DATA STRUCTURES AND ALGORITHMS

INDEX

NAME. : Smit Sekhadia Std. : Div. :

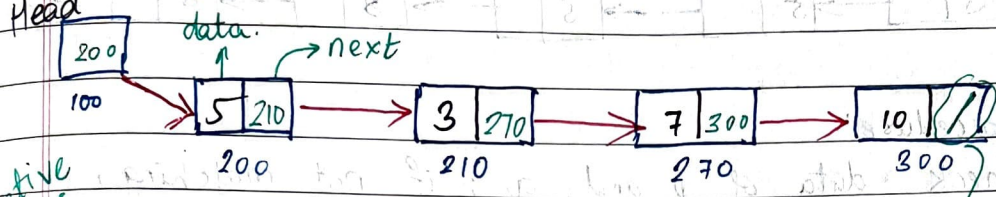
Roll No. : School / College. :

Sr. No	TITLE	Date	Teacher's Sign / Remark
01	LINKED LIST.		

Page No.
Date

LINKED LIST

→ Counting Nodes in LINKED LIST.



Last Node next = null.

Iterative

```
int count (struct Node *p)
 int c = 0
 while (p != 0)
  {
     c++;
     p = p->next;
  }
 return c;
}
```

Time: O(n); n = number of node
Space: O(1); one pointer & one int var

Recursive

```
int count (struct Node *p)
int {
    if (p == 0)
        return 0;
    else
        return count (p->next) + 1;
    }
```

① if(p) ② if (p != NULL)

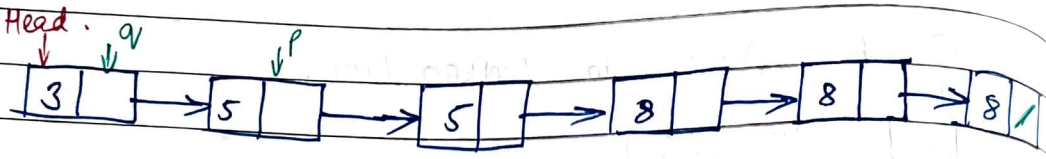
Time: O(n)
Space = O(n)

C(200) = 5
C(210) +1 = 3+1 = 4
C(270) +1 = 2+1 = 3
C(300) +1 = 1+1 = 2
C(0) +1 = 0+1 = 1
to

In recursive, addition of +1 is done at returning time and not at calling time.

DSA 450.

→ Removes duplicated elements from a Sorted Linked List



Procedure:

Check data of p and q; if not matching, move q upon p and p to next node. if data matches; make q point on p's next, delete p and move p to next node.

Code:

```
Node *q = head                    TIME : O(n)
Node * p = head->next.            Space : 2pointers .:. O(1)
while (p != NULL)
 {
    if (p->data != q->data)
    {
        q = p;          //move q to p
        p = p->next;    // move p to next node.
    }
    else                //if data is equal
    {
      q->next = p->next;  //make q to point next of p
      delete p;           //delete duplicated element from heap
      p = q->next
    }
 }
```

Deletes the node from heap memory.

DSA 450.

Reversing Linked List using Iterative & Recursive methods.

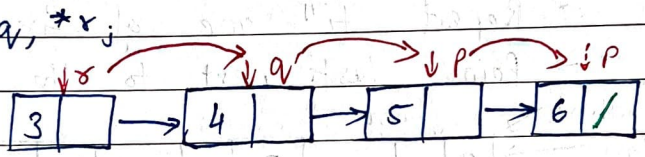
→ Reversing using Sliding Pointers

Procedure

→ Have 3 ptrs one before other in a line p, q, r.
→ make p = head & q = r = NULL.
→ move r to q; q to p and p to next node till p becomes null.
→ all the three pointers appears to slide on list
→ when p is null; q will be on last node.
→ make the node on q as head (as we have to reverse)
→ point the next of q on r after every increment in node.

Code: Iterative:

```
Node *p, *q, *r;
p = head;
q = r = NULL;
while (p != NVLL)
{
    r = q
    q = p
    p = p->next
    q->next = r
}
head = q;
}
```



TIME: O(n) as traversing through list once.
SPACE: 3 ptrs are required ∴ O(1)

Recursive:

```
void Reverse (Node *q, Node *p)
{
  if (p != NULL) {
    Reverse (p, p->next);
    p->next = q;
  }
  else {
    first = q;
  }
}
```

2 ptrs reqd p is ahead & q is tail ptr of p

Page No.
Date

→ Merging 2 Linked List:

Merging 2 sorted list into another sorted Linked list.

