CLASS → Custom Data type

{ Roll no, Name } ⇒ Student

int x ⇒ 10

```
Class Student {
    int roll_no;
    String name;

    Student (int r
             String n ){
        roll_no ⇒ r;
        name    ⇒ n;
    }
}
```

Student yash = new Student();

↓

Object

yash.roll_no ⇒ 33;
yash.name = "Yash Ray";

Student yash =
new Student(33, "Yash Ray");

yash →
rahul →

roll_no = 33
name = "Yash Ry"

Student rahul = yash.

Static Arrays

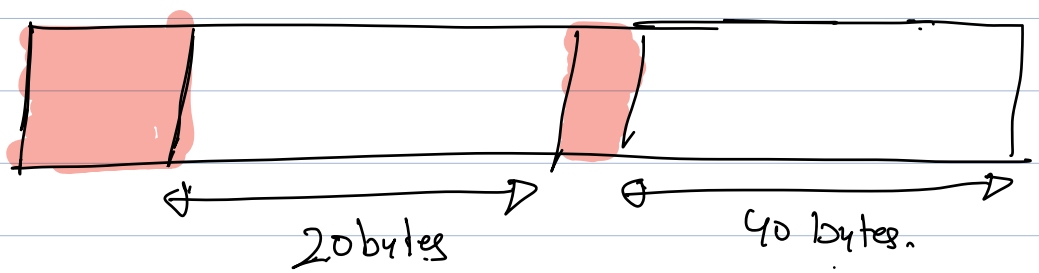
Advantages

1) O(1) access
2) O(1) insertion

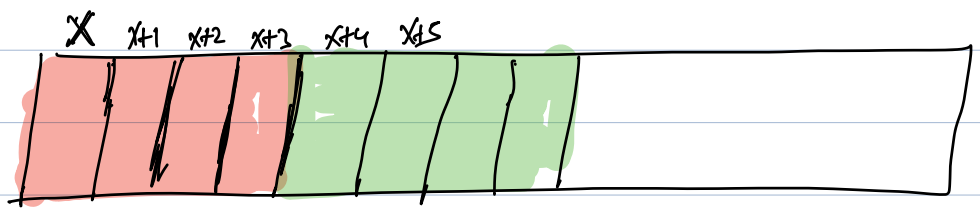
Disadvantage

1) No deletion position.
2) Fixed size.
3) Memory is not utilized optimally.

