
END;
```

Explanation:

In the above algorithm initially, there are two input Linked List. Start1 has the starting address of first Linked List and Start2 has starting address of second Linked List. A temporary pointer P is taken which traverses till the last node of first Linked List and stops at last node address. Then the Next part of last node is assigned the address of first node of second Linked List via Start2

Single/Linear Linked List

21) Concatenation of two Linked List

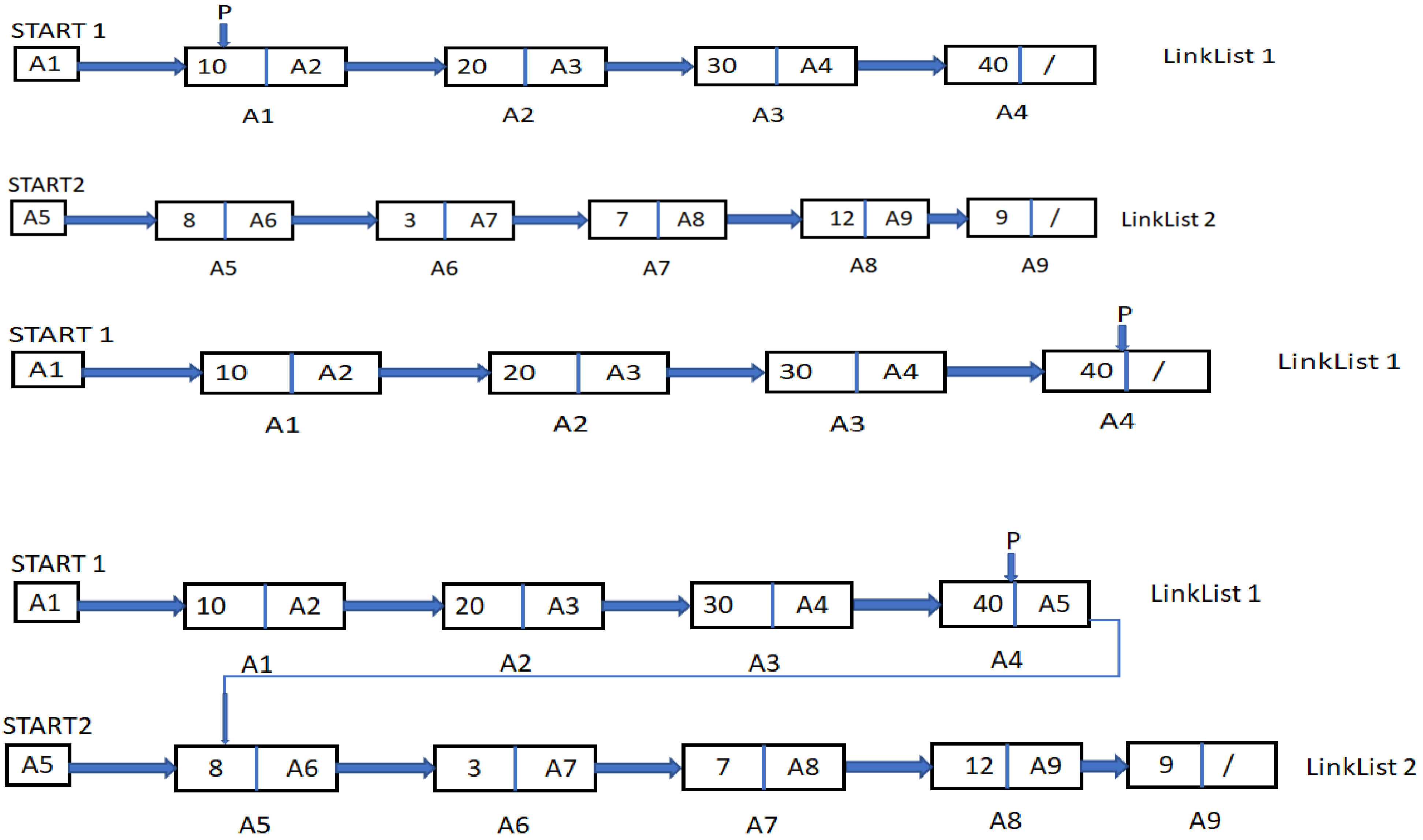
```
ALGORITHM ConcatLinkList(Start1, Start2)
BEGIN:
    IF Start1 == NULL THEN
        RETURN Start2
    ELSE
        P = Start1
        WHILE P→Next! = NULL
                    P = P→ Next
                    P→Next = Start2
    RETURN Start1
END;
```

Explanation:

In the above algorithm initially, there are two input Linked List. Start1 has the starting address of first Linked List and Start2 has starting address of second Linked List. A temporary pointer P is taken which traverses till the last node of first Linked List and stops at last node address. Then the Next part of last node is assigned the address of first node of second Linked List via Start2

Single/Linear Linked List

Concatenation of two Linked List



Single/Linear Linked List

22) Ordered Insertion

It is an operation, which inserts an element at its correct position in a sorted linked list.

ALGORITHM OrderInsert (START, Key)

BEGIN:

```
        Q = NULL
        P = START

        WHILE P != NULL ______ Key >= P→Info DO
                Q = P
                P = P→Next

        IF ______ THEN
                InsAft(Q, Key)
        ELSE
                InsBeg(START, Key)
```

Single/Linear Linked List

22) Ordered Insertion

It is an operation, which inserts an element at its correct position in a sorted linked list.

ALGORITHM OrderInsert (START, Key)

```
BEGIN:
        Q = NULL
        P = START
        WHILE P != NULL && Key >= P→Info DO
                Q = P
                P = P→Next
        IF _____ THEN
                InsAft(Q, Key)
        ELSE
                InsBeg(START, Key)
END;
```

Single/Linear Linked List

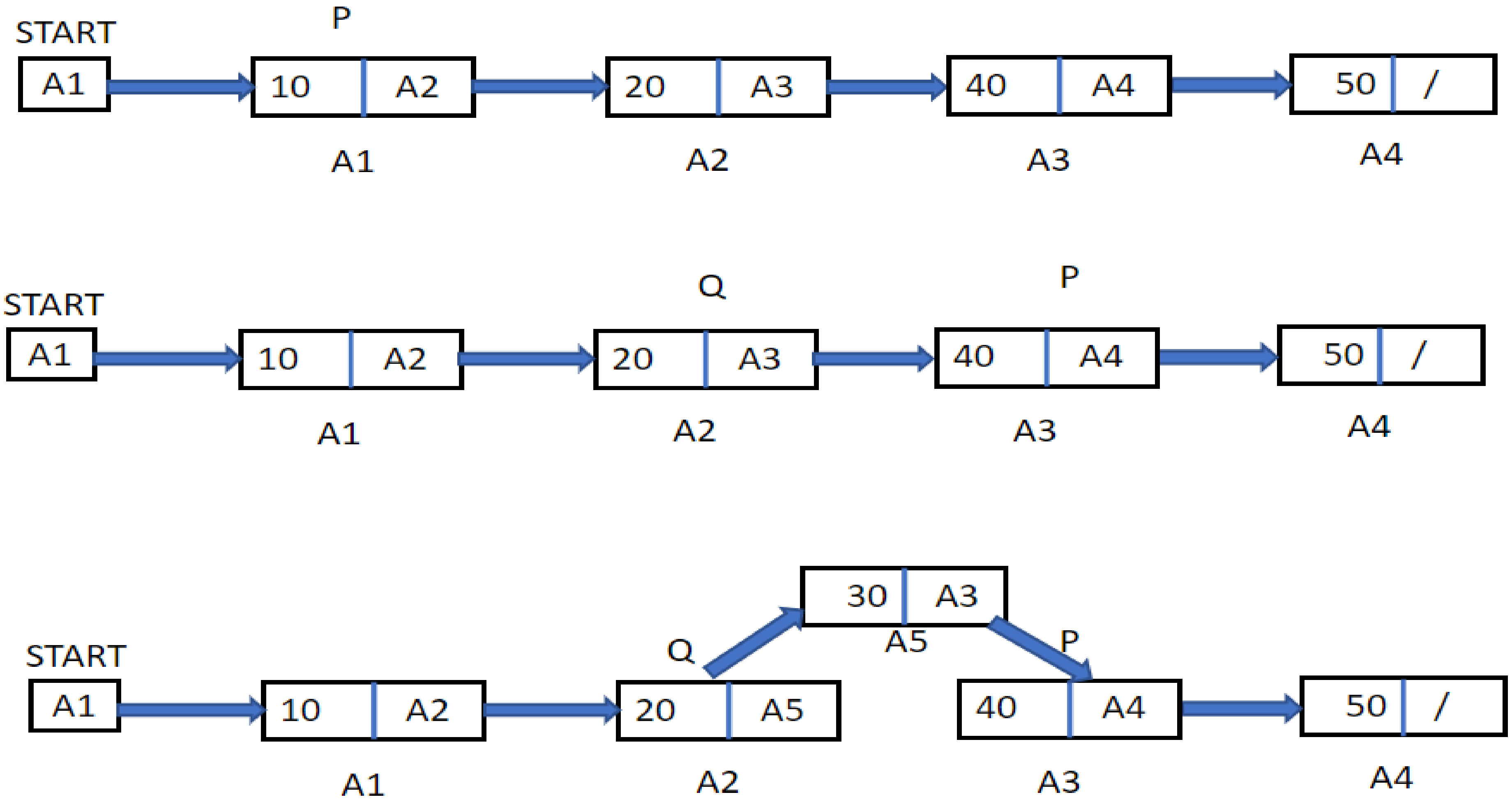
Ordered Insertion

It is an operation, which inserts an element at its correct position in a sorted linked list.

```
ALGORITHM OrderInsert (START, Key)
BEGIN:
        Q = NULL
        P = START
        WHILE P != NULL && Key >= P→Info DO
                Q = P
                P = P→Next
        IF Q != NULL THEN
                InsAft(Q, Key)
        ELSE
                InsBeg(START, Key)
END;
```

Single/Linear Linked List

Ordered Insertion



Single/Linear Linked List

Ordered Insertion

Complexity of Operation

No of Address node Adjustment = 2

Time Complexity = O(N)

Space Complexity = O(1)

Single/Linear Linked List

Merging-

It is an approach in which we combine two sorted linked list and generates third sorted list. Here first list contains m nodes, second list contains n nodes and the final sorted list contains m+n nodes.

```
 ALGORITHM Merging (START1, START2)
BEGIN:
P1 = START1
P2 = START2
START3 = NULL
 WHILE P1 != NULL AND P2 != NULL DO
        IF P1 →Info < = P2 →Info THEN
                InsEnd(START3, P1→Info)
                P1 = P1→Next
        ELSE
                InsEnd(START3, P2→Info)
                P2 = P2→Next
```

Single/Linear Linked List

Merging Continues…..-

