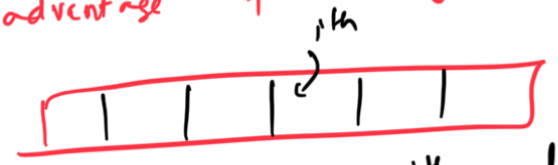
# Linked List -1

Biggest advantage of array ?

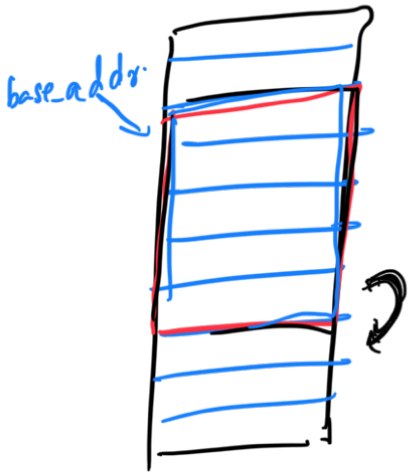
O(1) Random

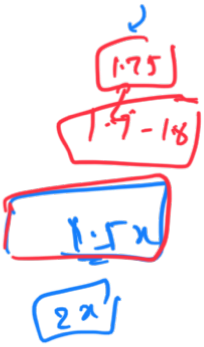



- Fetch the $i^{th}$ index Element => O(1)

addr(i) = base-addr + 4 * i

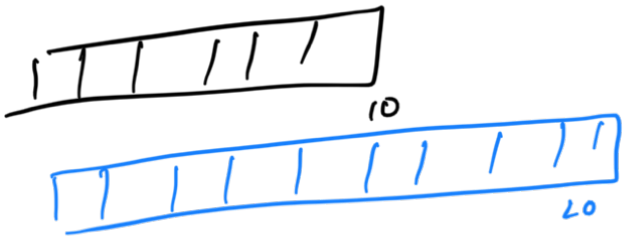
O(1)




Dynamic Array

Vector / ArrayList




Dynamic Array => Fixed Size array

1 element


=>


99 elements.

-> Unused space

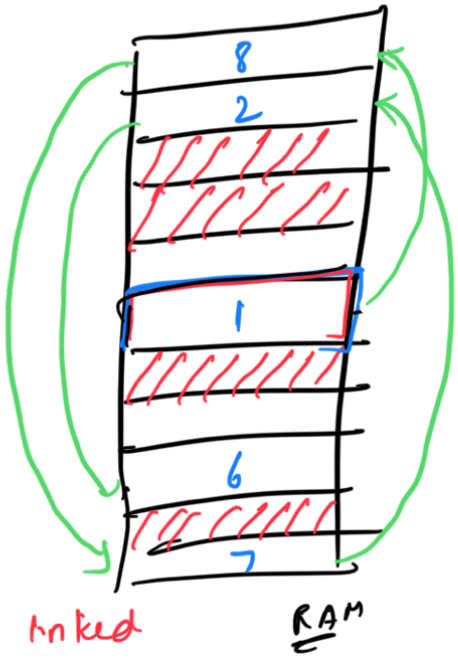
Linked List

If I have to store 100 elements, I will only allocate space for 100 elements

A = [ 1, 8, 7, 2, 6 ]

1 2 1 2 2

Stored in consecutive Memory Locations?




Every node stores the address/Reference to the next node.

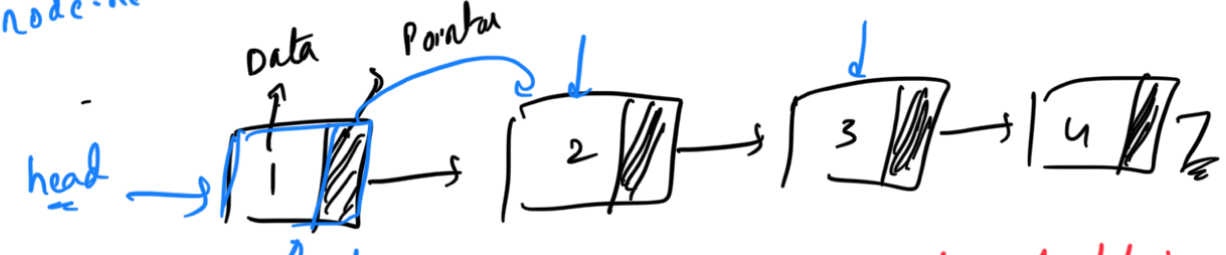
Linked List : Nodes are linked to each forming a list.

## Effective Space Utilization

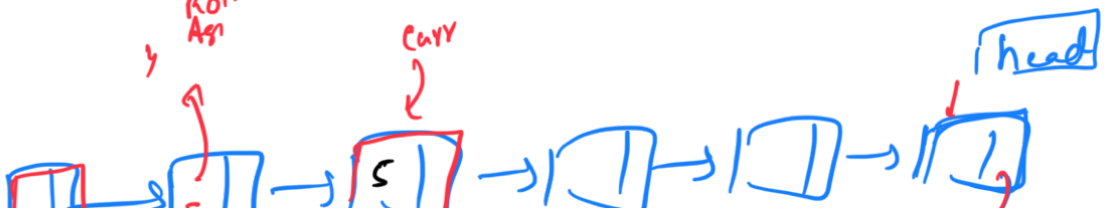
↳ Extra Space for pointers. as 4 bytes

node


node.data
node.next




Pointer to the first node of LL : head Pointer



NULL

Lastnode.next = NULL

if( head == NULL)
Empty Linked List

C++

```
struct Node {
    int data;
    float y;
    string s;
    Node * next;
};
```

} Data

// Pointer to next Node

Node obj = new Node( )

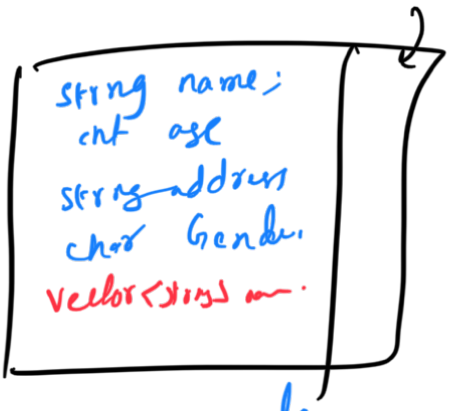
Java / Python

```
class Node {
    int data;
    Node next;
};
```

// Object of the same class

Reference of the object

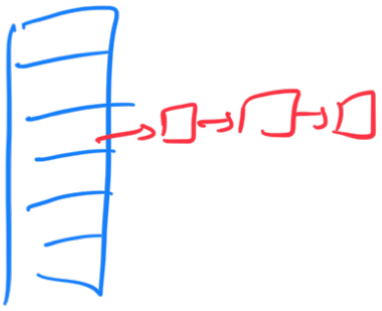
next Pointer.