Contigues memory allocation



int x[10] ⇒ 40 bytes



Zero ⇒ x

first ⇒ x+4 ⇒ x+(1)4

Second ⇒ x+8 ⇒ x+(2)4

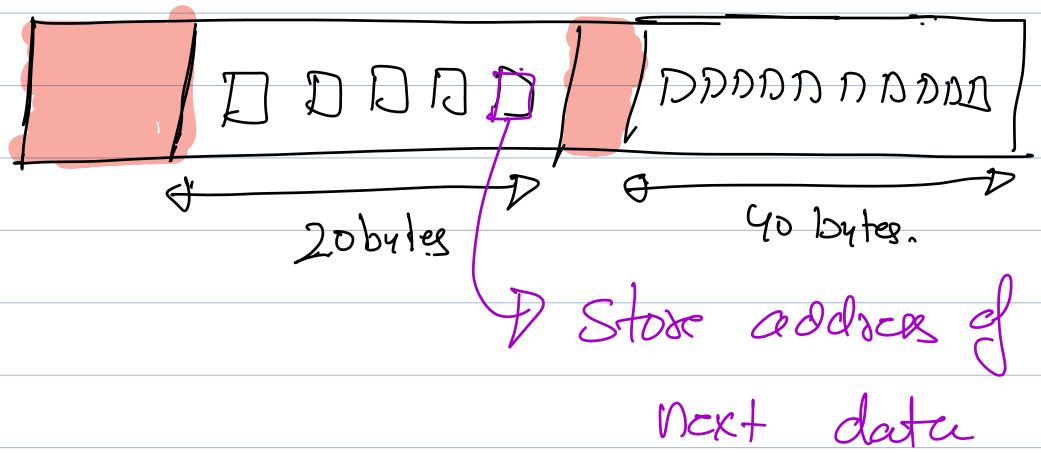
Dynamic Arrays

Advantages

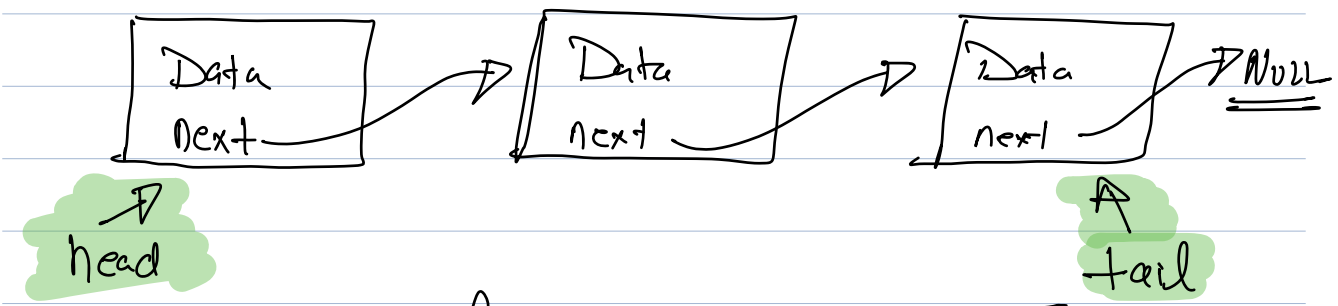
1) O(1) access
2) O(1) insertion.
3) Size can be changed.

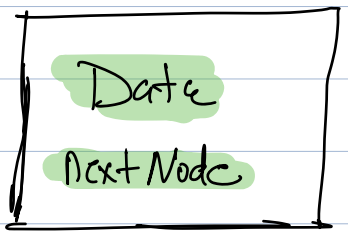
Disadvantage.

1) Memory is not utilized optimally.



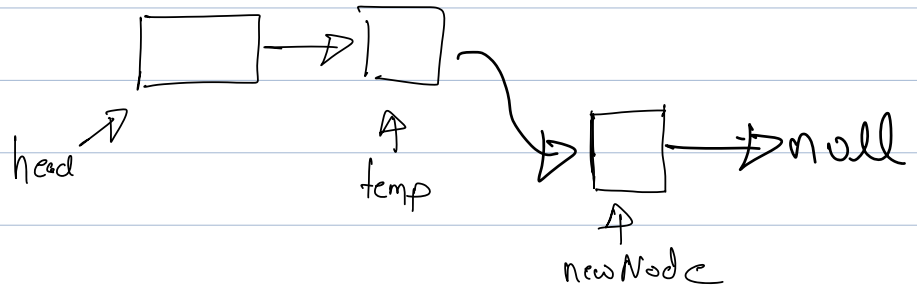
Linked list






```
class Node {

    int data;
    Node next;

    Node (int d) {

        data = d
        next = null
    }
}
```

Q Inserts a node at the end of a linked list.




```
Node insert_at_end ( int data , Node head)

    Node newNode = new Node (data)

    if (head == null) {
        head = newNode;
        return head
    }

    Node temp = head;

    while ( temp.next != null ) {
        temp = temp.next;
    }

    temp.next => newNode

    return head;
}
```