```
WHILE P1 != NULL DO
    InsEnd(START3, P1→Info)
    P1 = P1→Next
WHILE P2 != NULL DO
    InsEnd(START3, P2→Info)
    P2 = P2→Next
END;
```

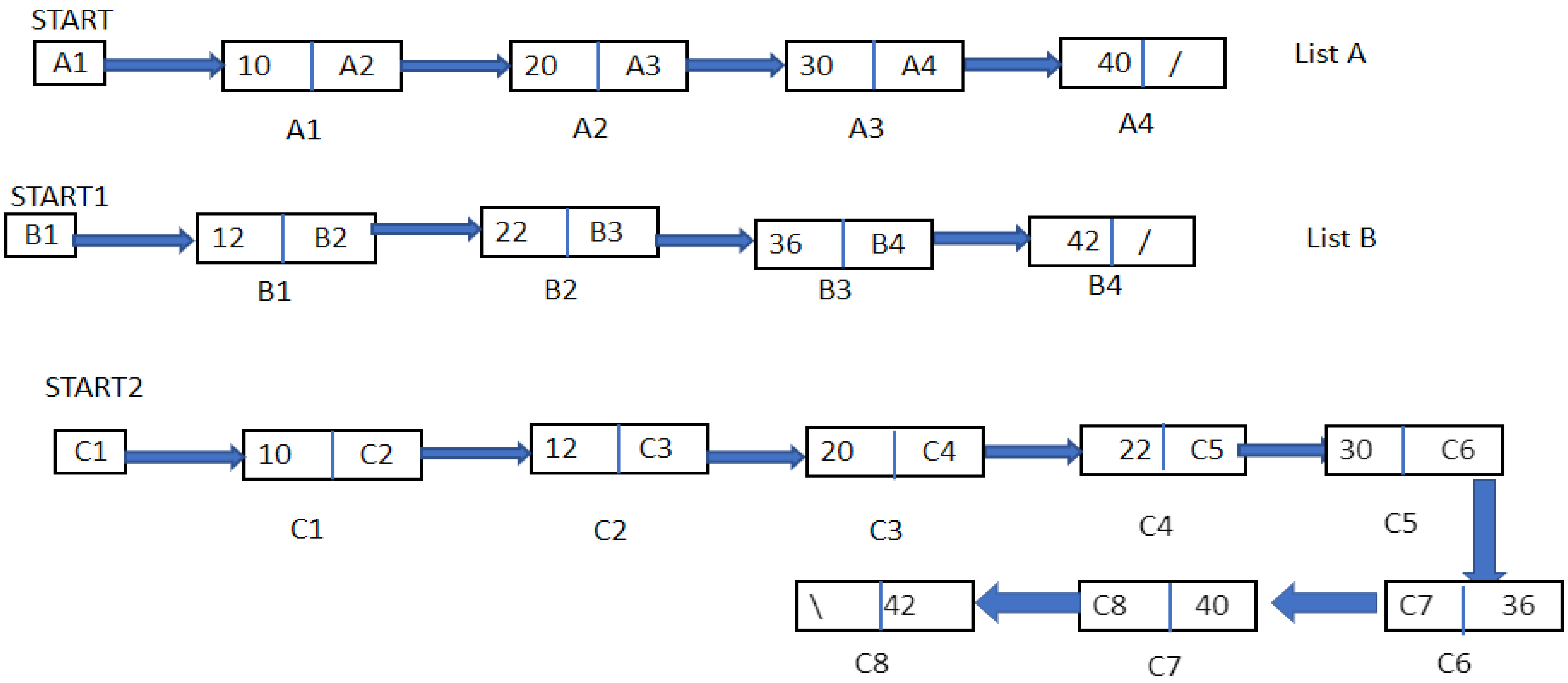
Complexity of Operation

Time Complexity = O(m+n)

Space Complexity = O(m+n) extra space

Single/Linear Linked List

Merging-



Single/Linear Linked List

Split list from middle-

It is an operation which splits a given linked list from its mid node and produces two linked list where first list ends at mid node and second list starts from the node next to mid node.

```
Split List From Mid
Method-1
ALGORITHM SplitMid(START1)
BEGIN:
Count = NodeCount(START1)
P = START1
i = 1
```

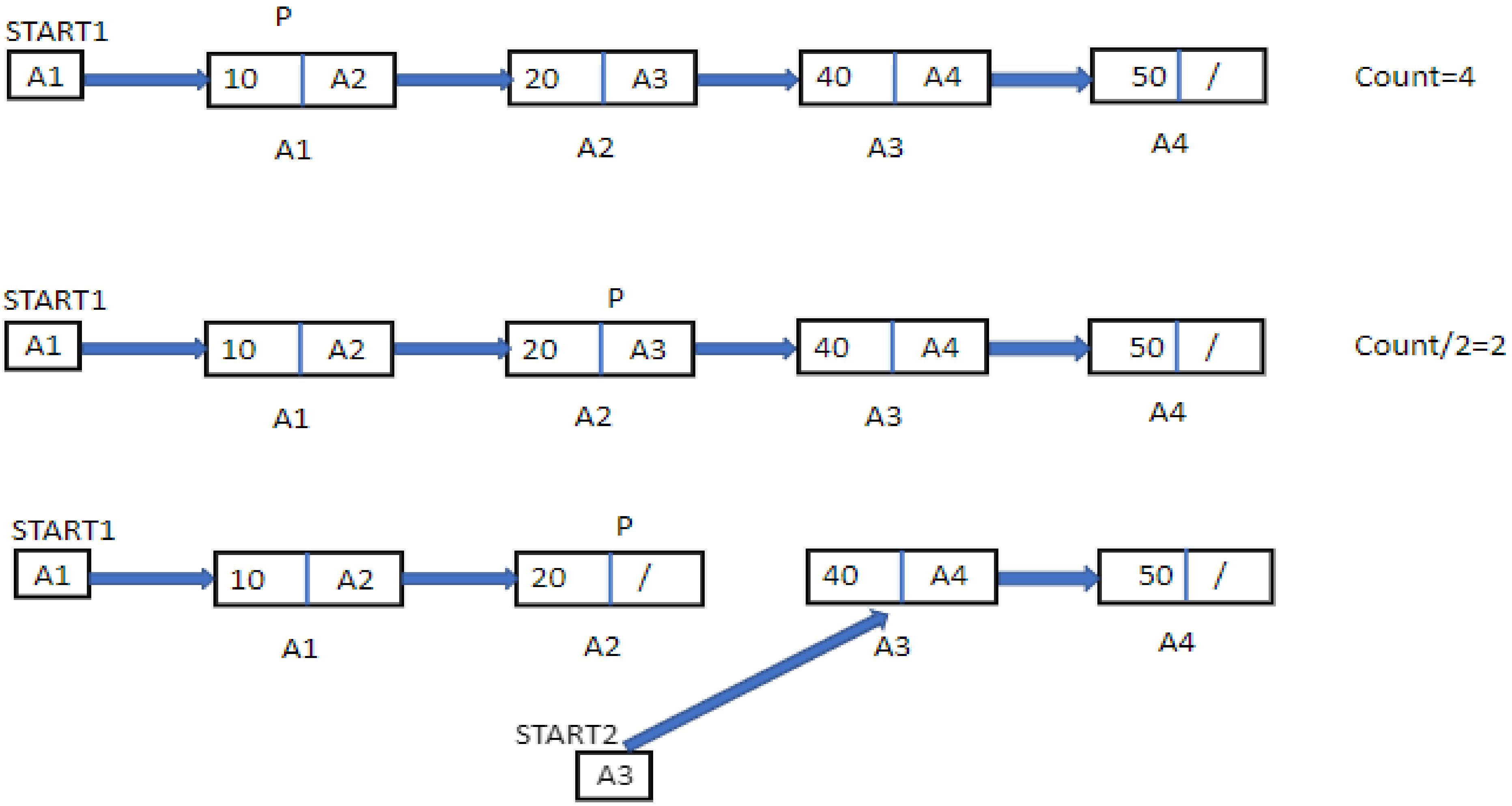
Single/Linear Linked List

Split list from middle-

```
WHILE i < Count/2 DO
        P= P→Next
        i = i + 1
IF Count != 1 THEN
        START2 = P→Next
        P→Next = NULL
ELSE
        WRITE “One node in List”
END;
```

Single/Linear Linked List

Split list from middle-



Single/Linear Linked List

Split list from middle-

Complexity of Operation:

No of address node Adjustment = 1

Time Complexity = O(N)

Space Complexity = O(1)

Single/Linear Linked List

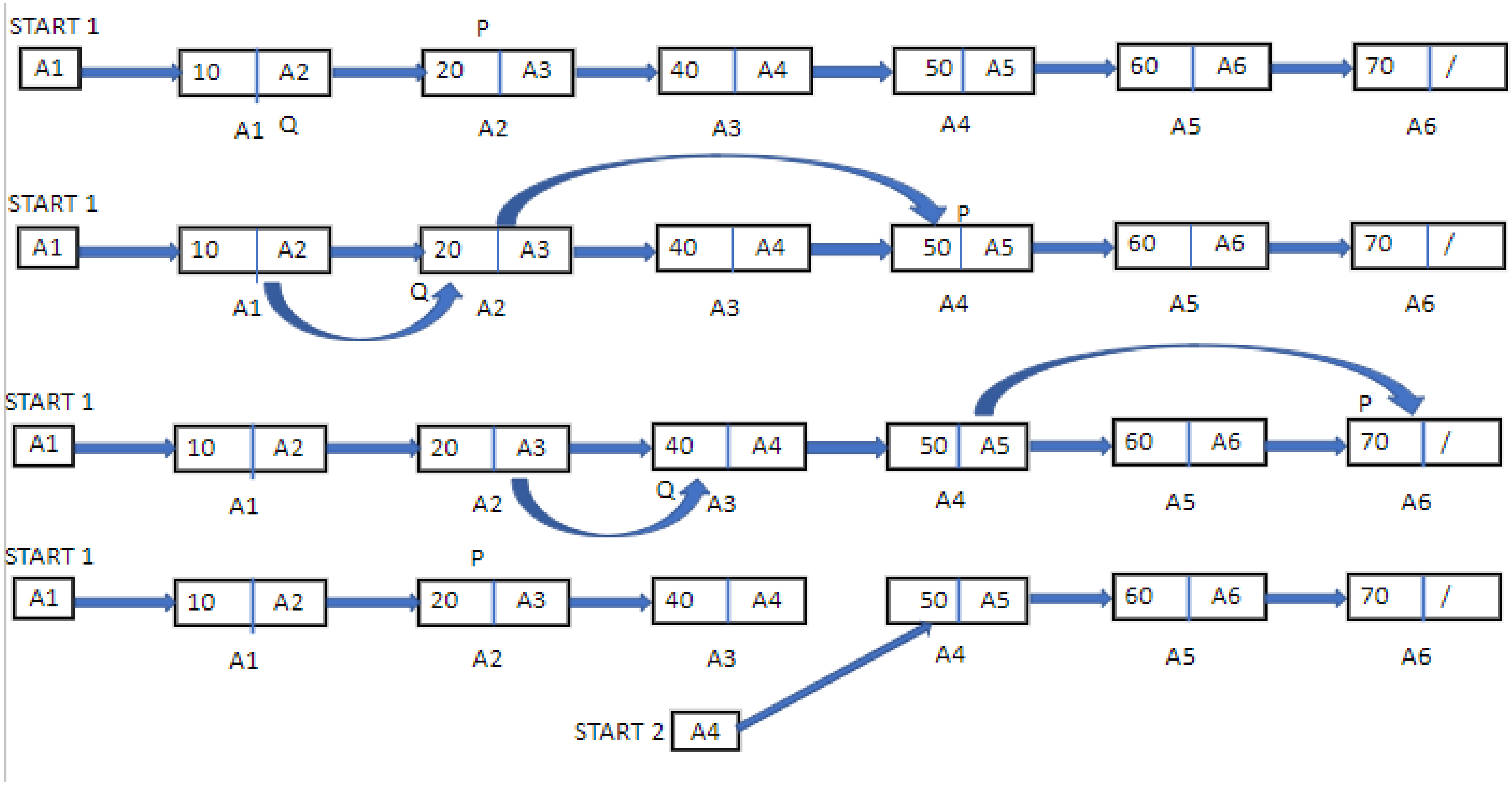
Split list from middle(method 2)-

```
ALGORITHM SplitMid(START1)
BEGIN:
Q = START1
P = START1→ Next
WHILE P != NULL || P→ Next != NULL DO
Q = Q→ Next
P = P→ Next → Next


START2 = Q→ Next
Q→ Next = NULL
END;
```

Single/Linear Linked List

Split list from middle(method 2)-



Single/Linear Linked List

Split list from middle(method 2)-

Complexity of Operation:

No of address node Adjustment = 1

Time Complexity = O(n/2) = O(n)

Space Complexity = O(1)

Single/Linear Linked List

Union Of Two Sorted List-

The Union of two sorted linked list is the new linked list which contains all the elements which are present in at least one of the linked lists.

