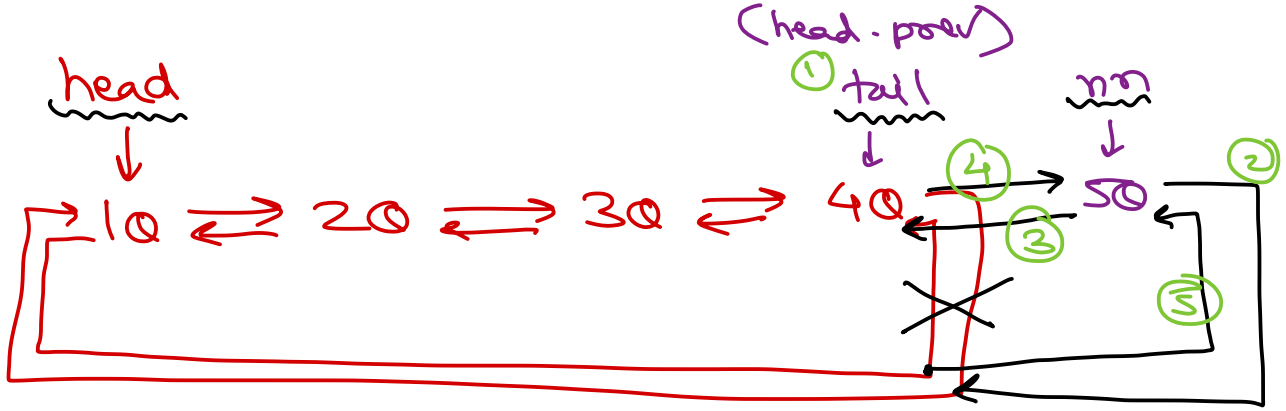
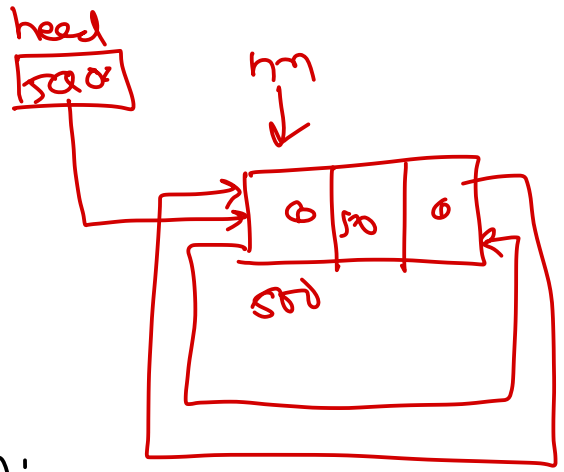
Data Structure & Algorithms

Sunbeam Infotech

Linked List - Doubly Circular List - addLast

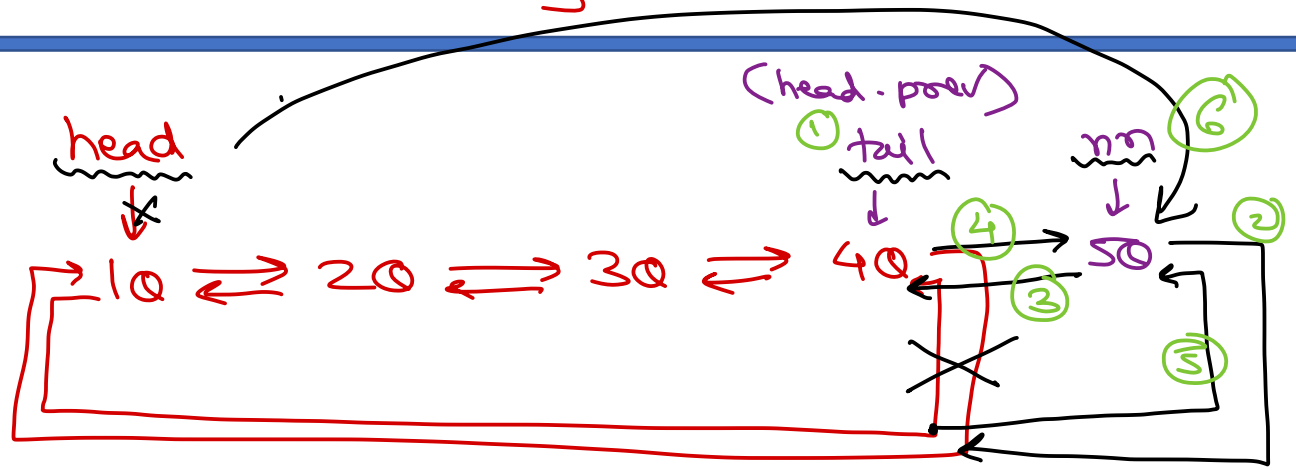


1. tail = head.prev;
2. nn.next = head;
3. nn.prev = tail;
4. tail.next = nn;
5. head.prev = nn;