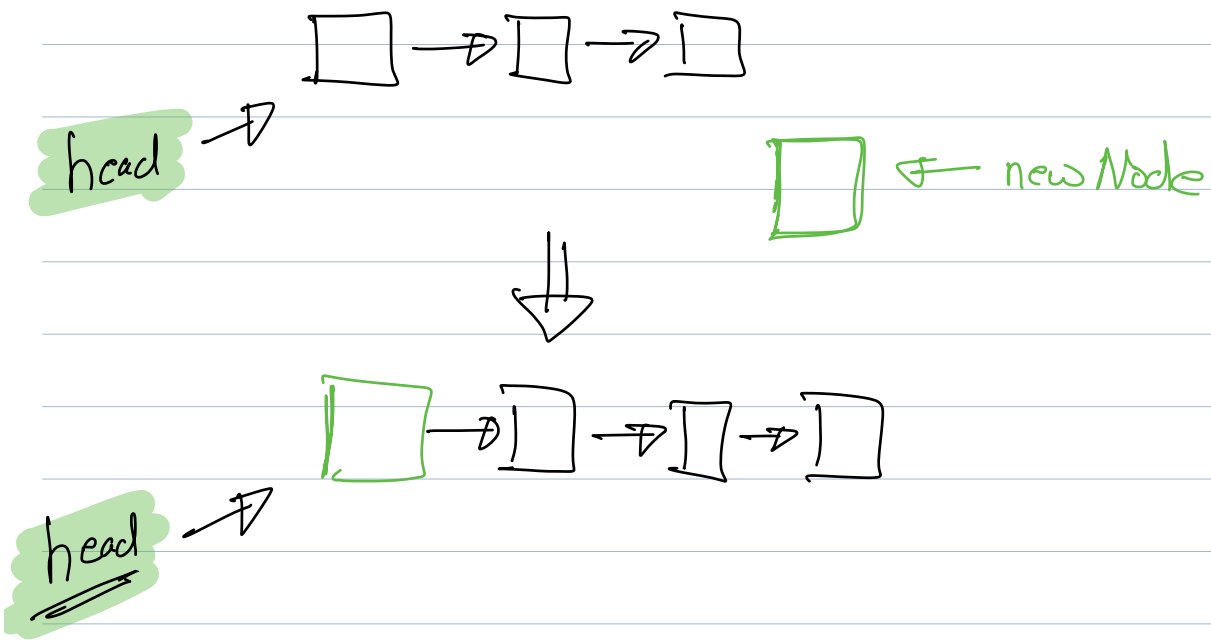
Edge Cases

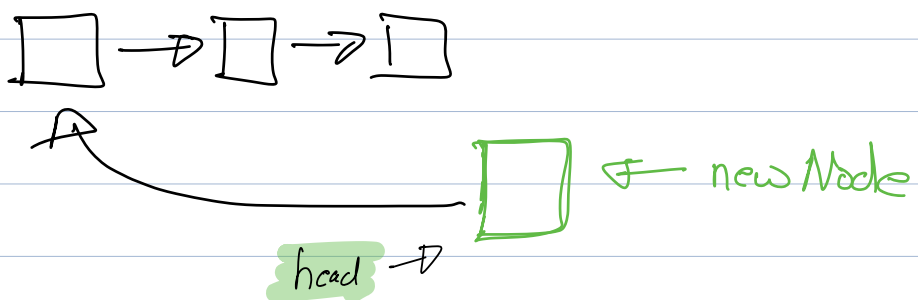
1) If head was pointing to null.

head → null

2) When linked list has only 1 node.

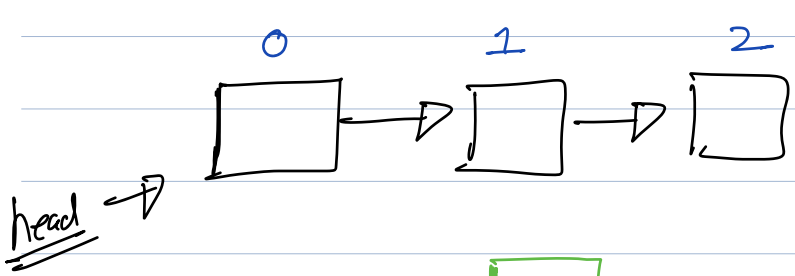
Q Insert at beginning





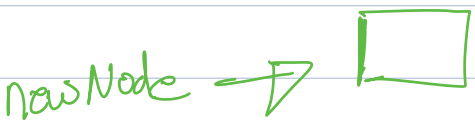

```
Node insert_at_beginning (int d, Node head) {

    Node newNode = new Node(d)
    newNode.next = head.
    head => newNode
    return head;
}
```