(In arrays 3rd array is reqd; in LL no need of 3rd list)

Procedure:

→ Need 4 ptrs → first, second, third, last. (first ptr of 1st list; 1st ptr of 2nd list; first ptr of 3rd list; last → helping ptr; last node of merged list)

→ Compare data of first node in 1st & 2nd List.

→ Bring last and third on the smaller data

→ Move first or second of that list to next node. (where 3rd & last are present that list)

→ Make

→ Again Compare which data is small

→ Point last's next to that node

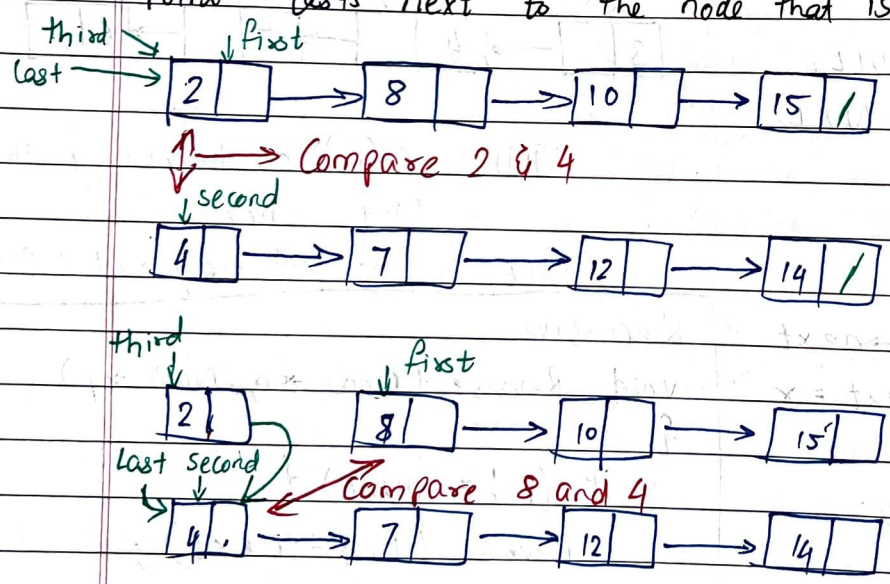
→ Bring last on that node.

→ Bring first/second to next node

→ Make last next as NULL.

→ Repeat till one of first or second is NULL.

→ Point last's next to the node that is not null.



Page No.
Date

Code:

```
if ( first -> data < second -> data)
{
   third = last = first ;
   first = first -> next ;
   last -> next = NULL ;
}
else {
  third = last = second
  second = second -> next
  last -> next = null ;
}
while ( first != NULL && second != NULL)
{   if ( first -> data < second -> data )
     {
         last -> next = first;
         last = first;
         first = first -> next ;
         last -> next = NULL;
     }else {
         last -> next = second;
         last = ~~a~~ second;
         second = second -> next ;
         last -> next = NULL;
     }
     if ( first != NULL)
      {
         Last -> next = first;
      }
     else
      { last -> next = second;
      }
}
```

TIME : O(m+n)
where m = no of nodes in LL1
& n = no of nodes in LL2.

SPACE : O(max(m, n))

O(1)
if you dont make new LL for ans

if you make a new 3rd LL for ans

DSA 450 →

Check for Loop in Linked List

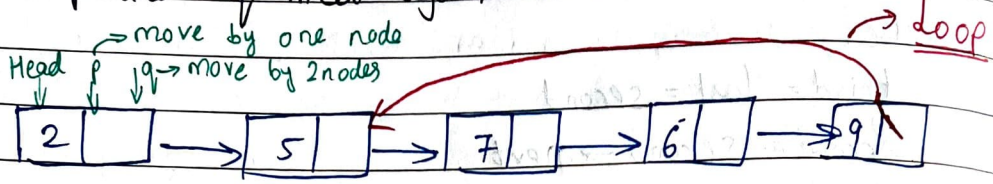
Procedure: (Slow and fast pointer method)

→ Have two ptrs p & q

→ move p by one node every time.