Node

## Applications / Uses

1) LRU cache => Doubly LL
2) Stack & Queues using LL
3) Graphs to represent adjacency list
4) Hashing (chaining)



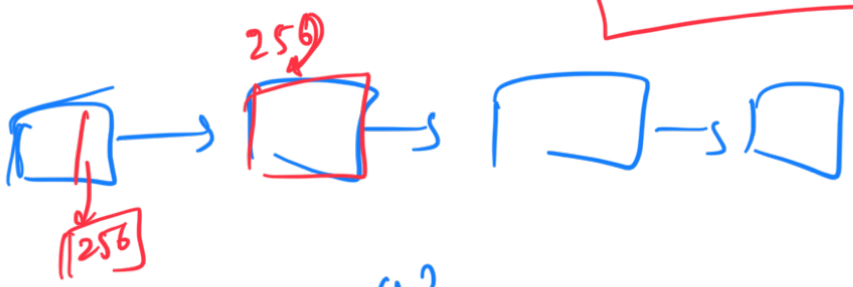
99% Edge cases

Stacks / Queues
LL

**Question:** Print a Linked List


3 1 5 6

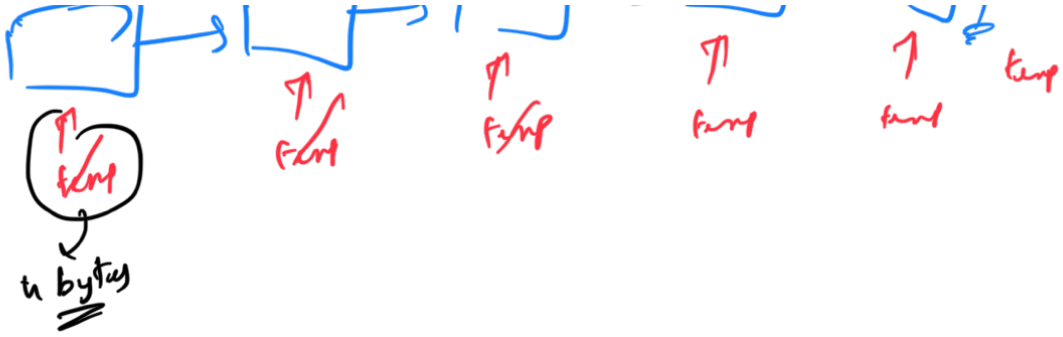
```
temp = head; , len = 0;
while (temp != NULL) {
    print(temp.data)
    temp = temp.next;
    len++;
}
return len
```




T.C: O(n), S.C: O(1)

**Question:** Length of LL

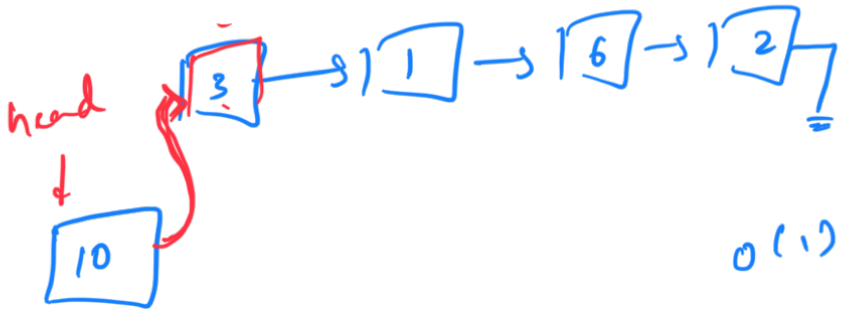
count = 1 2 3 4 5 = 5




## Operations :

### Insertion

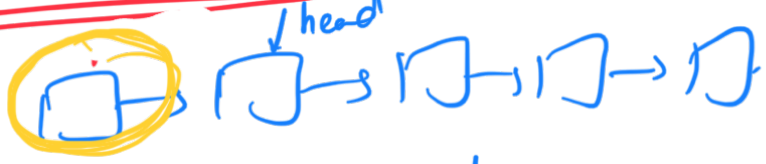
1) Insert at the start




O(1)

```
insertAtBeginning(head, data){
    newNode = new Node(data);
    newNode.next = head
    head = newNode
return head;
}
```

### Delete from the start




```
head = head.next;

if(head == NULL)
```

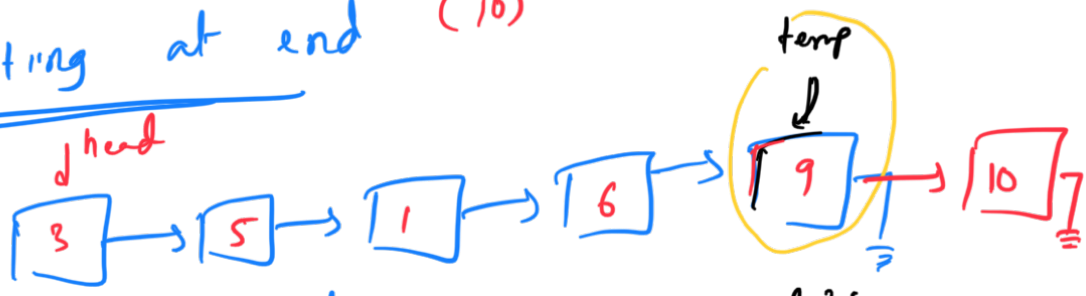
O(1)