```
ALGORITHM Union (START1, START2)
BEGIN:
P1 = START1
P2 = START2
    START3 = NULL
      WHILE P1 != NULL AND P2 != NULL DO
        IF P1 →Info <  P2 →Info THEN
            InsEnd(START3, P1→Info)
            P1 = P1→Next
```

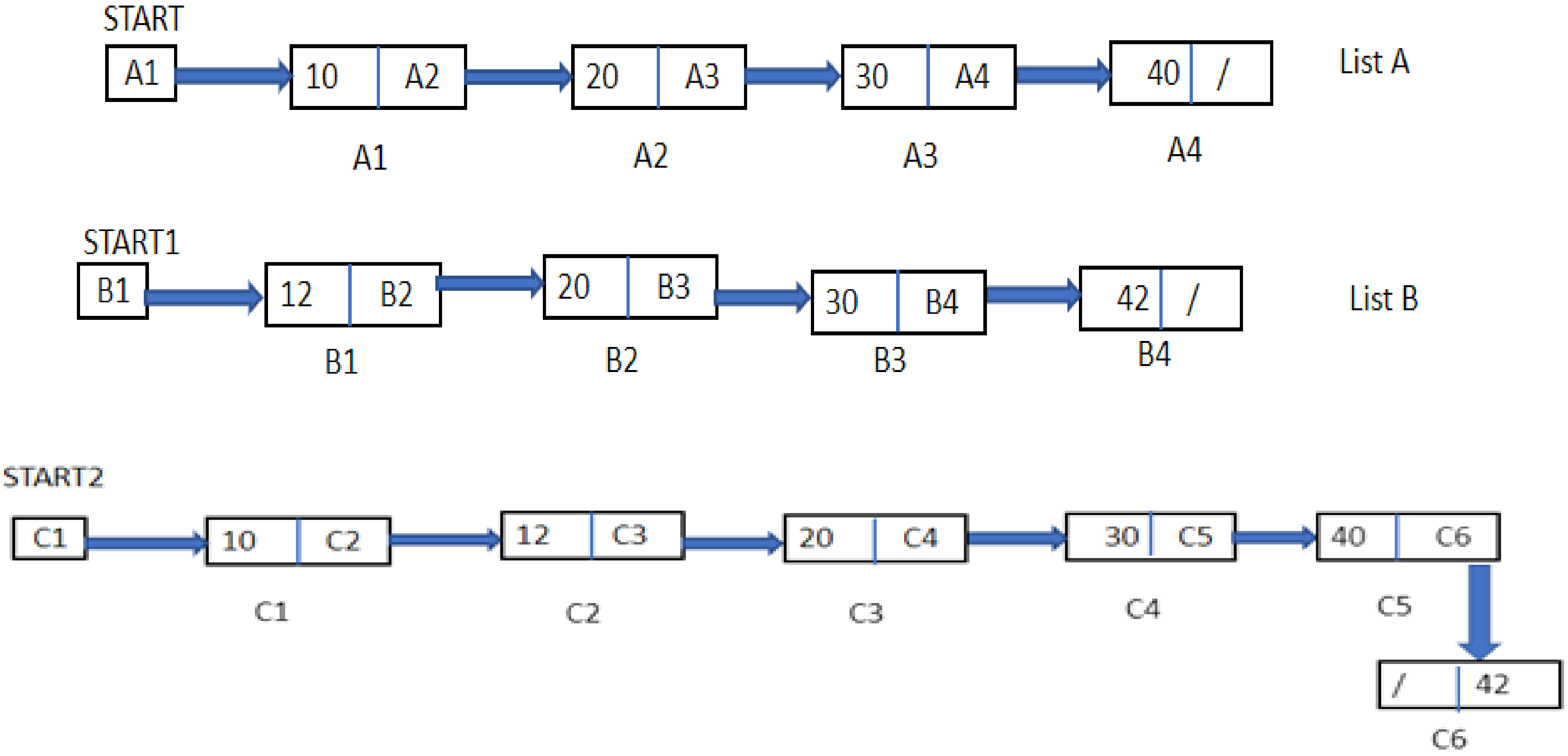
Single/Linear Linked List

Union Of Two Sorted List-

```
        ELSE IF P2 →Info < P1→Info THEN
            InsEnd(START3, P2→Info)
            P2 = P2→Next
        ELSE
            InsEnd(START3, P1→Info)
            P1 = P1→Next
            P2 = P2→Next
    WHILE P1 != NULL DO
        InsEnd(START3, P1→Info)
        P1 = P1→Next
    WHILE P2 != NULL DO
        InsEnd(START3, P2→Info)
        P2 = P2→Next
END;
```

Single/Linear Linked List

Union Of Two Sorted List-



Single/Linear Linked List

Union Of Two Sorted List-

Complexity of Operation

Time Complexity = O(m+n)

Space Complexity = O(m+n) extra space

Single/Linear Linked List

Intersection Of Two Sorted List-

This algorithm performs set intersection operation on given two Linked List.

ALGORITHM SetIntersectionLL(START1, START2)

BEGIN:

```
    P1 = START1
    P2 = START2
    START3 = NULL
    WHILE  P1 ! = NULL && P2 ! = NULL DO
        IF P1→ Info ! =  P2 → Info THEN
            IF (P1 → Info < P2 → Info)
                        P1 = P1→Next
                ELSE
                        P2=P2 → Next
    ELSE
```

Single/Linear Linked List

Intersection Of Two Sorted List-

```
        Insertend(START3, P1→Info)
        P1=P1→Next
          P2=P2→Next
    RETURN START3
END;
```

Explanation: In the above algorithm initially there are two input Linked List. START1 has the starting address of first Linked List and START2 has starting address of second Linked List. A temporary pointer P1 is taken which traverses till the last node of first Linked List and P2 is taken which traverses till the last node of second Linked List. Every Information field of first list is compared with second and the common Information field is picked. The common Information field is then added in third Linked List with START3 pointer using Insertend( ).

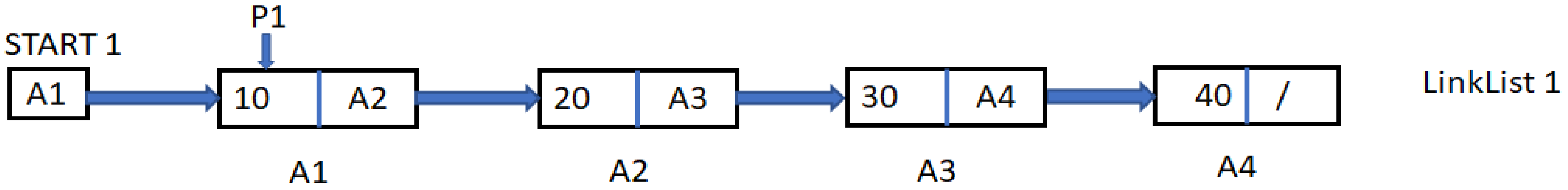
Complexity:

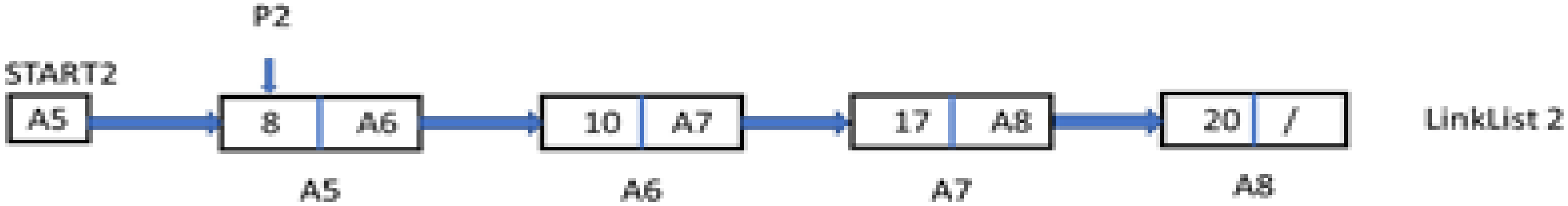
Time Complexity: O(m + n)

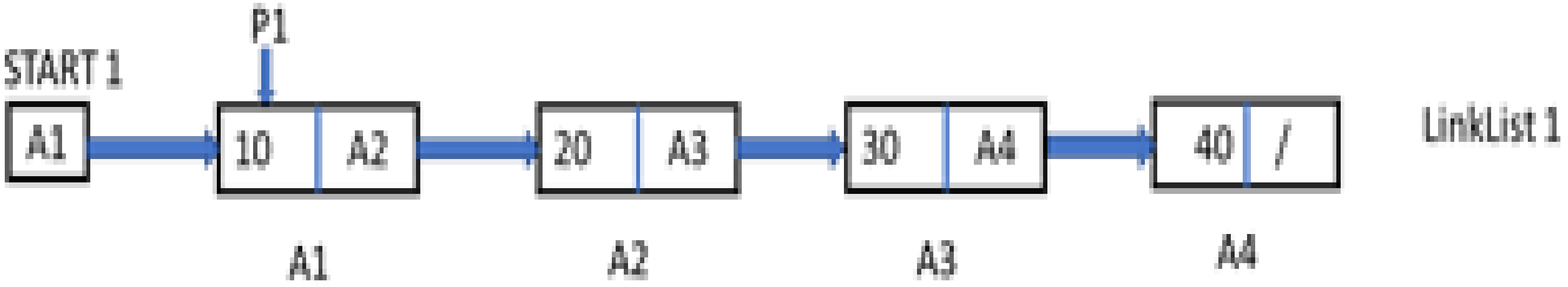
Space Complexity: O(x) where x is maximum between m and n

Single/Linear Linked List

Intersection Of Two Sorted List-

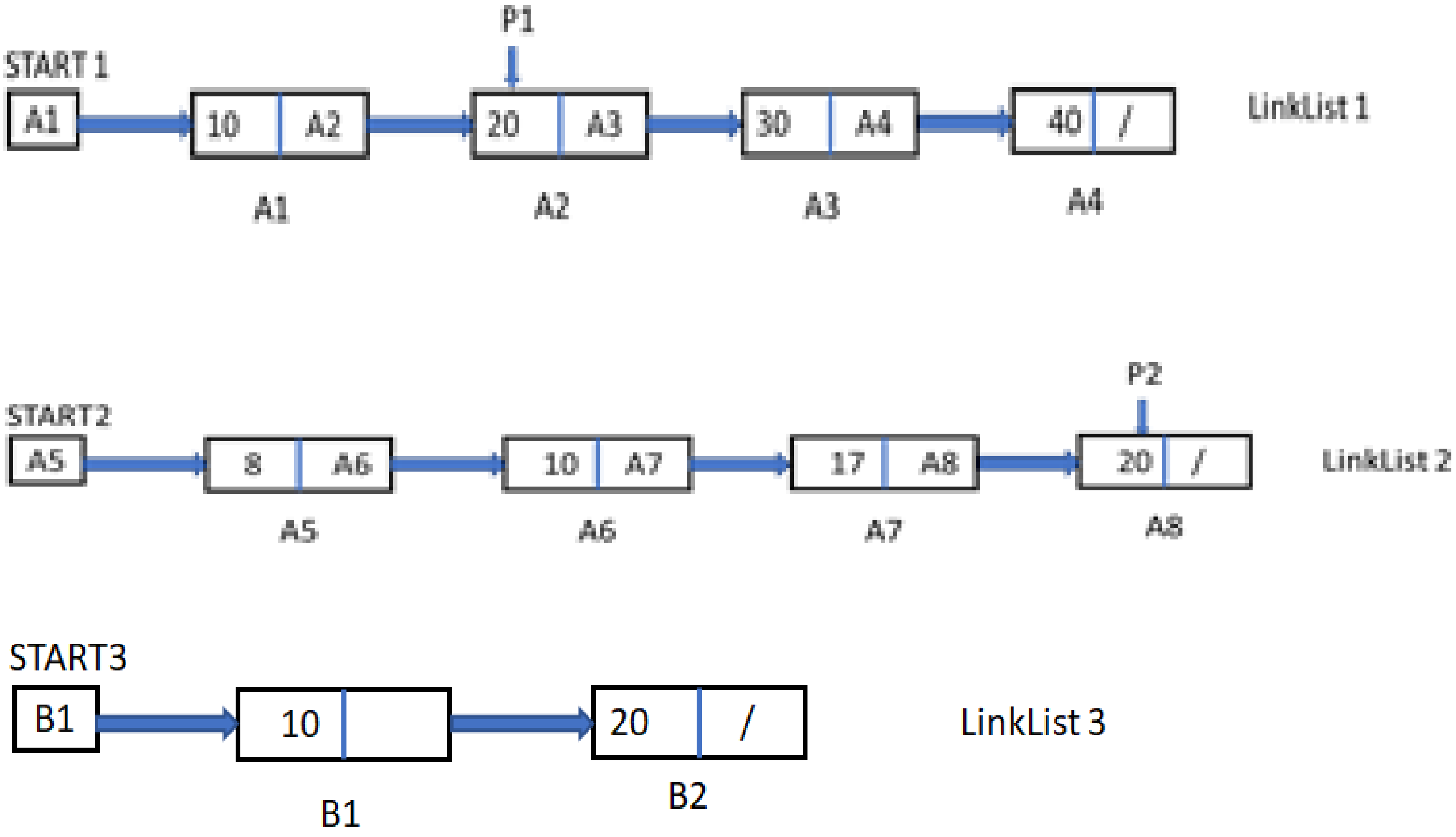






Single/Linear Linked List

Intersection Of Two Sorted List-



Single/Linear Linked List

Set Difference Of Two sorted Link List-

Considering A – B