→ move q by two nodes every time

→ if p and q meet again it has a loop.



Code:

```
bool isLoop (Node *head) {
    Node *p, *q;
    p = q = head;

    do {
        p = p->next;
        q = q->next;
        q = q != NULL ? q->next : NULL; // again increasing q
    } while (p && q && p != q);

    if (p == q)
    return true;
    else
      return false;
}
```

TIME: O(n)

SPACE: 2 ptrs reqd ∴ O(1)

only if q is not null after first increament.

Page No.
Date

CSA 450 ⭐⭐⭐

Removing Loop from Linked List

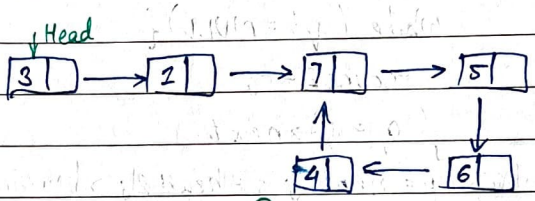
Procedure:

→ Check if Loop is present
→ bring p on head.
→ q is currently in the node which is in loop as we checked loop is present or not.
→ move p and q ahead by one node till both's next reach the same node.
→ This node is the starting point of the loop.
→ Make q's next NULL to remove the loop.

Code:

```
Node *p, *q;
p = q = head
while (p && q && q->next != NULL)
{
    p = p->next;
    q = q->next->next;
    if (p == q)
    {
        break;
    }
}
if (p == q)
{ p = head;
  while(p->next != q->next)
  { p = p->next;
    q = q->next;
  }
  q->next = NULL;
}
else if (p = head)
{ while (q->next != p)
  { q = q->next;
  }
  q->next = NULL;
}
```

TIME : O(n)
SPACE : O(1)



p	3	2	7	5	6	4	7	
q	3	7	6	7	6			

→ p = q ∴ loop present and now bring p to Head.

Now check p's next & q's next are equal or not; they will be equal on node 7. Now break the loop

→ What if you get p=q on first node. now this is a circular LL whose last node is pointing on first node.

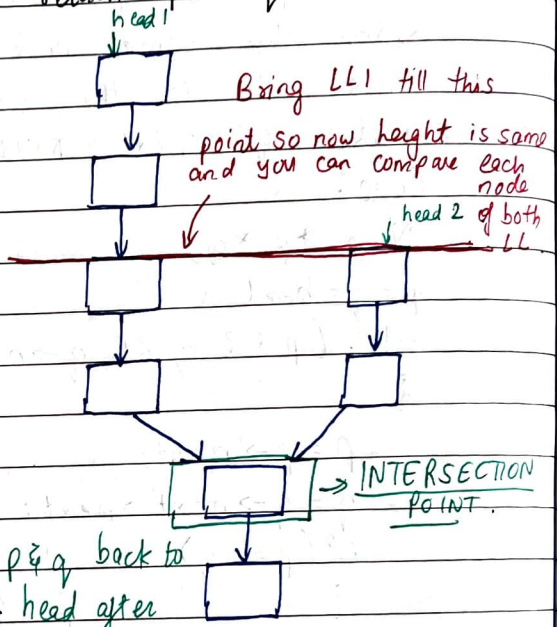
So you will get last node; now break the loop

Special case to be forgotten

GSA 450. → Intersection Point in Y Shaped Linked list.

Procedure:

→ Take 2 ptrs p and q and point them on heads of LL1 & LL2.
→ Count nodes of both LL.
→ store the diff of no of nodes in a var d
→ Traverse the first d nodes of longer LL
→ Now both the LL are on same height.
→ Traverse till p==q and return p's or q's data.



Code:

```
Node *p, *q;
p = head1; q = head2;
while (p != NULL)
{ count1++;
  p = p->next;
}
while (q != NULL){
  count2++;
} q = q->next
```

gives absolute value of count1-count2 ← `int d = abs(count1 - count2);`

p = head1; q = head2; → bringing p & q back to head after counting number of nodes

```
if (count1 > count2) {
  for (int i=0; i<d; i++){
    p = p->next;
  }
}
else if (count2 > count1){
  for (int i=0; i<d; i++) {
    q = q->next;
  }
}
while (p != q)
{ p = p->next;
  q = q->next;
} return p->data;
}
```

TIME: O(m+n)
where m are nodes of LL1 & n are nodes of LL2.
SPACE: O(1) as 2 extra ptrs reqd.