ERROR

T.C:

temp = head;
head = head.next;
free(temp);

2) Inserting at end (10)




temp = head;
if (temp == NULL) { head = newNode; }
while (temp.next != NULL) {
temp = temp.next;

head = NULL

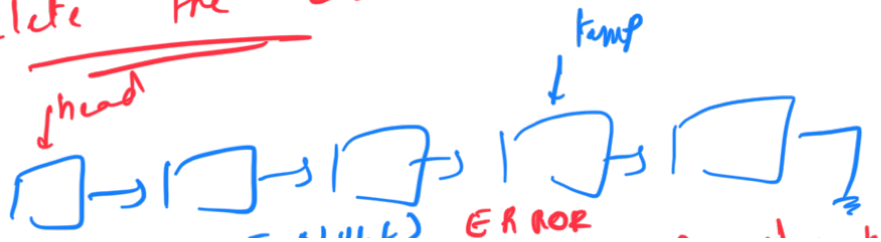
T.C: O(n)
S.C: O(1)

}
temp.next = newNode;

newNode.next = NULL

Constructor (int data) {
newNode = new Node();
newNode.data = data;
newNode.next = NULL;
return newNode;
}

Delete the Last Node




head = NULL;

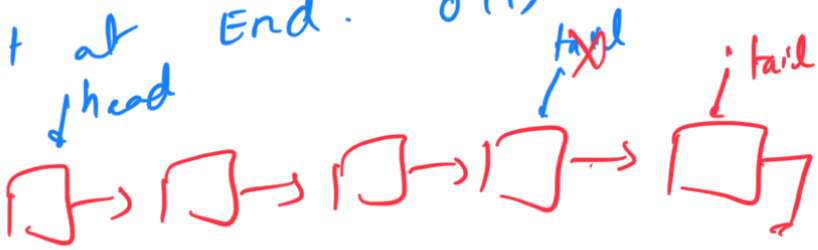
if(temp == NULL) { free(head); head = NULL; }
if(head.next == NULL) {

```
while (temp.next.next != NULL) {
    temp = temp.next;
}
free(temp.next);
temp.next = NULL;
```

T.C: O(n)
S.C: O(1)


Insert at End : O(1)

S.C: O(1)
T.C: O(1)



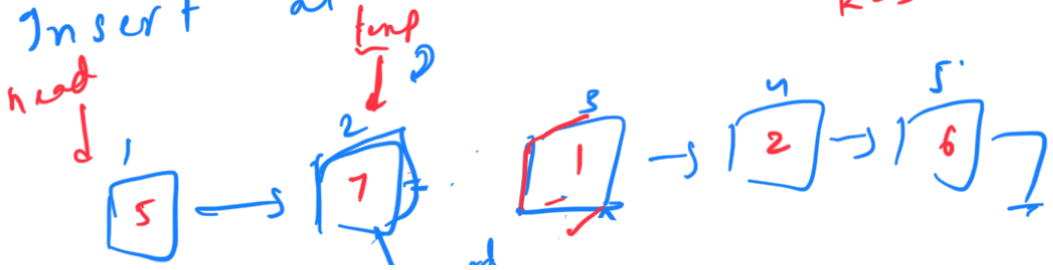

```
insertAtEnd ( ) {
    if (head == NULL) {
        head = tail = newNode;
    }
    tail.next = newNode;
    tail = newNode;
}
```

(tail = tail.next)

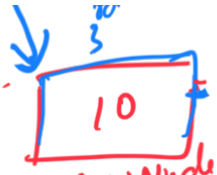
T.C: O(1)
S.C: O(1)


3) Insert at $k^{th}$ Position (val 10)

k = 3



temp.next = newNode




newNode.next = temp.next

temp.next = newNode;

Insert at $K^{th}$ Position => Iterate till $(K-1)^{th}$ node.

if(K = 1)

temp = NULL

$0^{th}$ Node

## Edge Cases:

1) Empty LL
2) Length 1 LL
3) Length 2/3 LL
4) Normal Test Cases
5) Problem Specific Problem

(K == 1

Length = 10

K > 11

```
if( K == 1) {
    newnode.next = head;
    head = newNode;
```

}

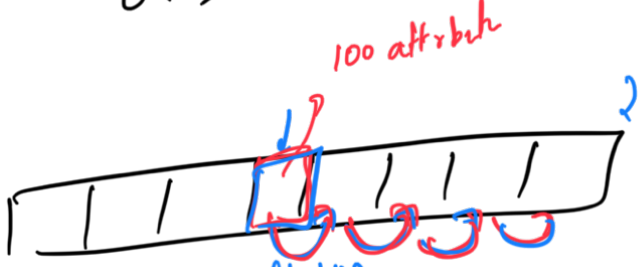
| Insertion | Begin | End | Middle |
|---|---|---|---|
| Array | O(n) | O(1) | O(n) |
| Linked List | O(1) | O(1) | O(n) |

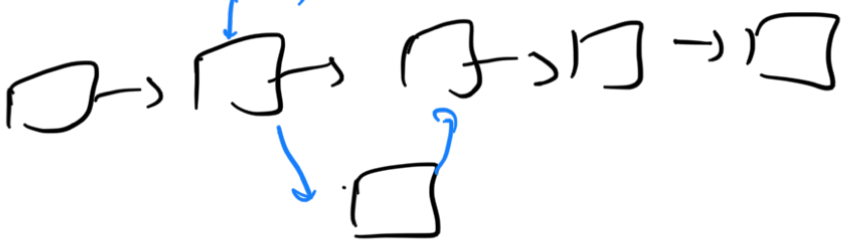
Linked List:

Random Access

vector < Node >

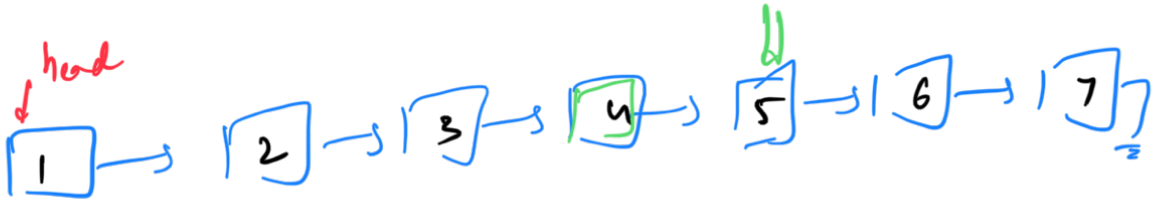
100 attribute



Question: Find the $k^{th}$ Node from the last

$k = 3$




$N = 7$

$K^{th}$ Node from Last $\Rightarrow$ $(N-K+1)^{th}$ node from first

$N - K = 4$

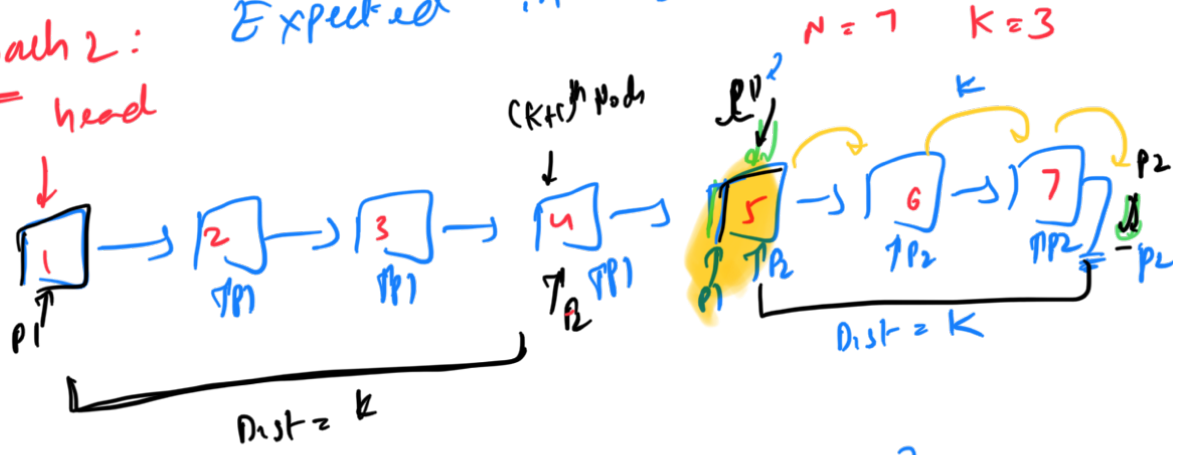
Approach 1

1) Compute Length

2) Iterate till the $(N-K+1)^{th}$ node

T.C: O(N)
S.C: O(1)

Approach 2: Expected in Interviews




p1 = head ✓
p2 = (K+1)th Node from head ?

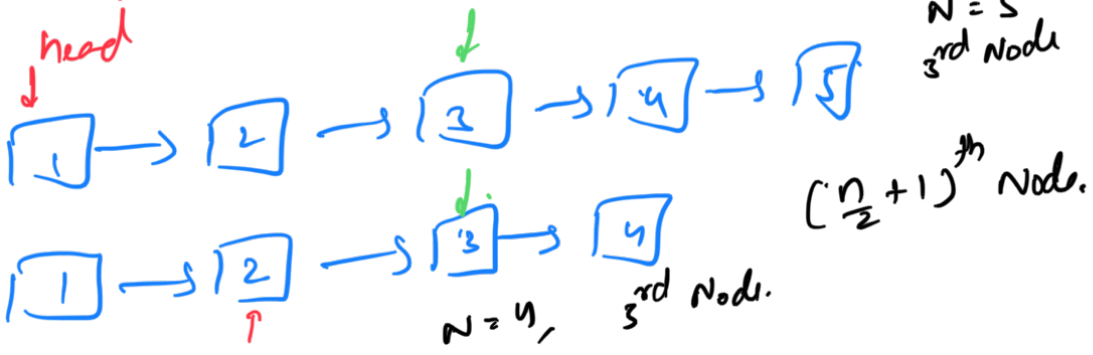
1 iteration ← p2 = NULL ?

i iterations [ while (p2 != NULL) {
p1 = p1.next;
p2 = p2.next;
}

return p1;

T.C: O(K) + O(n)
O(n) + O(n)
= O(n)

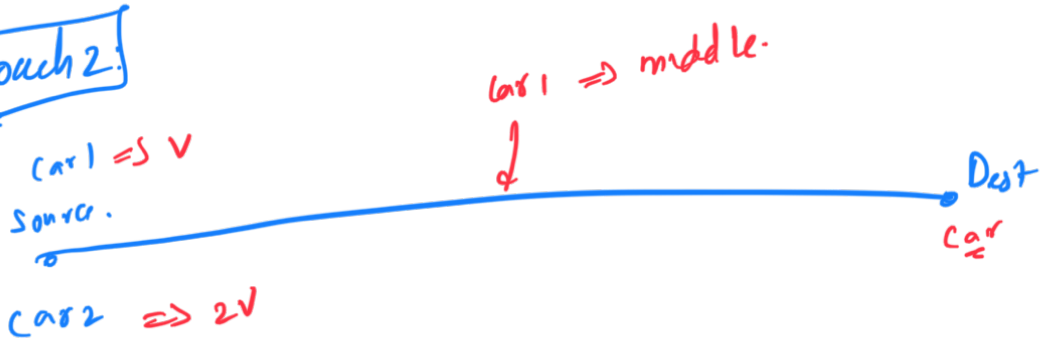
Question: Find the middle node of LL




Approach 1:
Find length of LL = n } 2 iterations
Find the (n/2 + 1)th node

T.C : O(n)