```
 ALGORITHM SetDifference(START1, START2)
BEGIN:
P1 = START1
P2 = START2
START3 = NULL
            WHILE P1 != NULL DO
WHILE P2 != NULL DO
IF P1 →  Info == P2→Info THEN
BREAK
ELSE
P2 = P2→Next
IF P2 == NULL THEN
InsEnd(START3, P1→Info)
P1 = P1→Next
END;
```

Single/Linear Linked List

Set Difference Of Two sorted Link List-

Complexity of Operation

Time Complexity = O(m+n) ~ $O(n^2)$

Space Complexity = O(m)

Single/Linear Linked List

Reverse the Linked List (In Place)

Problem: Given address to the head node of a linked list, the task is to reverse the linked list. (We need to reverse the list by changing the links between nodes).

Analogy

The weblink can be reversed using browser cache which allow you to hit the back button. By pressing Back button we can reach to the starting web page. In this case when we start to reverse than the last node will be the first node and link direction will be reversed.

Solution

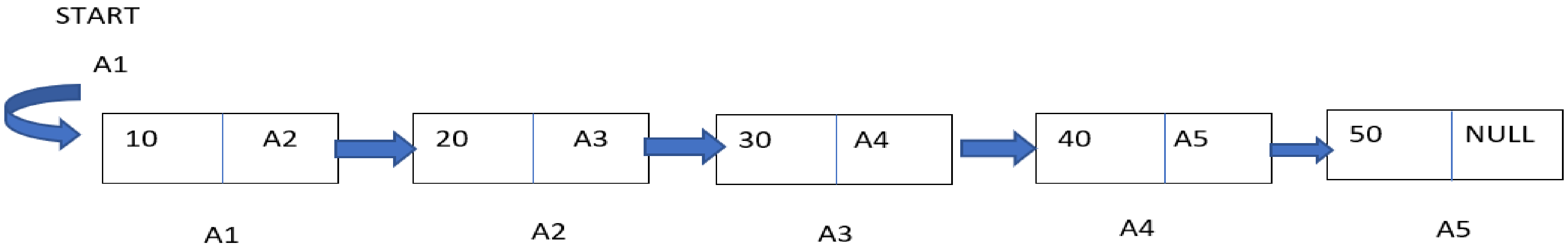
Given address of the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing the links between nodes.

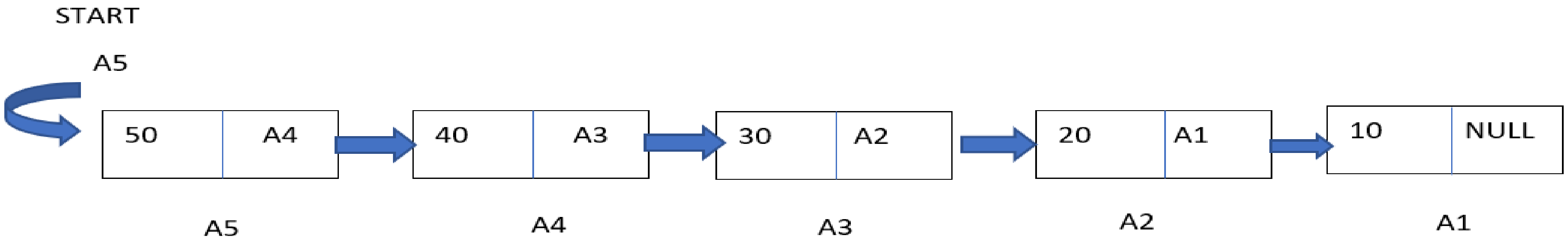
Initialize three nodes R as NULL, P as start and Q as NULL.

Iterate through the linked list.

Single/Linear Linked List

Reverse the Linked List (In Place)





Single/Linear Linked List

Reverse the Linked List (In Place)

```
ALGORITHM  ListReverse(START)
BEGIN:
    IF START= NULL THEN
            WRITE (“List is empty”)
    ELSE
            P = START, R = NULL
         WHILE P! = NULL DO
            Q = P→ Next
              P→ Next = R
              R = P
               P = Q
          START = R
END;
```

Single/Linear Linked List

Reverse the Linked List (In Place)

Complexity of Operation

Time Complexity = O(n)

Space Complexity = O(1)

Single/Linear Linked List

Searching In a Link List-

Analogy

Ten students having roll number from 1 to 10 are standing in line in increasing order holding hands of each other. In other words, student one has held the hand of second one, second student had held the hand of third one and so on. The first student can be considered as the base or starting point. Let we have to search student name Rahul among them, then student here can be regarded as node of list while hand can be considered as link for searching.

Single/Linear Linked List

Searching In a Link List-

ALGORITHM Searching (START, item)

```
BEGIN:
    P = START
    WHILE P→ next! = NULL DO
        IF P→ info == item THEN
                WRITE (“Search successful”)
                RETURN P
           ELSE
                P = P→ next;
     RETURN NULL
 END:
```

Single/Linear Linked List

Searching In a Link List-

Complexity of Operation

Time Complexity = O(n)

Space Complexity = O(1)

Single/Linear Linked List

Middle Element In a Link List-

```
ALGORITHM Find (START)
BEGIN:
    P = START
    count = 0
    WHILE P!= NULL DO
        count++
        P = P →next;
    count = count/2
    Q = START
    i = 1
    WHILE Q != count DO
        Q = Q →next
        i = i + 1
    RETURN Q
END;
```

Single/Linear Linked List

Middle Element In a Link List-

Complexity of Operation

Time Complexity = O(n)

Space Complexity = O(1)

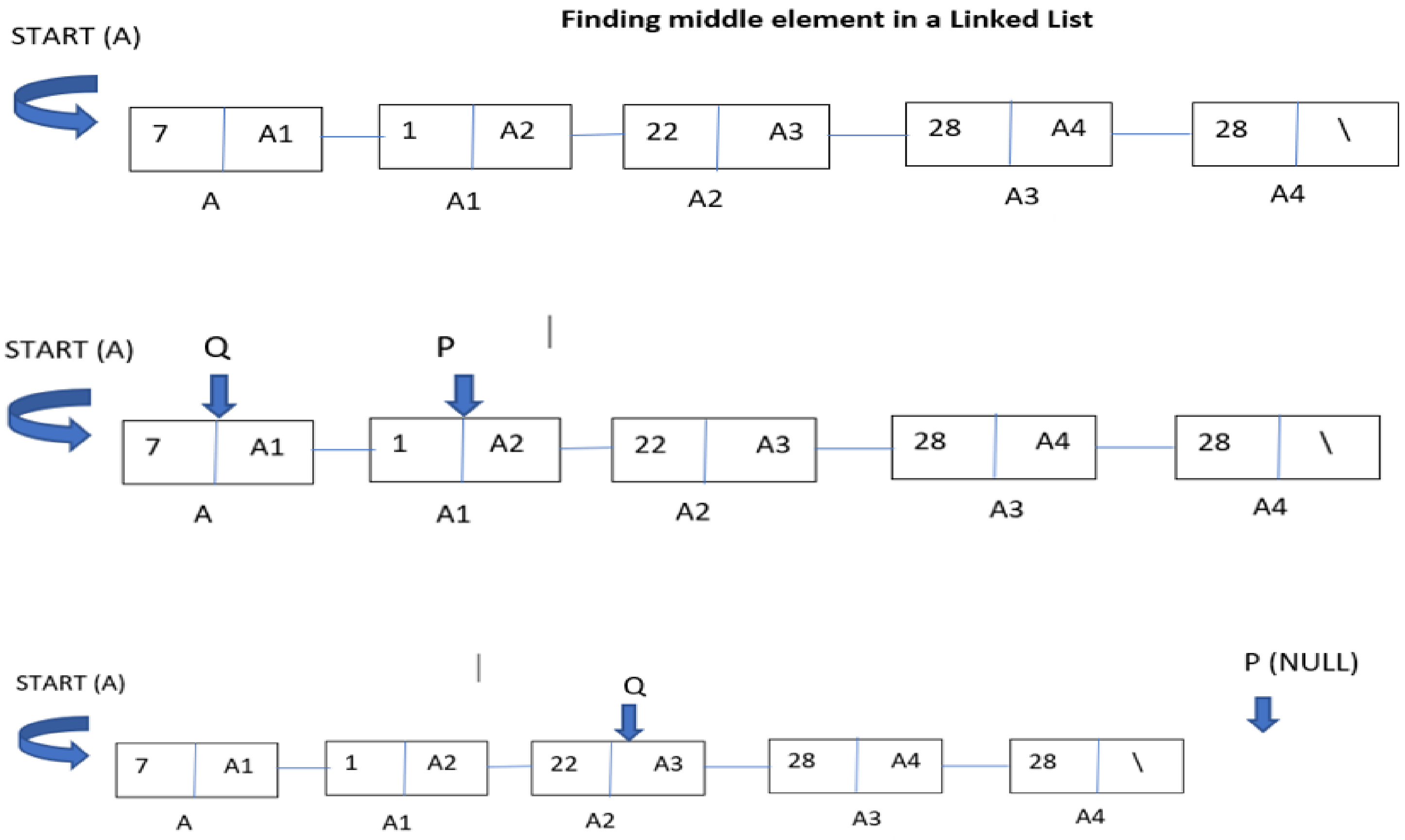
Single/Linear Linked List

Middle Element In a Link List(slow and fast pointer concept)-

```
ALGORITHM FindMiddle(START)
BEGIN:
    Q = START
    P = START→next
    WHILE P! = NULL || P→next != NULL DO
        Q = Q→next
        P = P→next→next
    RETRUN Q
END
```

Single/Linear Linked List

Middle Element In a Link List(slow and fast pointer concept)-


Finding middle element in a Linked List

Single/Linear Linked List

Middle Element In a Link List-

Complexity of Operation

Time Complexity = O(n)

Space Complexity = O(1)

1 → 2 → 3 → 4 → 5 → 6

Finding Middle Element

Approach 1:

Step 1: Count all nodes

step 2: Traverse till half of count

Finding Middle Element

1 → 2 → 3 → 4 → 5 → 6

Approach 2: . Use of fast and slow Pointer

Step1: Initialize fast and Slow Pointers at first and Second node respectively

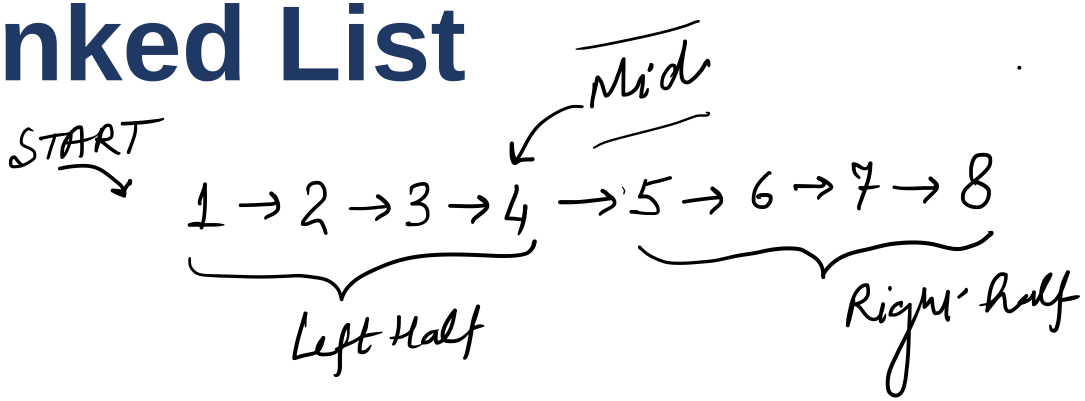
Traverse with slow pointer moving 1 step fast pointer moving 2 steps

Stop if fast Pointer either reached or Crossed the last node

Step3 : Return the address of slow pointer

Single/Linear Linked List

Binary Search