DSA 450 =>

Reverse Delete nodes having greater value on right.

→ Procedure:

→ Reverse the LL

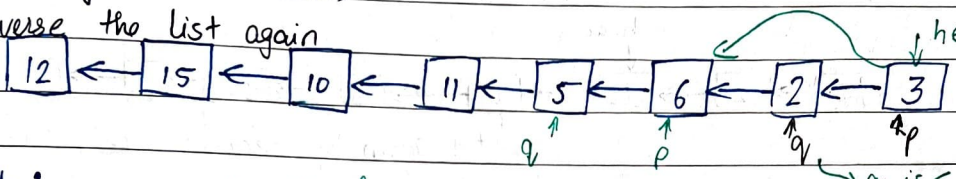
→ Create a var max and store head's data.

→ Take two ptrs p & q.

→ Compare each data with max if it is bigger update max to that data and bring p & q forward

→ If it is smaller than make p on q and take q ahead (unlinking that node)

→ Reverse the list again



Code:

```
Node *p, *q;
head = Reverse(head);
p = q = head;
int max = p->data;
while (q != NULL)
{
   if (q->data < max)
   {
      p->next = q->next;
      q = p->next;
   }
   else {
      max = q->data;
      p = q;
      q = q->next;
   }
}
q = Reverse(head);
return q;
}
```

See the function to reverse LL behind.

Look at 12. on right side it has 15 > 12 ∴ remove 12.

Look at 15. on right all elements are smaller than 15 ∴ Keep it

OUTPUT: 15 11 6 3

max = 3 ∴ unlink it

max = 6, as 6 is > than 3

Here we reverse the LL as if we don't; The Time becomes $O(n^2)$ as you be on each node and then check if the right is smaller or bigger.

TIME: O(n)

SPACE: O(1)

DSA 450 → Reverse Linked List in groups of given size

★★★ VIMP

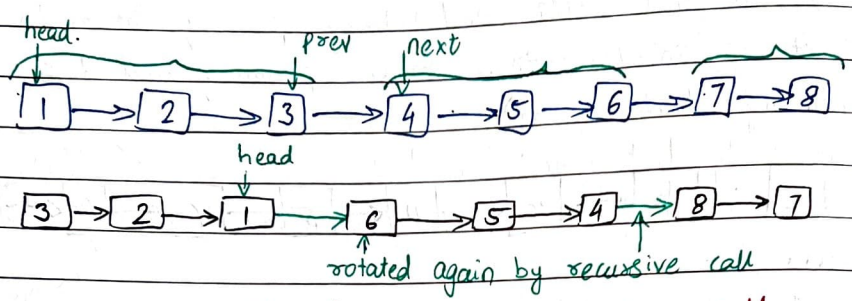
→ Procedure:

→ Write normal code for reversing a linked list

→ With one extra condition of $c < k$ where c is 0 and we increment it in every while loop.

→ Point head's next to recursive call

K = 3



∴ Point head's next to recursive call

Code: Node * reverse (struct Node * head, int k) {

Node * cur, prev, nex;

cur = head;

prev = NULL;

int c = 0

extra condition

Normal reversing

while [cur != NULL && (c < k)] {

nex = cur → next;

cur → next = prev;

prev = cur;

cur = nex;

c++;

}

TIME: O(n)

SPACE: O(1)

for reversing in groups & linking them

if (nex != NULL)

{

head → next = reverse (nex, k)

}

return prev;

}

DSA 450

→ Remove duplicate elements from Unsorted LL.

Method 1: Using Maps.

→ Create a map of type int by map<int, int> mp;

→ Take 2 ptrs prev and curr

→ Traverse through entire List.

→ store each data of List as key in map and make its value as one

→ if we again find that value != 1 then we have to remove it

Method 2: Using unordered sets.

→ Create an unordered set s by unordered_set<int> s;

→ take two pointers prev and temp at head and heads next

→ insert heads data in set.

→ traverse the LL and use count function to check if data is already present or not.