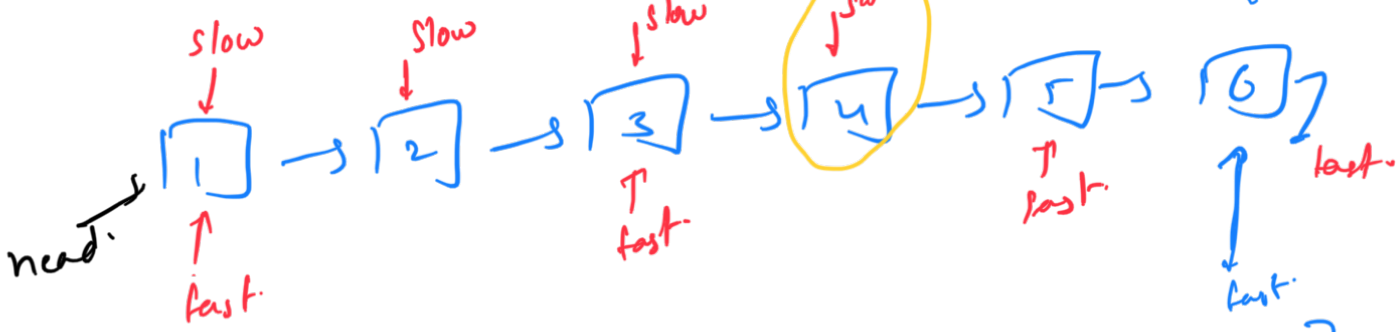
Approach 2:



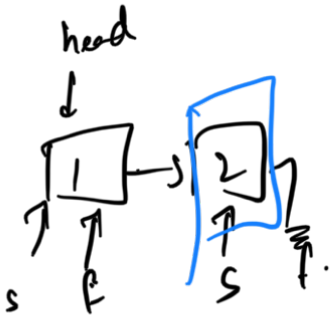

2 Pointers

1) slow => V speed

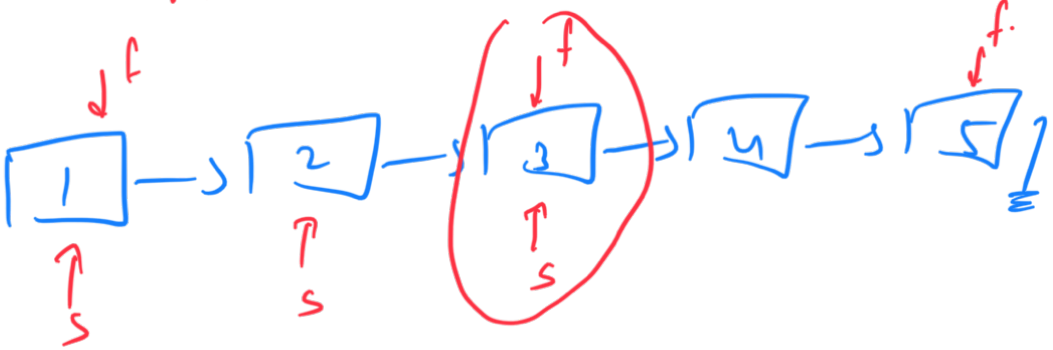
2) fast => 2V speed.




```
slow = head
fast = head;
while (fast!=NULL && fast.next != NULL)
    slow = slow.next;
    fast = fast.next.next;
```




return slow;



T.C: O(n)

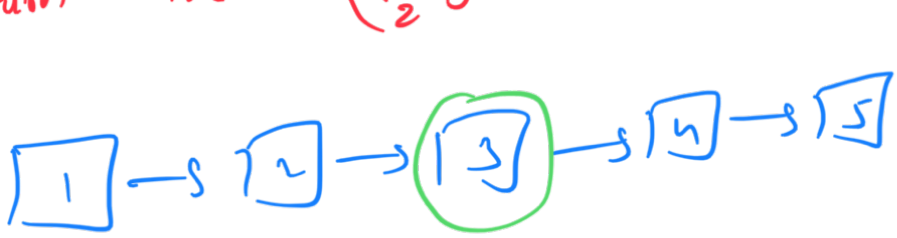
S.C: O(1)
only 1 iteration.

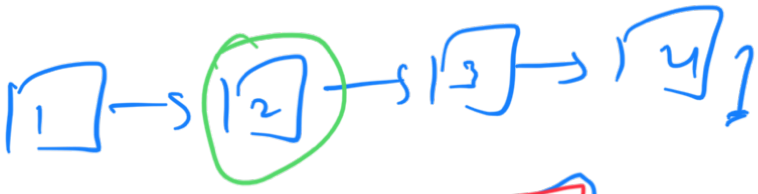
Question:

slow = head
fast = head.next;