```
ALGORITHM   Binary Search(START, Key)
BEGIN:
    IF START != NULL THEN
        Mid = MiddleElement(START)
        IF Key == Mid->info THEN
            RETURN Mid
        ELSE IF Key < Mid->info THEN
                Mid->Next = NULL                 } Left half search
                Binary Search(START, Key)
        ELSE
                START = Mid->Next                } Right half Search
                Binary Search(START, Key)
    ELSE
        RETURN NULL
END;
```

Single/Linear Linked List

Binary Search

Complexity of Operation

Time Complexity =

Space Complexity =

Other Disadvantage

Single/Linear Linked List

Loop Detection-

Problem: - Write an algorithm to check if the Linked List has a loop or not. If Linked List has a loop then find a length of loop and also find start point of loop.

It is already understood that if there is loop or cycle in the Linked List then there is no end. It means in this case normal traversal does not work because there is no idea where to stop.

Method1- Marking Visited Node

It is considered that along with the information and next address field, the nodes in the linked list have an additional field named as "visited". It is considered that while creating the linked list, the visited field of all nodes have been set as false.

LOGIC-

1-First initialize visited field of all nodes is FALSE.

2 After that traverse every node and set visited field of node is true.

3- If again comes true then it means loop exist otherwise not.

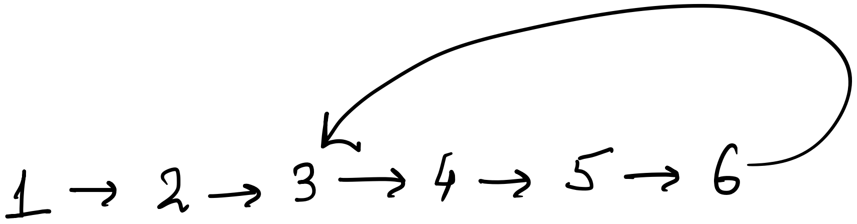
Single/Linear Linked List

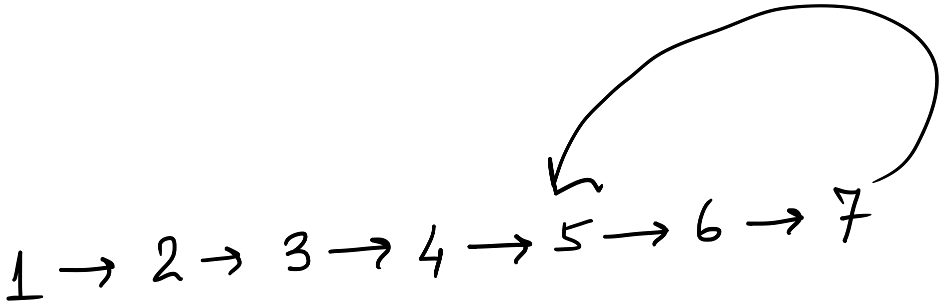
Loop Detection-

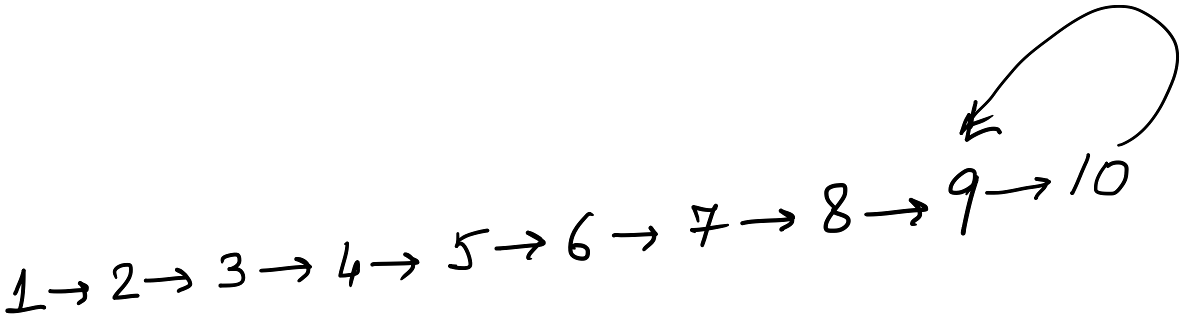
```
ALGORITHM LoopFind(START)
BEGIN:
    P = START
    While (P! =NULL && P→link! =NULL && P→visited==FALSE) DO
        P→visited=TRUE
        P=P→link
    If(P→visited==TRUE)
        WRITE (loop found)
    ELSE
        WRITE (loop not found)
    END;
```

Here we traverse the one time the entire link list so complexity of link list is O(N) but initialize visited field of all nodes so it takes O(N) extra space.

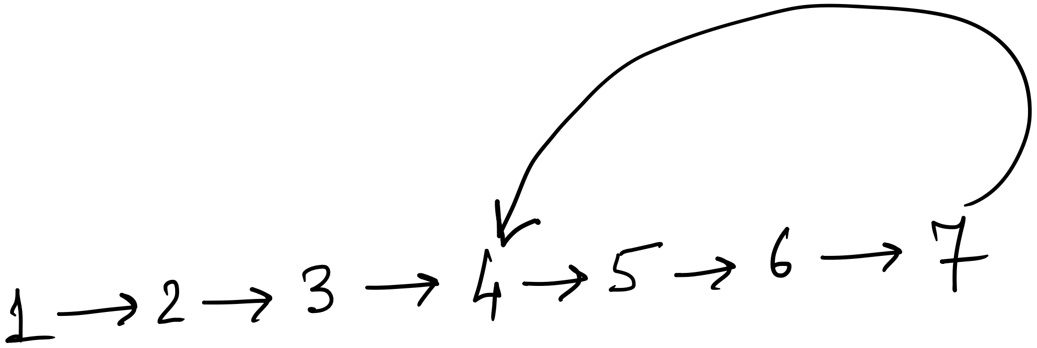
Finding Loop



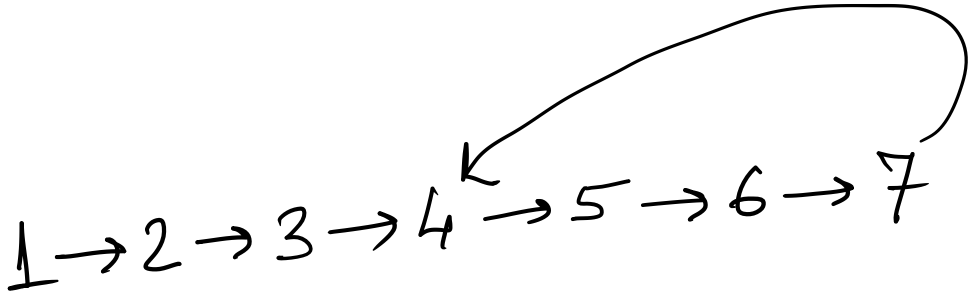


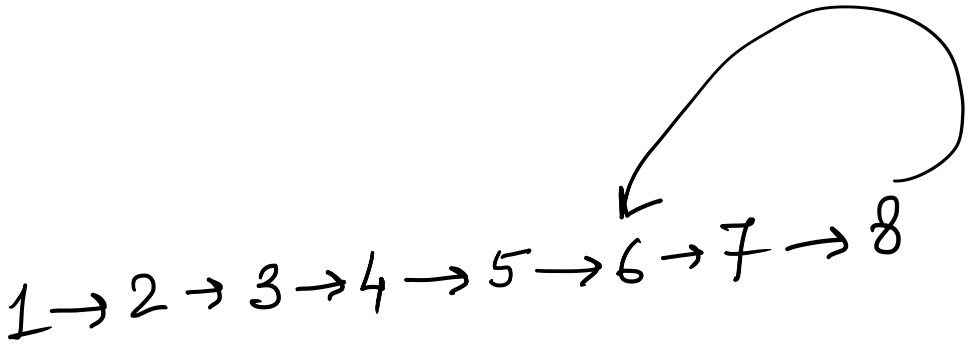


length of loop



Starting Point of Loop

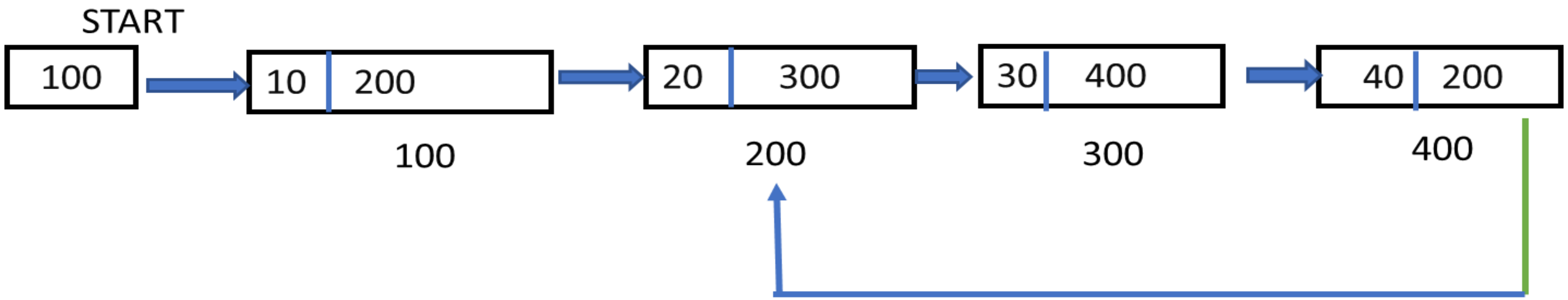


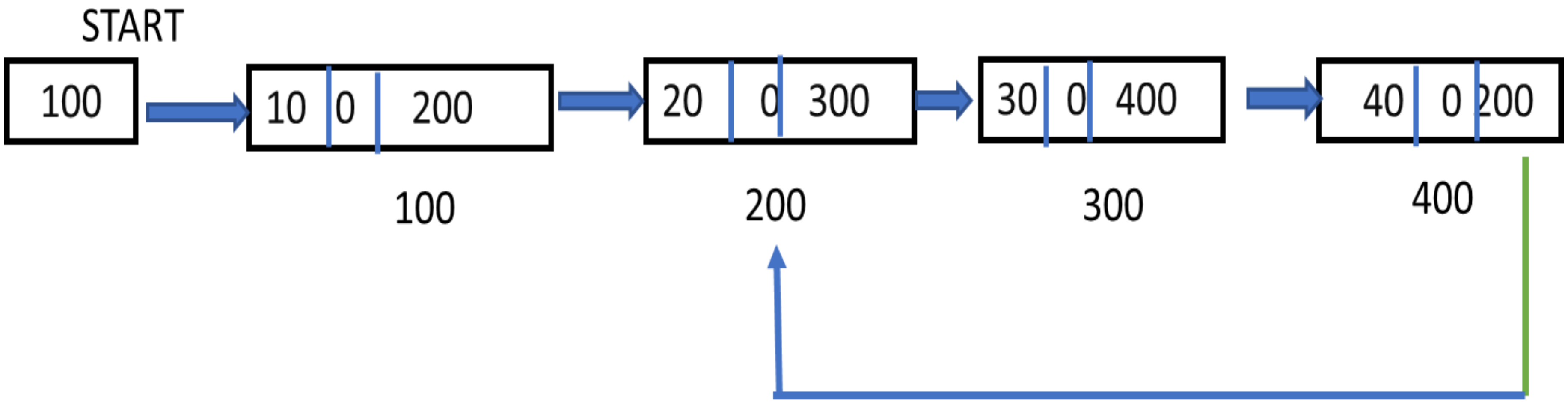


Single/Linear Linked List

Loop Detection-

.





Single/Linear Linked List

Loop Detection(METHOD-2)

Method-2 (Reversing the List)

LOGIC-

1-Take a pointer P which points START of node.

2- Reverse the entire list

3-If there is a loop in the list, again P points START of node (because cycle exist)

4-If P not points START of node then there is no loop/cycle exist

5-Again reverse the list to get original list

Single/Linear Linked List

Loop Detection(METHOD-2)

```
ALGORITHM LoopFind(START)
BEGIN:
    P = START
    Reverse(P)
        If(P==START) THEN
            WRITE (loop find)
        ELSE
            WRITE (loop not found)
```

This algorithm again takes O (N) time and constant extra space because just traverse the list single time.

Single/Linear Linked List

Loop Detection(METHOD-3)

(Tortoise and Hare Approach or concept of slow and fast pointer)

Analogy-

Once upon a time there live a hare who runs very fast due to which he was very proud of this, there was a tortoise who was very hard worker. One day due to some reason they both quarreled on some issue and it was decided one who will cover a distance of 50 meter first will won the race. Hare, being proud of his ability to run fast agreed at once and Tortoise also agreed at last. They both start the race but hare was over confident due to which he started to take rest under the tree. **At average, it was taken that tortoise was able to take two steps and hare was only able to take only one step. So at last Hard work wins.**

Single/Linear Linked List

Loop Detection(METHOD-3)

LOGIC-

1-Here take two pointers slow and fast pointers and both pointers point to START node.

2-Move slow pointer to normal speed but fast pointer to double speed.

3-If fast pointers reach to NULL then it means there is no cycle because fast pointers move to double speed and that time slow pointer points middle of the list.

4-If slow and fast pointers are equal then there is a loop.

Single/Linear Linked List

Loop Detection(METHOD-3)