Code:

USING MAPS

```
map <int, int> mp;
Node * prev = head;
Node* cur = head -> next;
mp[prev -> data] = 1;
while (cur) {
  if (mp[cur -> data] > 0){
    prev -> next = cur -> next;
    cur = prev -> next;
  } else {
    mp[cur -> data] = 1
    prev = cur;
    cur = cur -> next;
  }
}
return head;
}
```

TIME: O(n)

SPACE: O(n)

USING UNORDERED SETS

```
if (head -> next = NULL) {return head}
unordered_set <int> s;
s.insert (head -> data);
Node * prev = head;
Node * temp = head -> next;
while (temp != NULL) {
  if (s.count (temp -> data) == 0){
    s.insert (temp -> data);
    prev = prev -> next;
    temp = temp -> next;
  } else {
    prev -> next = temp -> next;
    temp = temp -> next;
  }
return head;
}
```

TIME = O(n)

SPACE = O(n)

DSA 450

→ Add 1 to a number represented as Linked List.

Procedure:

→ Reverse the Linked List.

→ Check if the last node is 9 in reversed LL.

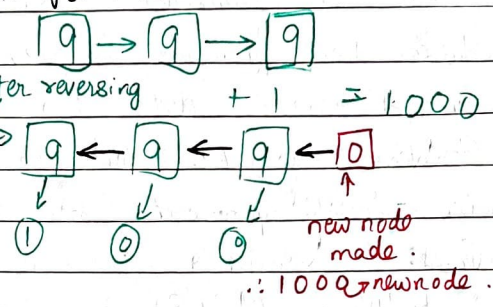
→ if it is 9 then make that as 10 and make a newnode at the start o [999 wala case]

→ Check if p's data is 9

→ if it is 9 then make that as 0 and move ahead

→ else add 1 to the data if it is not 9.

→ Code: Reverse the LL again to get final ans




```
Node * addOne (Node *head)
{
  head = reverse (head);
  Node * p = head;
  while (p != NULL)
  {                          → 999 case
    if (p->data == 9 && p->next == NULL)
    { p->data = 1;
Making New Node → Node* temp = new Node (0);
as a digit increase
if we add 1 to 999
      temp->next = head;   } → Bringing Head at
      head = temp;             the new node created.
      p = p->next;
    }                        → 399 case
    else if (p->data == 9)
    { p->data = 0;
      p = p->next;
    }
    else {                   → 344 case
      p->data += 1;
      p = p->next;
      break;
    }
  head = reverse (head);
} return head;
```

TIME: O(n)

SPACE: O(1)

DSA 450

→ Add two No's represented by Linked List.

Procedure:

→ Reverse both Linked Lists

→ Make a sum = carry + first's data + second's data.

→ & carry as if sum is >= 10 then carry is 1 else 0

→ Sum = sum % 10 as is addition of 2 nos is 18 we wont & as Sum & 1 as carry.

→ Make a new node with value sum and make it a LL

→ if carry is >0 after while loop (99+1) case) then make a new node to get ans as (100 carry).

→ Code: reverse resultant LL again to get final ans.

```
Node addTwoLL (Node *first, Node *second){
  Node *res = NULL, *cur = NULL, *temp;
  int s = 0, c = 0;
  first = reverse(first)
  second = reverse(second)
  while(first != NULL || second != NULL){
  s = c + (first ? first->data : 0) + (second ? second->data : 0);
  c = (s >= 10 ? 1 : 0);
  s = s % 10;
  temp = new Node(s);
  if (res == NULL){
    res = temp;
  }else{
    cur->next = temp;
  }
  cur = temp;
  if(first) first = first->next;
  if (second) second = second->next;
  }
  if (c > 0){
    temp = new Node(c);
    cur->next = temp;
    cur = temp;
  }
  return res = reverse(res);
  return res;
}
```

↳ if second exist then add second's data else add 0.

↳ if sum is >= 10 then carry is 1 else 0

↳ Creating new Node for ans.

↳ & Adding first Node to res.

↳ Linking all nodes of result.