{ HW }

Return the $\left(\frac{n+1}{2}\right)^{th}$ Node



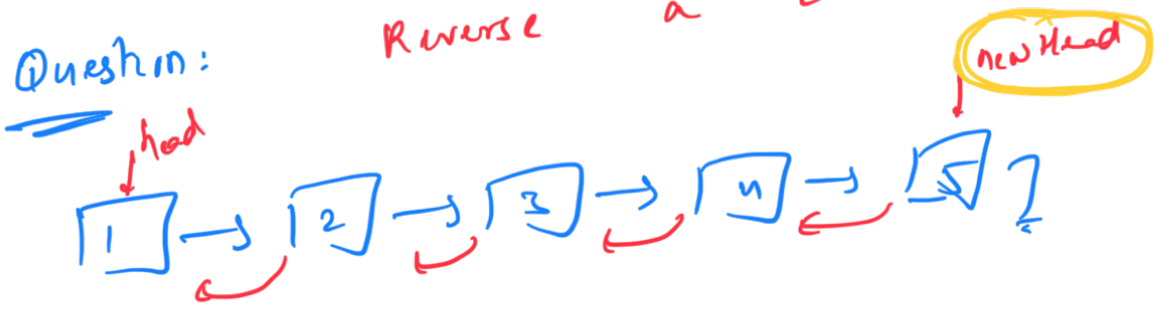




fast.next.next.next

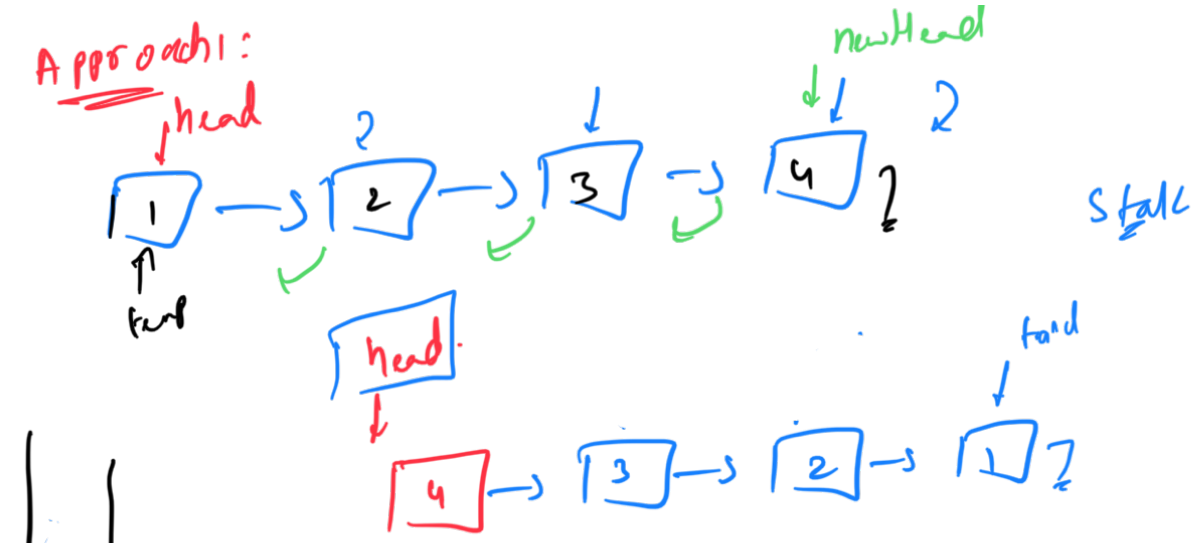
-> fast != NULL
-> fast.next != NULL
-> fast.next.next != NULL

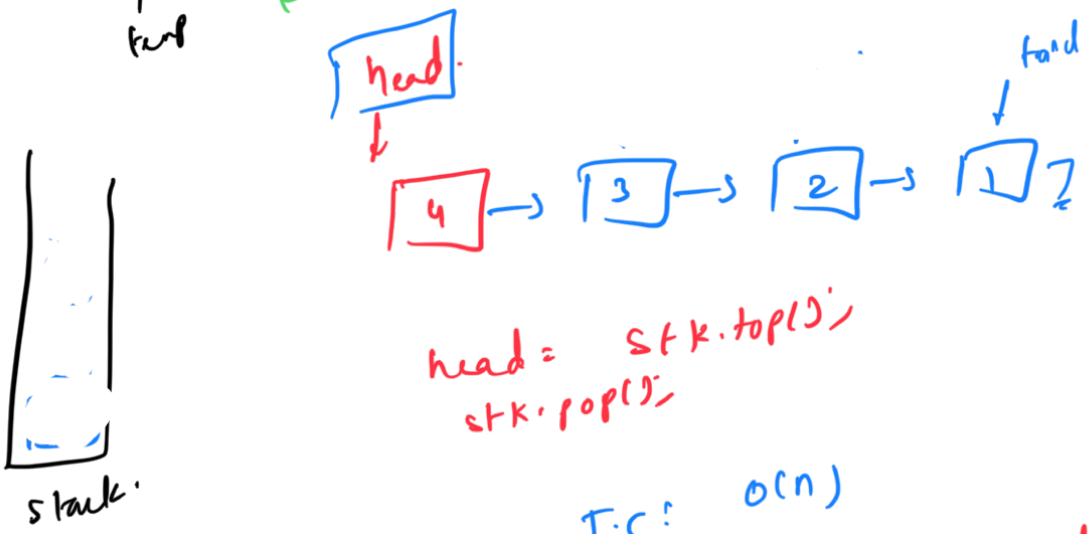
null.next

Question: Reverse a Linked List.



Approach1:



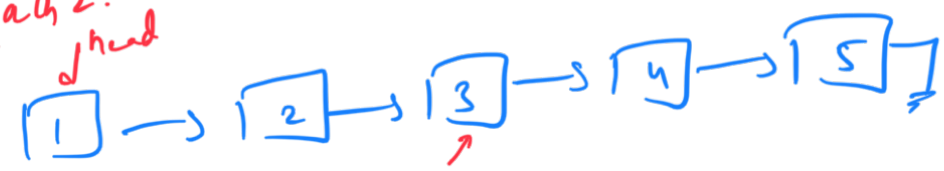




```
head = stk.top();
stk.pop();
```

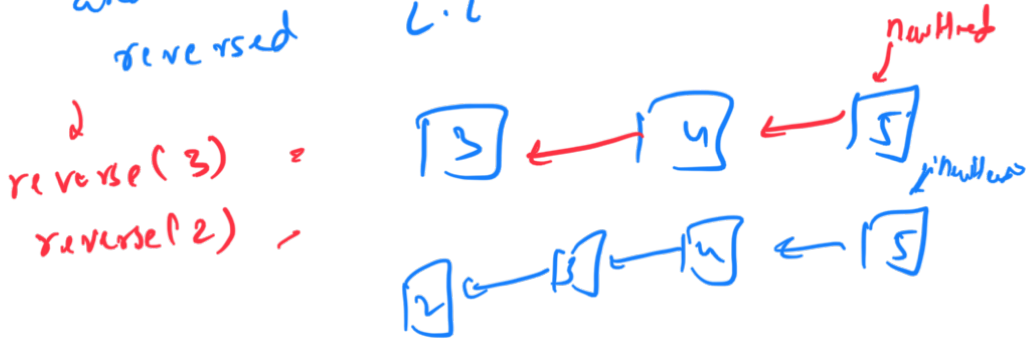
T.C: O(n)

S.C: O(n) → stack

Approach 2: Recursive call stack.