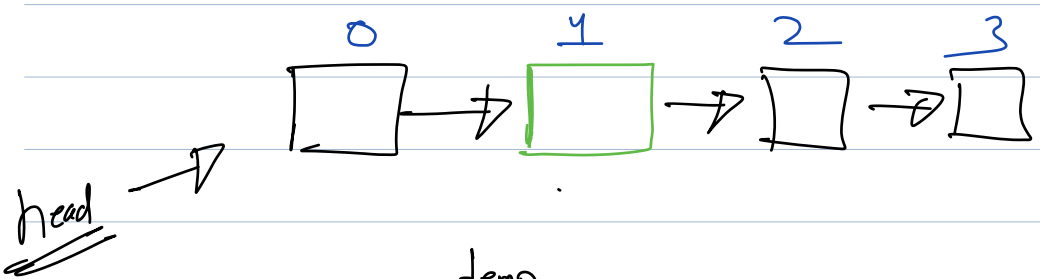
Q Insert at k^{th} position.

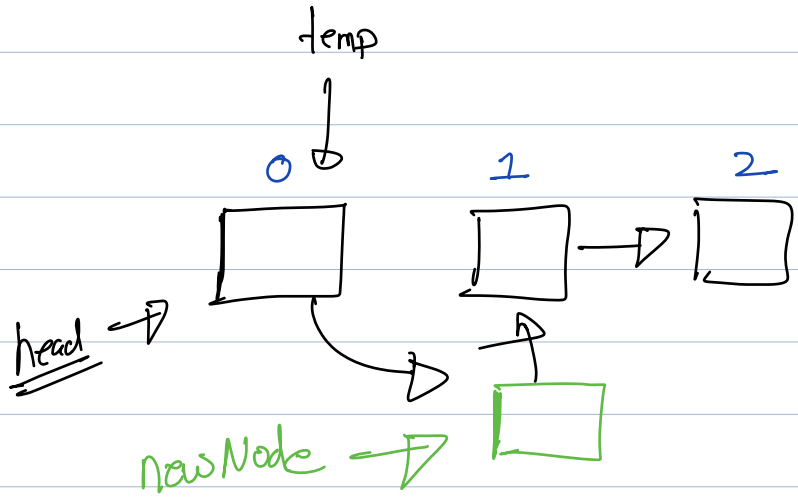


⇒ k = 1

⇒ data.







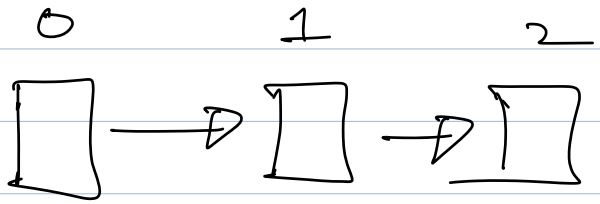
k = 1

newNode.next ⇒ temp.next

temp.next ⇒ newNode.

```
Node insertAtK (int d, int K, Node head) {

    if (K==0)
        return insert_at_beginning (d, head);

    if (head == null)
        return head;

    Node newNode = new Node (d);
    Node temp = head;

    for (int i=1 ; i<K; i++) {

        if (temp.next == null)
            return head;

        temp = temp.next;
    }

    newNode.next ⇏ temp.next
    temp.next ⇒ newNode.

    return head;

}
```

1) If k is larger than size of linked list.