TIME: O(n)

SPACE: O(1)

1 carry left at end so make new node for that

99 + 1 = 100

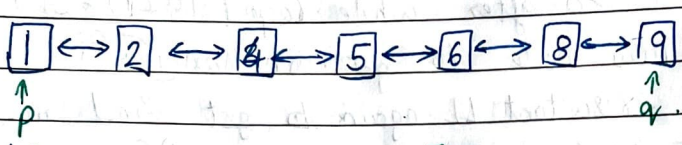
Page No.
Date

IMP ★★★ DSA 450

→ Find pairs with given sum in doubly linked List

Procedure:

→ Have 2 Ptrs pat start & q at end i.e last node of LL.
→ p's data + q's data = sum
→ if sum < x ; move p ahead by one node
→ if sum > x ; move q back by one node
→ if sum == x ; move p ahead & q back by one node.
→ if p & q crosses each other, stop.



Code: x = 7.

```
Node* p = head, *q = head;
while (q->next != NULL){
    q = q->next;
}
bool found = false;
while (p != q && q->next != p)
{
    if ((first->data + second->data) == x){
    { found = true;
      cout<< first->data << second->data;
      first = first->next;
      second = second->prev;
    }
    else{
      if ((first->data + second->data) < x){
        first = first->next;
      }else{
        second = second->prev;
      }
    }
}
if (found == false)
{ return 0;
```

This is only possible if DLL is sorted. if it was unsorted we needed to travel through LL n^2 times as we needed to check all the elements with each pair.

if data is found take p ahead & q back

★★★ TIME: O(n)
SPACE: O(1)

if sum < x ; take first ahead else take second back

Page No.
Date

→ Count triplets whose Sum is equal to X in Linked List.

METHOD - 1 (Native approach using 3 nested for loops) TIME: $O(n^3)$

METHOD - 2 (Hashing using unordered MAPS) TIME: $O(n^2)$ SPACE: $O(n)$

- Create hash table with (key, value) as (data, Node*) tuples.
- Traverse LL and store each node's data and its ptr pair in hash table
- Generate each possible pair of nodes.
- For each pair, Calculate sum of data of two nodes and check wether (x - psum) exists in the table.
- If it exist then two nodes and data in the table are a triplet

METHOD 3: (Optimized Method) (2 ptr approach) TIME: $O(n^2)$; SPACE: $O(1)$

- Make 3 ptrs first, current & last.
- For each current, traverse the LL once with first as head & last as last node.
- Check wether x-first→data + Last→data is equal to current→data.
- If it is equal; they form a triplet.; Continue till like prev sum (on left pg).

CODE for METHOD 2 (Hash Maps)

TIME: $O(n^2)$
SPACE: $O(n)$

```
Node* ptr, *ptr1, *ptr2;
int count = 0;
unordered_map <int, Node*> um;
for (ptr = head; ptr != NULL; ptr = ptr->next)
    um[ptr->data] = ptr;
for (ptr1 = head; ptr1 != NULL; ptr1 = ptr1->next)
    for (ptr2 = ptr1->next; ptr2 != NULL; ptr2 = ptr2->next)
        int p_sum = ptr1->data + ptr2->data;
        if (um.find(x - psum) != um.end() && um[x - psum] && um[x-psum] != ptr2)
        {
            count++;
        }
Count = count / 3; → As each triplet is counted 3 times.
return count;
}
```

Checking if x - psum is present in Map.

Page No.
Date

CODE for METHOD-3 TIME: $O(n^2)$; SPACE: $O(1)$

```
int CountTriplets (Node* head, int x)
{
    if (head == NULL)
    { return 0; }
    Node *current, *first, *last;
    int count = 0;
    last = head
    while (last != NULL)
    { last = last -> next; }
    for (current = head; current != NULL; current = current -> next)
    {
        first = current -> next.
        count = count + CountPairs (first, last, x - current -> data);
    }                    ↳ Calling function CountPairs.
    return count;
}
int CountPairs (Node * first, Node * second last, int value)
{
    int count = 0;
    while (first != NULL && second != NULL && first != second &&
                                    second -> next != first)
    {
        if ((first -> data + second -> data) == value)
        {
            count ++;
            first = first -> next;
            second = second -> prev;
        } ↳ if less than take first ahead.
        else if ((first -> data + second -> data) < value) {
            first = first -> next; }
        else { ↳ if more than take second behind.
            second = second -> prev; }
    }
    return count;
}
```

$a + b + c = x$

$a = (x - (b + c))$ ↳ value.

So if value = a;
a, b, c form triplet.

↳ go on till both the pointer crosseach other or becomes equal.