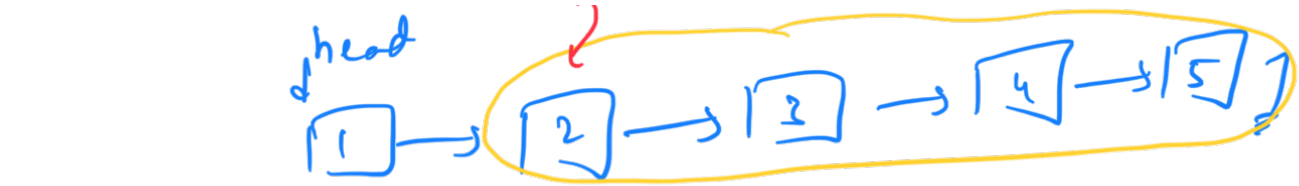



1) Assumption: reverse(head) reverses the Linked List and returns the head of the reversed L.L

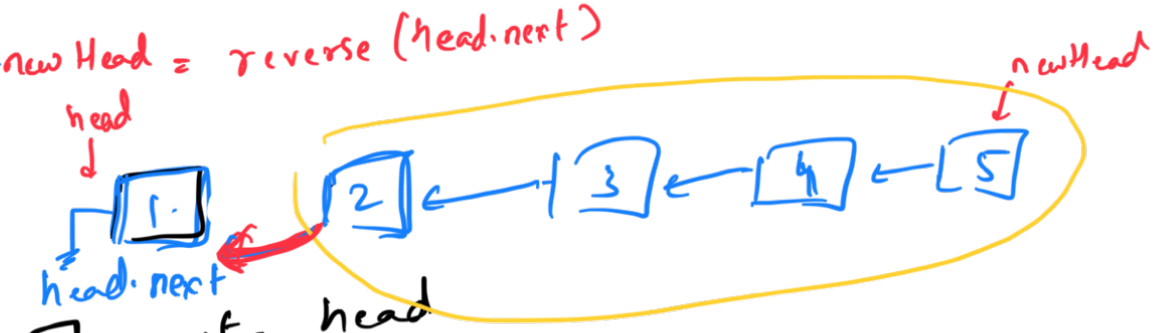



2) Main Logic / Recursive Relation:

How to break a bigger problem into smaller subproblem




5 ← newHead = reverse (head.next)

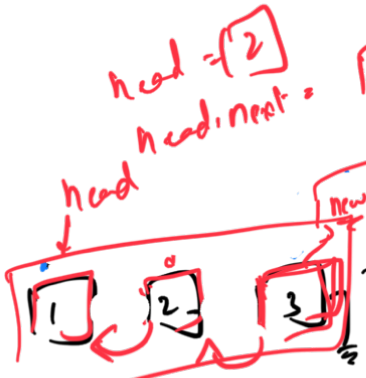



2.next = head

2 ⇒ head.next

(head.next).next = head

head.next = null







```
reverse (head) {
  if(head == NULL || head -> next == NULL)
    return head;
  newHead = reverse (head.next);
  head.next.next = head;
  head.next = null;
  return newHead;
}
```

rev(2.next)

newHead = 3

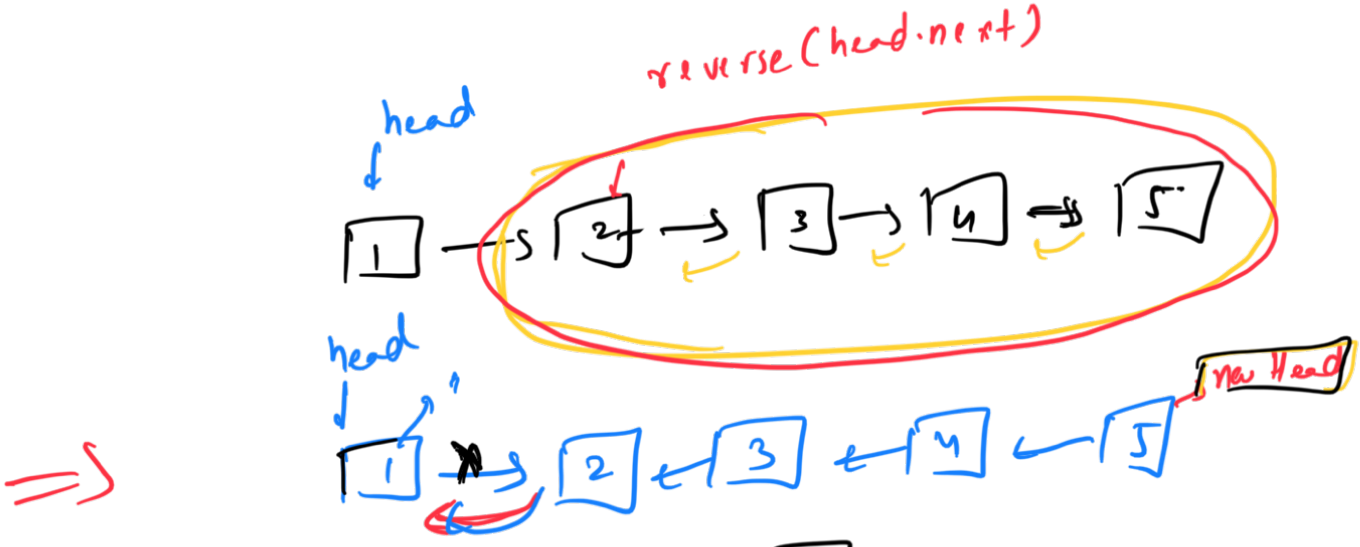
Steps: Base Case

1) Empty LL

if(head == NULL) return NULL;

2)


if(head.next == NULL) {
  return head;


head.next = 1

(head.next).next = head.

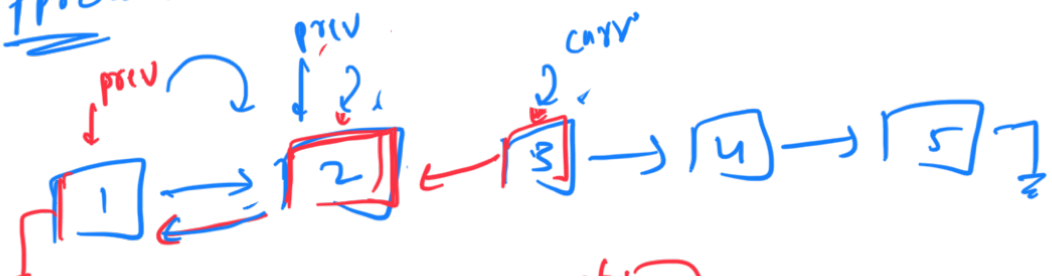
head.next.next = head.

head.next = NULL;

T.C: O(n)

S.C: O(n) [Recursion Stack]

Approach: Iteration.