```
ALGORITHM LoopFind (START)
BEGIN:
P = START
Q = START
While P!= NULL and P→link != NULL DO
        Q = Q→link
        P = P→link→link
        If (P == Q)
                Return TRUE
Return FALSE
END;
```

Complexity- here is the complexity is O(N) because traverse the list only once by slow pointer.

Single/Linear Linked List

Length of Loop

ALGORITHM LoopFind(START)

BEGIN:

Single/Linear Linked List

Length of Loop

ALGORITHM LoopFind(START)

BEGIN:

Single/Linear Linked List

Start Point of Loop

ALGORITHM StartPoint(START)

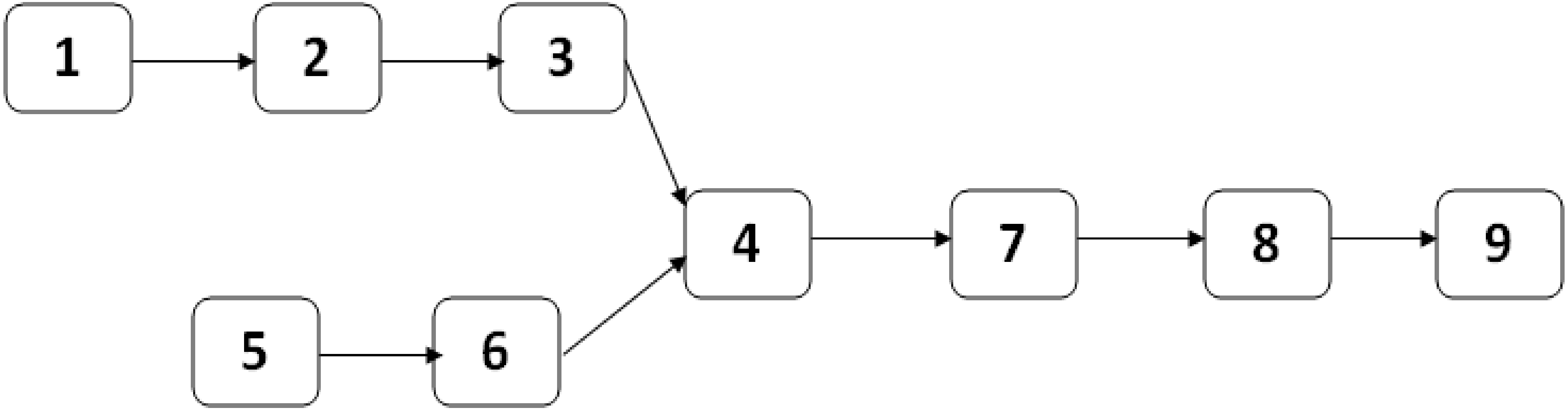
BEGIN:

Single/Linear Linked List

Merging Point of Loop

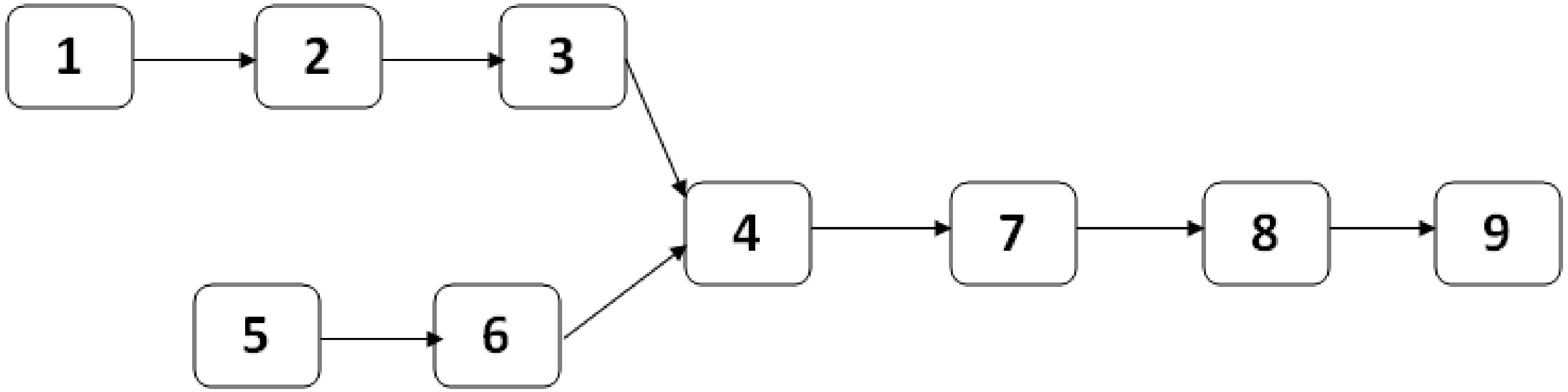
Let us suppose that given two link list that merge at certain point that is shown in diagram below.

a) Find the number of unique elements in each list.
b) Find number of nodes common in each link list.



Single/Linear Linked List

Merging Point of Loop



Single/Linear Linked List

Merging Point of Loop

1-let us suppose that there are (m + c) nodes in first list and (n + c) nodes in second list.

2- c denotes last nodes common in each list.

3-calculate the number of nodes in each list by traversing them linearly. They have m + c and n + c nodes respectively.

4-take difference of two list and this gives excess node in larger list.

5-Move START1 by this difference.

6-now move START1 and START2 forward in a loop and compare the address of nodes they are pointing.

Single/Linear Linked List

Merging Point of Loop

```
ALGORITHM merging Point (START1, START2)
BEGIN:
    P = START1
    Q=START2
    C1=0
    C2=0
    WHILE P! =NULL DO
        P=P→link
        C1 = C1+1
    WHILE Q! =NULL DO
        Q=Q→link
        C2 = C2+1
    P = START1
    Q=START2
    If C1>C2
        FOR I = 0 to c1-c2 DO
            P=P→link
    ELSE
        FOR I=  0 to c2-c1 DO
            Q=Q→link
    WHILE P! = Q && P! =NULL && Q! =NULL DO
        P =P→link
        Q = Q→link
    RERTURN P
```

Single/Linear Linked List

Palindrome Check-

Following list are palindrome

1 → 2 → 3 → 3 → 3 → 2 → 1

M → A→L→A→Y→A→L→A→M

Constraints – Function should take O(N)Time

Single/Linear Linked List

Palindrome Check-

Write an algorithm to check if a link list is palindrome or not.

Following list are palindrome

1 → 2 → 3 → 3 → 3 → 2 → 1

M → A→L→A→Y→A→L→A→M

Constraints – Function should take O(N)Time

The simplest solution is that to take reverse function.

Solution-

1-first find middle of link list

2-then compare first half and second half of link list.

3- finally again reverse the second half of list to store original one.

Total complexity O(N)

Single/Linear Linked List

Palindrome Check-

```
ALGORITHM palindrome(START)
BEGIN:
P = START
Ispaindrome = TRUE
Mid = middle(P)
Q=mid
Reverse(mid)
//if number of elements is odd then let first half have the extra elemrnt
//check if corresponding elements of the list pointed by START and mid are equal
While mid != NULL repeat
        If(mid→info != P→info)
                Ispalindrome= FALSE
        P =P→link
        Mid=mid→link
  Reverse(Q)
```

Single/Linear Linked List

Priority Queue Implementation-

For priority queue implementation using link list, add extra field priority in the node and follow these steps-

1. If start contains NULL then it means first node of list and simply insert it.
2. If start does not contains NULL then first compare priority field and insert according to the priority.
3. In case of deletion just free the first node as lowest priority.

```
Algorithm pqInsert(start,P)
        If start→priority < P→priority
                P→link = start
                Start = P
        Q = start
    While Q→link !=NULL && Q→link→priority > P→ priority Repeat
                Q=Q→link
        P→link = Q→link
        Q→link = P
```

Single/Linear Linked List

Priority Queue Implementation-

```
Algorithm pqDelete(start)
        P=start
        Start=start→link
        Free(P)
```

Complexity- Algorithm takes O(N) time because traverse the list and compare the priority in just single pass.

Single/Linear Linked List

Sorting of Linked List

Bubble Sort

```
ALGORITHM SortingLinkedList(START)
BEGIN:
    N = CountNodes(START)
    FOR i = 1 TO N-1 DO
        P=START
        FOR j = 1 TO N-i DO
                IF P→info > P→Next→info THEN
                        Exchange(P→info, P→Next→info)
            P=P→Next
    END;
    Complexity- O(n^2)
```

Single/Linear Linked List

Selection Sort

Method-1

```
ALGORITHM SortingLinkedList(START)
BEGIN:
    N=CountNodes(START)
    P=START
    FOR i = 1 TO N-1 DO
        Min = P
        Q = Min→Next
        FOR j = 1 TO N-i DO
                IF Q→info < Min→ info THEN
                    Min = Q
            Q=Q→Next
        Exchange(P→info, Min→info)
        P=P→Next
END;
```

Single/Linear Linked List

Addition Of Numbers-

Ex1

1 → 2 → 3

3 → 6 → 9

4 → 9 → 2

Ex2

4 → 0 → 8

9 → 3 → 6

1 → 3 → 4 → 4

Single/Linear Linked List

Addition Of Numbers-

Ex1:

$$\begin{array}{l} 4 \rightarrow 8 \rightarrow 6 \\ 3 \rightarrow 5 \rightarrow 1 \\ \hline 1 \rightarrow 3 \rightarrow 5 \\ \hline \end{array}$$

Ex2

$$\begin{array}{l} 9 \rightarrow 3 \rightarrow 4 \\ 2 \rightarrow 5 \rightarrow 5 \\ \hline 6 \rightarrow 7 \rightarrow 9 \\ \hline \end{array}$$

Single/Linear Linked List

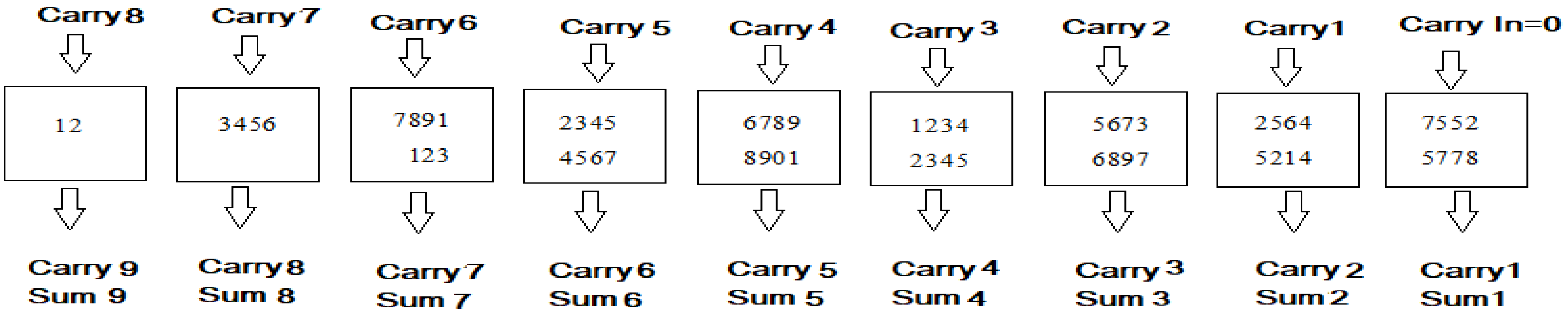
Addition Of very Long Numbers-

There are some situations when we need to do the Arithmetic computations with very long numbers which cannot normally be handled through 2-bit, 4-bit or 8bit integers. To perform computations with such numbers, We can divide numbers into parts which can then be handled through through 2-Byte, 4-Byte or 8-Byte integers.

e.g.

Number 1:- 12 3456 7891 2345 6789 1234 5673 2564 7552

Number 2:- 123 4567 8901 2345 6897 5214 5778



Carry 8 → 12 → Carry 9 Sum 9

Carry 7 → 3456 → Carry 8 Sum 8

Carry 6 → 7891 123 → Carry 7 Sum 7

Carry 5 → 2345 4567 → Carry 6 Sum 6

Carry 4 → 6789 8901 → Carry 5 Sum 5

Carry 3 → 1234 2345 → Carry 4 Sum 4

Carry 2 → 5673 6897 → Carry 3 Sum 3

Carry 1 → 2564 5214 → Carry 2 Sum 2

Carry In=0 → 7552 5778 → Carry 1 Sum 1

Answer of Sum of these numbers: Carry9 Sum9 Sum8 Sum7 Sum6 Sum5 Sum4 Sum3 Sum2 Sum1

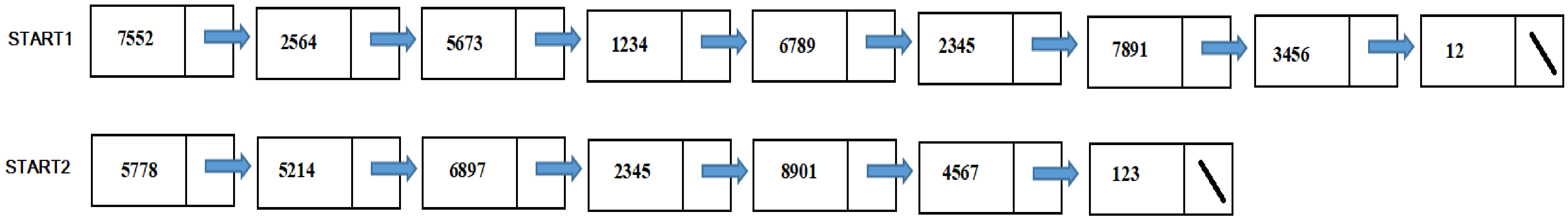
Single/Linear Linked List

Addition Of Long Numbers-

Segregation of number is done in a group of 4 digits because 2-Byte integer has the range -32769 to +32767. Highest number in this range is not the highest number of 5 digit. Hence if 4 digit numbers are added with the carry, the answer remains in the range of integer:

9999+9999+1 = 19999

We can perform such addition using linked list. For the numbers in the example above, two linked lists can be created as in the given diagram below (taking last group in the first node, second last group in the second node, … , first group in the last node). The addition can be performed node by node in linked list 1 and linked list 2. Add previous carry while performing the addition (Initial carry should be taken as 0).



Single/Linear Linked List

Addition Of Long Numbers-