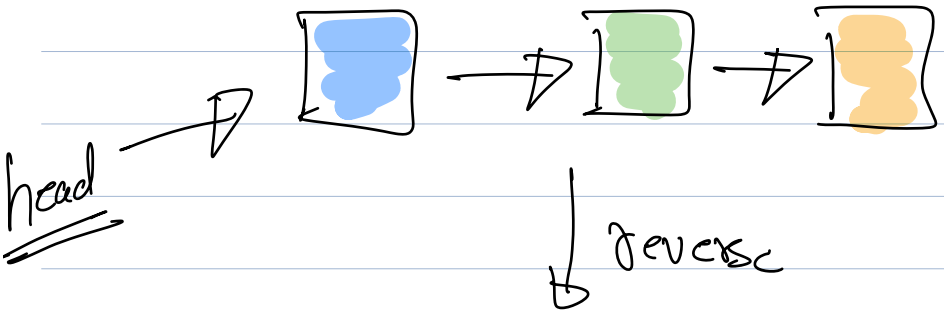
k = 3

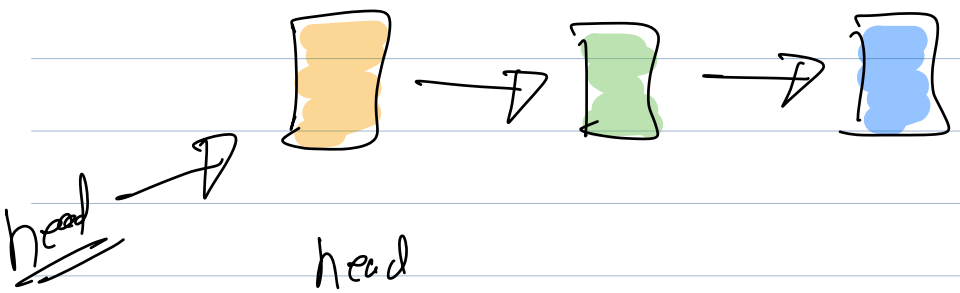
if (size < k)
return head.

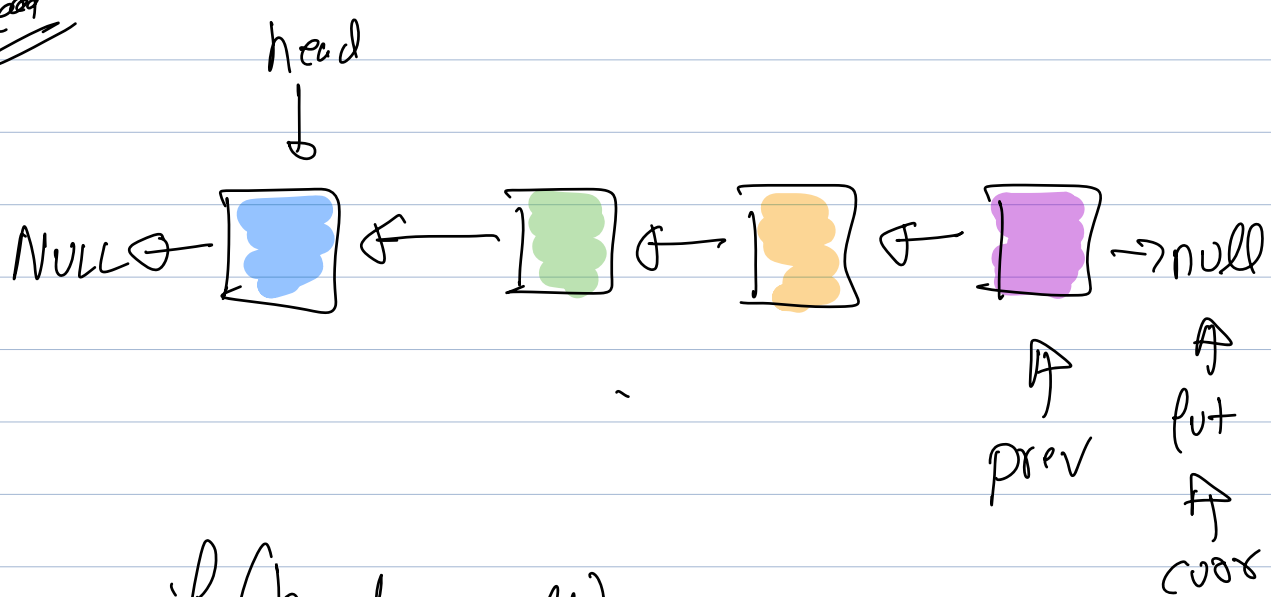
2) k = 0

3) head == null

Q Reverse a linked list. SC: O(1)








```
if (head == null)
    return head;
```

curr → head

prev → null.

nxt → head.next

```
while (curr != null) {
    curr.next = prev;
    prev = curr;
    curr = fut;
    if (fut != null) {
        fut = fut.next;
    }
}
head = prev;
return head;
}
```

2nd Approach

```
curr = head
prev = null.
while (curr != null)

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

}

return prev
```