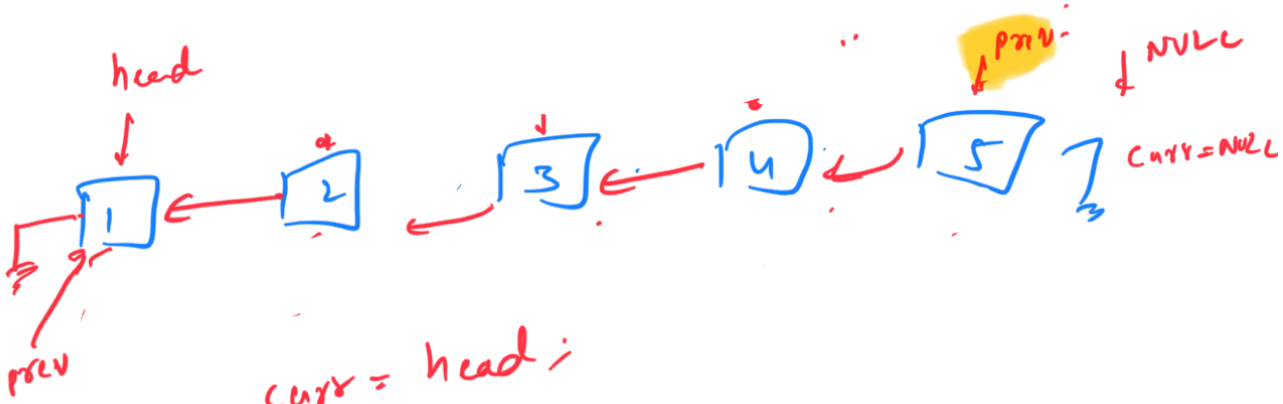


next = curr.next;
curr.next = prev;

curr = curr->next;

curr = 1

prev = curr
curr = next;




curr = head;
prev = NULL

```
next = curr.next;
curr.next = prev;
prev = curr;
curr = next
```

reverse (head) {

prev = NULL, curr = head

while ( curr != NULL ) {

next = curr.next;
curr.next = prev;
prev = curr;
curr = next;

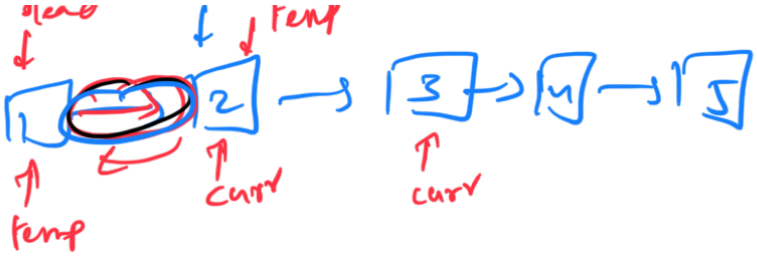
}

return prev;

}

T.C: O(n)
S.C: O(1)


node fwp




node = temp.next
temp.next = NULL ;

**Question**: Print LL

```
void printLL(head){
      temp = head;
      while(temp != null){
            print(temp.data);
            temp = temp.next;
      }
}
```

T.C : O(n), S.C : O(1)

**Question**: Length of LL

```
int lengthLL(head){
      temp = head;
      int len = 0;
      while(temp != null){
            len++
            temp = temp.next;
      }
return len;
}
```

**Insert at the beginning of LL**

```
insertBeginning(head, data){
      newNode = new Node(data);
      newNode.next = head;
      head = newNode;
return head;
}
```

**Insert at end of LL using tail pointer**

```
head = tail = NULL;
insertAtEnd(int val){
      Node* newNode = new Node(val);
      if(head == NULL){
            head = tail = newNode;
      }
      else{
            tail->next = newNode;
            tail = newNode;
      }
}
```

**Insert at end of LL**

```
Node* insertEnd(Node* head, data)
      Node* newNode = new Node(data);
      if(head == NULL)
            head = newNode;

      Node* temp = head;
      while(temp->next != NULL)
            temp = temp->next;

      temp->next = newNode;

return head;
}
```

**Insert at kth position**

```
Node* insertAtPosition(Node* head, int pos, int data){
      int num_nodes = countNodes(head);
      if(pos > num_nodes + 1 || pos < 1)
            return NULL;     // Invalid Input

      Node* newNode = createNode(data);
      if(head == NULL){
            head = newNode;
            return head;
      }

      if(pos == 1){
            newNode ->next = head
            head = newNode;
            return head;
      }

      curr = head;
      int count = 1;
      while(curr && count != pos-1){
            curr = curr->next;
            count++;
      }
      newNode->next = curr->next;
      curr->next = newNode;

return head;
}
```

**Reverse a LL using stacks**

```
reverseUsingStacks(Node* head){
      stack<Node*> s;
      Node* temp = head;
      while(temp!=NULL){
            s.push(temp);
            temp=temp->next;
      }

      head = s.top();
      Node* tail = s.top();
      s.pop();

      while(!s.empty()){
            tail->next = s.top();
            tail = s.top();
            s.pop();
      }
      tail->next = NULL;
      return head;
}
```

**Reverse a LL using recursion**

```
Node* reverse(Node* head){
      if(head == NULL) return head;
      if(head->next == NULL) return head;

      Node* nH = reverse(head->next);
      head->next->next = head;
      head->next = NULL;
      return nH;
}
```

**Reverse a LL using iterative method**

```
Node* reverse(Node* head){
      if(head == NULL || head->next == NULL)
            return head;

      Node* curr = head, prev = NULL, next = NULL;
      while(curr != NULL){
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
      }
      head = prev;
return head;
}
```

**Middle Node**

```
Node* middleNode(Node* head){
     Node* slow = head, *fast = head;
     while(fast != NULL && fast->next !=NULL){
           slow = slow->next;
           fast = fast->next->next;
     }
     return slow;
}
```