```
ALGORITHM AddLongNumbers(START1, START2)
BEGIN:
    START3=NULL
    P=START1
    Q=START2
    WHILE P!=NULL AND Q!=NULL DO
        Total = P→info + Q→info + Carry
        Sum = Total %10000
        Carry=Total/10000
        InsBeg(START3,Sum)
        P=P→Next
        Q=Q→Next
    WHILE P!=NULL DO
        Total = P→info + Carry
        Sum = Total %10000
        Carry=Total/10000
        InsBeg(START3, Sum)

        P=P→Next
    WHILE Q!=NULL DO
        Total = Q→info + Carry
        Sum = Total %10000
        Carry=Total/10000
        InsBeg(START3, Sum)
        Q=Q→Next
    IF Carry==1 THEN
        InsBeg(START3,Carry)
    RETURN START3
END;
```

Single/Linear Linked List

```
struct node* insert_At_Beg(struct node *first, int x)
{
        struct node *ptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr→data=x;
        ptr→next=NULL;
        if(first==NULL)
        {
                first=ptr;
                return first;
        }
        ptr→next=first;
        first=ptr;
        return first;

}
```

What is the time complexity of insert_At_Beg() function ?

A.O(n) **B.**O(1)

C.O(nlogn) **D.** O(logn)

Single/Linear Linked List

```
struct node* insert_At_Beg(struct node *first, int x)
{
        struct node *ptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr→data=x;
        ptr→next=NULL;
        if(first==NULL)
        {
                first=ptr;
                return first;
        }
        ptr→next=first;
        first=ptr;
        return first;
}
```

What is the time complexity of insert_At_Beg() function ?

A.O(n) [✓ **B.**O(1)]

C.O(nlogn) **D.** O(logn)

Single/Linear Linked List

```
fun(struct node* head)
{
        if(head == NULL)
                return;
        fun(head->next);
        printf("%d ", head->data);
}
```

What is the output of following C function?

A.Print node

B.Print node in reverse order

C.error

D. Print node in increasing order

Single/Linear Linked List

```
fun(struct node* head)
{
        if(head == NULL)
        return;
        fun(head→next);
        printf("%d ", head→data);
}
```

What is the output of following C function?

A.Print node

[✓ **B.**Print node in reverse order]

C.error

D. Print node in increasing order

Single/Linear Linked List

```
struct node* middle(struct node *first)
{
        struct node *ptr=first;
        int c=0;
        while(ptr!=NULL)
        {
                ptr=ptr→next;
                c++;
        }
        c=c/2;
        ptr=first;
        while(c!=1)
        {
                pt=ptr→next;
                c--;
        }
}
```

What is the time complexity of middle() function

A.(n)+c

B.(n/2)+c

C.(3n/2)+c

D. (logn)+c

Single/Linear Linked List

```
struct node* middle(struct node *first)
{
        struct node *ptr=first;
        int c=0;
        while(ptr!=NULL)
        {
                ptr=ptr→next;
                c++;
        }
        c=c/2;
        ptr=first;
        while(c!=1)
        {
                pt=ptr→next;
                c--;
        }
}
```

What is the time complexity of middle() function

A.(n)+c

B.(n/2)+c

C.(3n/2)+c

D. (logn)+c

Single/Linear Linked List

```
struct node* middle(struct node *first)
{
      struct node *ptr,*cpt;
      ptr=cpt=first;
      while(cpt!=NULL && cpt→next!=NULL && cpt→next→next!=NULL)
      {
            ptr=ptr→next;
            cpt=cpt→next→next;
      }
      return ptr;
}
```

What is the time complexity of middle() function

A.(n)+c　　　　**B.** (n/2)+c

C.(3n/2)+C　　　　**D.** (logn)+c

Single/Linear Linked List

```
struct node* middle(struct node *first)
{
      struct node *ptr,*cpt;
      ptr=cpt=first;
      while(cpt!=NULL && cpt→next!=NULL && cpt→next→next!=NULL)
      {
            ptr=ptr→next;
            cpt=cpt→next→next;
      }
      return ptr;
}
```

What is the time complexity of middle() function

A.(n)+c [✓ **B.** (n/2)+c]

C.(3n/2)+C **D.** (logn)+c

Single/Linear Linked List

```
void fun(struct node **head_ref)
{
  struct node *temp = NULL;
  struct node *current = *head_ref;

  while (current != NULL)
  {
    temp = current->prev;
    current->prev = current->next;
    current->next = temp;
    current = current->prev;
  }

  if(temp != NULL )
    *head_ref = temp->prev;
}
```

Assume that reference of head of following doubly linked list is passed to above function 1 2 3 4 5 6.

What should be the modified linked list after the function call?

A. 5 4 3 2 1 6.

B. 6 5 4 3 1 2

C. 2 1 4 3 6 5

D. 6 5 4 3 2 1.

Single/Linear Linked List

```
void fun(struct node **head_ref)
{
  struct node *temp = NULL;
  struct node *current = *head_ref;

  while (current != NULL)
  {
    temp = current→prev;
    current→prev = current→next;
    current→next = temp;
    current = current→prev;
  }

  if(temp != NULL )
    *head_ref = temp→prev;
}
```

Assume that reference of head of following doubly linked list is passed to above function

1 2 3 4 5 6.

What should be the modified linked list after the function call?

A. 5 4 3 2 1 6.

B. 6 5 4 3 1 2

C. 2 1 4 3 6 5

D. 6 5 4 3 2 1.

Single/Linear Linked List

```
struct node* bin_Search(struct node *first, int x)
{
        struct node *mid;
        if(first→next==NULL)
        {
                if(first→data==x)
                        return first;
                ELSE
                        return NULL;
        }
        ELSE
        {
                mid=middle(first);
                if(mid→data==x)
                        return mid;
                ELSE if(mid→data>x)
                {
                        mid→next=NULL;
                        bin_Search(first,x);
                }
                ELSE
                        bin_Search(mid→next,x);
        }
}
```

What is the time complexity and space complexity of following function-

A.O(n) and logn **B.**O(n/2) and logn

C.O(3n/2)+C and logn **D.** O(logn) and O(1)

Single/Linear Linked List

```
struct node* bin_Search(struct node *first, int x)
{
        struct node *mid;
        if(first→next==NULL)
        {
                if(first→data==x)
                        return first;
                ELSE
                        return NULL;
        }
        ELSE
        {
                mid=middle(first);
                if(mid→data==x)
                        return mid;
                ELSE if(mid→data>x)
                {
                        mid→next=NULL;
                        bin_Search(first,x);
                }
                ELSE
                        bin_Search(mid→next,x);
        }
}
```

What is the time complexity and space complexity of following function-

[✓ **A.**O(n) and logn]

B.O(n/2) and logn

C.O(3n/2)+C and logn

D. O(logn) and O(1)

Single/Linear Linked List

```
void fun(struct node **head_ref)
{
  struct node *temp = NULL;
  struct node *current = *head_ref;

  while (current != NULL)
  {
    temp = current→prev;
    current→prev = current→next;
    current→next = temp;
    current = current→prev;
  }

  if(temp != NULL )
    *head_ref = temp→prev;
}
```

What is the output of following function for start pointing to first node of following linked list? 1→2→3→4→5→6

A. 5 4 3 2 1 6.

B. 6 5 4 3 1 2

C. 2 1 4 3 6 5

D. 1 3 5 5 3 1

Single/Linear Linked List

```
void fun(struct node **head_ref)
{
  struct node *temp = NULL;
  struct node *current = *head_ref;

  while (current != NULL)
  {
    temp = current→prev;
    current→prev = current→next;
    current→next = temp;
    current = current→prev;
  }

  if(temp != NULL )
    *head_ref = temp→prev;
}
```

What is the output of following function for start pointing to first node of following linked list? 1→2→3→4→5→6

A. 5 4 3 2 1 6.

B. 6 5 4 3 1 2

C. 2 1 4 3 6 5

D. 1 3 5 5 3 1

Single/Linear Linked List

```
int f(struct node *p)
{
   return((p==NULL)||(p→next==NULL) || ((p→data<=p→next→data) && f(p→next)));
}
```

The below function f() on given linkedlist p always return 1, the linkedlist p should contain

A. 0 element

B. 1 element

C. Ascending order

D.Descending order

Single/Linear Linked List

```
int f(struct node *p)
{
   return((p==NULL)||(p→next==NULL) || ((p→data<=p→next→data) && f(p→next)));
}
```

The below function f() on given linkedlist p always return 1, the linkedlist p should contain

A. 0 element

B. 1 element

[✓ **C.** Ascending order]

D.Descending order

Single/Linear Linked List

```
int f(struct node *p)
{
        return((p==NULL)||(p→next==NULL)||((p→data<p→next→data)&&f(p→next)));
}
```

The below function f() on given linkedlist p always return 1, the linkedlist p should contain

A. 0 element

B. 1 element

C. Strictly Ascending order

D.Descending order

Single/Linear Linked List

```
int f(struct node *p)
{
      return((p==NULL)||(p->next==NULL)||((p->data<p->next->data)&&f(p->next)));
}
```

The below function f() on given linkedlist p always return 1, the linkedlist p should contain

A. 0 element

B. 1 element

[✓ **C.** Strictly Ascending order]

D.Descending order

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

C. Find starting point of loop

D.None of these

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
```

What does following function perform-

[✓ **A.** Detect loop]

B. Find loop length

C. Find starting point of loop

D.None of these

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
fast=first;
        while(slow!=fast)
        {
                slow=slow→next;
                fast=fast→next;
        }
        return slow;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

C. Find starting point of loop

D.None of these

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
fast=first;
        while(slow!=fast)
        {
                slow=slow→next;
                fast=fast→next;
        }
        return slow;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

[✓ **C.** Find starting point of loop]

D.None of these

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        int count;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
fast=first;
        while(slow!=fast)
        {
                slow=slow→next;
                fast=fast→next;
                count++
        }
        return slow;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

C. Find starting point of loop

D. None of these

Single/Linear Linked List

```
bool detectcycle(struct node *first)
{
        struct node *fast,*slow;
        int count;
        fast=slow=first;
        while(fast!=NULL && fast→next!=NULL)
        {
                slow=slow→next;
                fast=fast→next→next;
                if(slow==fast)
                        return true;
        }
        return false;
}
fast=first;
        while(slow!=fast)
        {
                slow=slow→next;
                fast=fast→next;
                count++
        }
        return slow;
}
```

What does following function perform-

A. Detect loop

[✓ **B.** Find loop length]

C. Find starting point of loop

D.None of these

Single/Linear Linked List

```
struct node* reverse(struct node *first)
{
        struct node *a,*b=NULL;
        while(first!=NULL)
        {
                a=first→next;
                first→next=b;
                b=first;
                first=a;
        }
        return b;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

C. Find starting point of loop

D.Reverse the list

Single/Linear Linked List

```
struct node* reverse(struct node *first)
{
        struct node *a,*b=NULL;
        while(first!=NULL)
        {
                a=first→next;
                first→next=b;
                b=first;
                first=a;
        }
        return b;
}
```

What does following function perform-

A. Detect loop

B. Find loop length

C. Find starting point of loop

[✓ **D.** Reverse the list]

Single/Linear Linked List

The following C function modifies the list by moving the last element to the front of the linklist and returns the linklist. Some part of the code is left blank. Fill the blank space-

```
typedef struct node
{
        int value;
        struct node *next;
}Node;

Node *move_to_front(Node *head)
{
        Node *p, *q;
        if ((head == NULL: || (head→next == NULL))
                return head;
        q = NULL; p = head;
        while (p→ next → next !=NULL)
        {
                q = p;
                p = p→next;
        }
        -------
        --------
        --------
        return head;
}
```

Single/Linear Linked List

The following C function modifies the list by moving the last element to the front of the linklist and returns the linklist. Some part of the code is left blank. Fill the blank space-

```
typedef struct node
{
        int value;
        struct node *next;
}Node;

Node *move_to_front(Node *head)
{
        Node *p, *q;
        if ((head == NULL: || (head→next == NULL))
                return head;
        q = NULL; p = head;
        while (p→ next → next !=NULL)
        {
                q = p;
                p = p→next;
        }
        q→next=NULL;
        p→next=head;
        head=p;
        return head;
}
```

Single/Linear Linked List

The rearrange() is called with the linklist containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the linklist after the rearrange () completes execution?

```
struct node
{
        int value;
        struct node *next;
};
void rearrange(struct node *list)
{
        struct node *p, * q;
        int temp;
        if ((!list) || !list→next)
                return;
        p = list;
        q = list→next;
        while(q)
        {
                temp = p→value;
                p→value = q→value;
                q→value = temp;
                p = q→next;
                q = p?p→next:0;
        }
}
```

Single/Linear Linked List

The rearrange() is called with the linklist containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the linklist after the rearrange() completes execution?

```
struct node
{
        int value;
        struct node *next;                              output-2,1,4,3,6,5,7
};
void rearrange(struct node *list)
{
        struct node *p, * q;
        int temp;
        if ((!list) || !list→next)
                return;
        p = list;
        q = list→next;
        while(q)
        {
                temp = p→value;
                p→value = q→value;
                q→value = temp;
                p = q→next;
                q = p?p→next:0;
        }
}
```

Single/Linear Linked List

In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is (GATE CS 2002)

A. log2n

B. n/2

C. log2n-1

D. n

Single/Linear Linked List

In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is (GATE CS 2002)

A. log2n

B. n/2

C. log2n-1

D. n

Single/Linear Linked List

Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest? (GATE CS 2004)

A. union

B. intersection, membership

C. membership, cardinality

D. Union ,intersection

Single/Linear Linked List

Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest? (GATE CS 2004)

A. union

B. intersection, membership

C. membership, cardinality

D. Union ,intersection

Single/Linear Linked List

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

int f(struct item *p)
{
return ((p == NULL) || (p→next == NULL) || (( P→data <= p→next→data) && f(p→next)));
}
```

For a given linked list p, the function f returns 1 if and only if (GATE CS 2003)

A. not all elements in the list have the same data value.

B. the elements in the list are sorted in non-decreasing order of data value

C. the elements in the list are sorted in non-increasing order of data value

D.none

Single/Linear Linked List

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

int f(struct item *p)
{
return ((p == NULL) || (p→next == NULL) || (( P→data <= p→next→data) && f(p→next)));
}
```

For a given linked list p, the function f returns 1 if and only if (GATE CS 2003)

A. not all elements in the list have the same data value.

B. the elements in the list are sorted in non-decreasing order of data value

C. the elements in the list are sorted in non-increasing order of data value

D.none

Single/Linear Linked List

AMAZON SUBJECTIVE QUESTION

Given a singly linked list of size **N** containing only English Alphabets. Your task is to complete the function **arrangeC&V()**, that arranges the consonants and vowel nodes of the list it in such a way that all the vowels nodes come before the consonants while maintaining the **order of their arrival**.

Input:
The function takes a single argument as input, the reference pointer to the **head** of the linked list. There will be **T** test cases and for each test case the function will be called separately.

Output:
For each test case output a single line containing space separated elements of the list.
User Task:
The task is to complete the function **arrange**() which should arrange the vowels and consonants as required.
Constraints:
1 <= T <= 100
1 <= N <= 100

Single/Linear Linked List

Example:

Input:

```
2
6
a e g h i m
3
q r t
```

Output:

```
a e i g h m
q r t
```

Explanation:

Testcase 1: Vowels like a, e and i are in the front, and consonants like g, h and m are at the end of the list.

Single/Linear Linked List

AMAZON INTERVIEW QUESTION

Given two numbers represented by two linked lists of size **N** and **M**. The task is to return a sum list. The sum list is a linked list representation of the addition of two input numbers.

Input: 9→9→3→4→5 and 8→9→1 (represent 99345 and 891)

Output: 1→0→0→2→3→6

Single/Linear Linked List

AMAZON INTERVIEW QUESTION

Delete nth node from end of a linked list in a single scan O(n) time

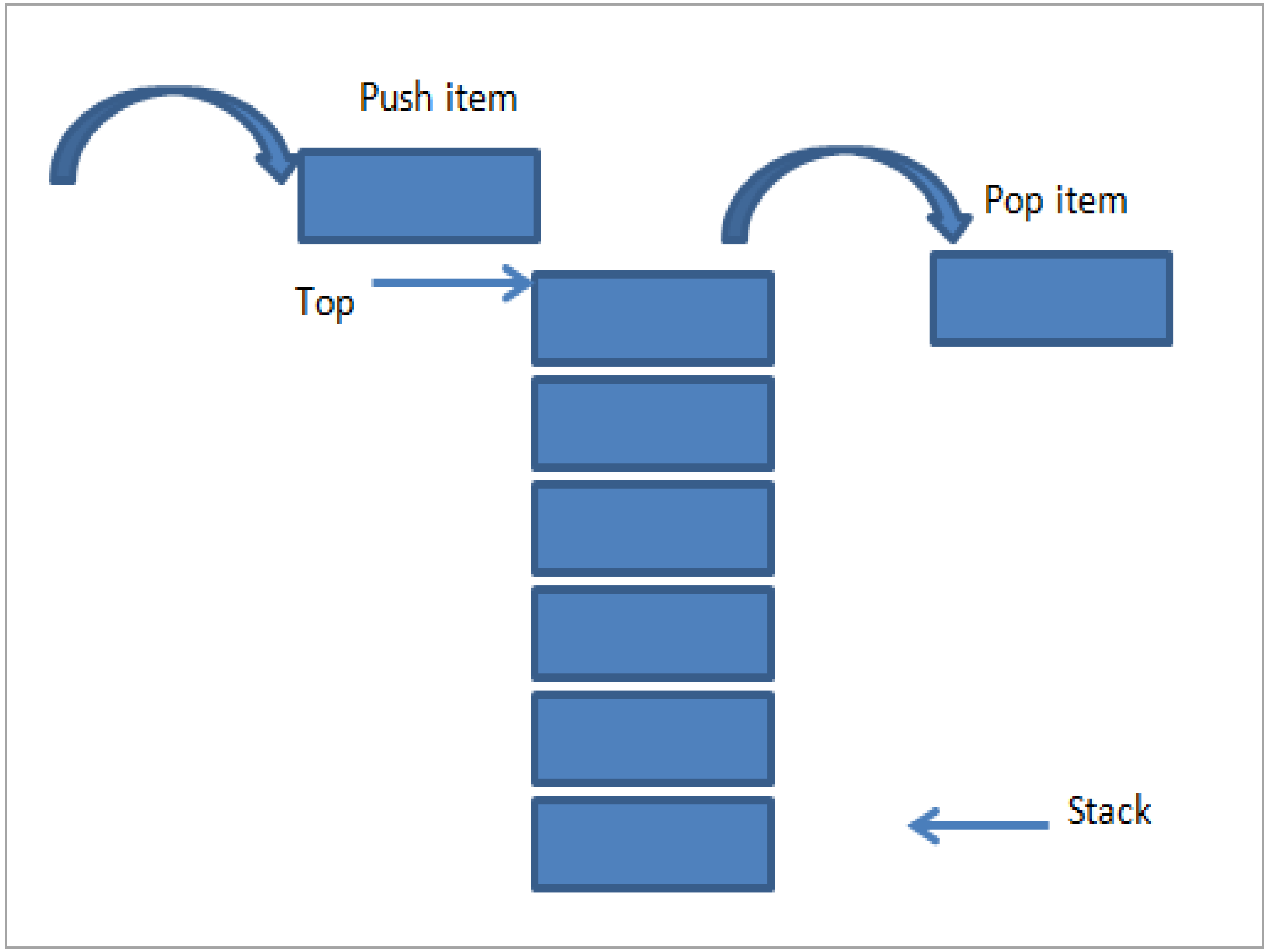
Single/Linear Linked List

AMAZON INTERVIEW QUESTION

Delete nth node from end of a linked list in a single scan OR O(n) time

Single/Linear Linked List

Stack using Linked List

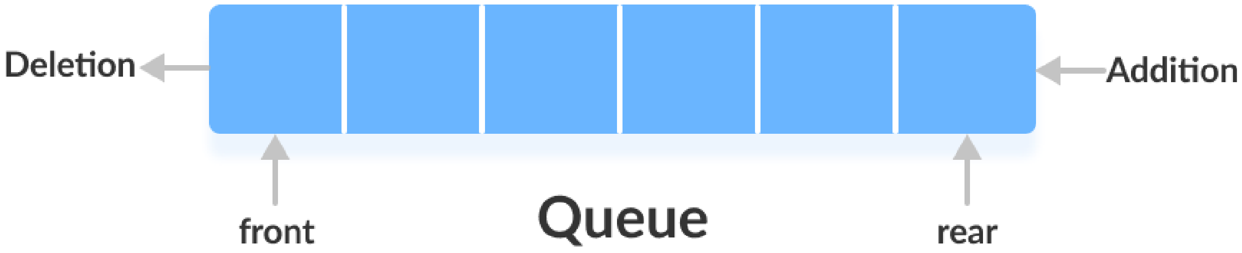


Single/Linear Linked List

Stack using Linked List

Single/Linear Linked List

Queue Using Linked List

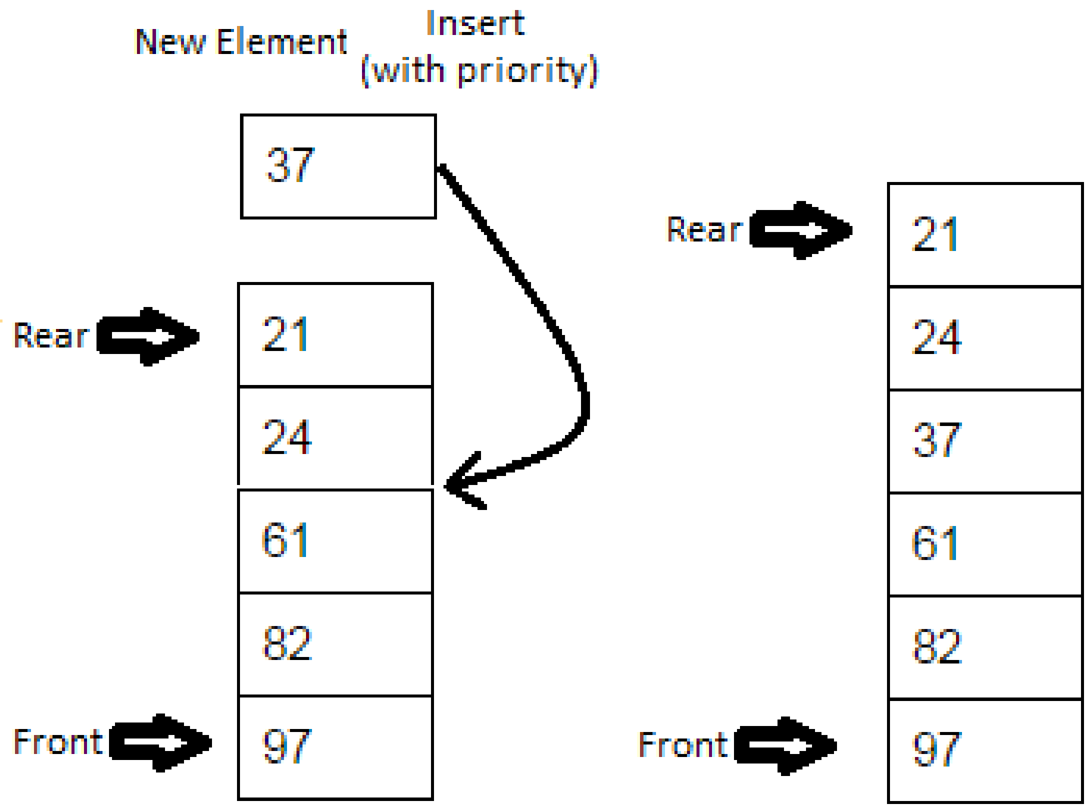


Single/Linear Linked List

Queue Using Linked List

Single/Linear Linked List

Priority Queue



Single/Linear Linked List

Priority Queue