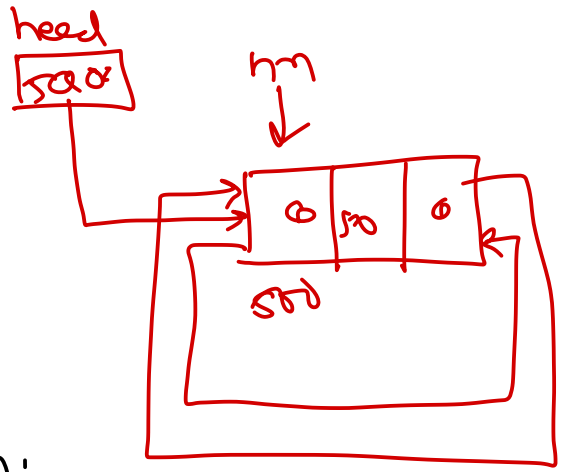


head = nn;
nn.next = head;
nn.prev = head;

Linked List - Doubly Circular List - add First

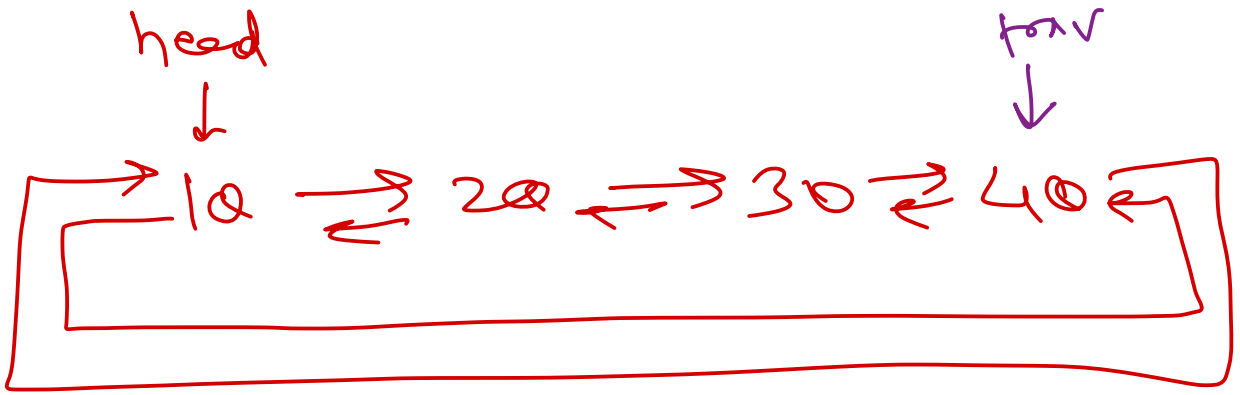


1. tail = head.prev;
2. nn.next = head;
3. nn.prev = tail;
4. tail.next = nn;
5. head.prev = nn;
6. head = nn;



head = nn;
nn.next = head;
nn.prev = head;

Linked List - Doubly Circular List



fwd display

```
trav = head;
do {
    pf(trav.data);
    trav = trav.next;
} while(trav != head);
```

rev display

```
trav = head.prev;
do {
    pf(trav.data);
    trav = trav.prev;
} while(trav != head.prev);
```

Linked List

Doubly Cir List

Time Complexities

1. add first - O(1)
2. add last - O(1)
3. del first - O(1)
4. del last - O(1)
5. display/trav - O(n)
6. add/del at pos - O(pos) ... O(n)
7. find ele - O(n)
   linear search

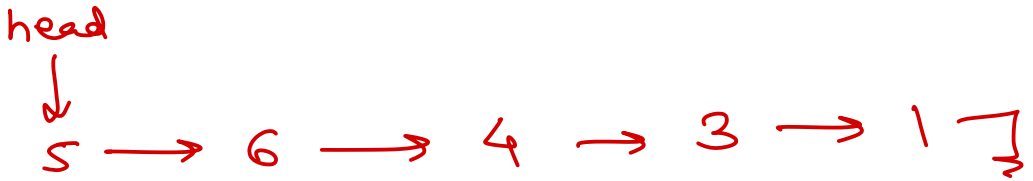
Linux Kernel

- all lists are doubly circular list.

1. job queue / process table
2. ready queue
3. waiting queue
4. message queue
5. inode table/ inode cache

Linked List – Competitive programming

- Sort the singly linked list.

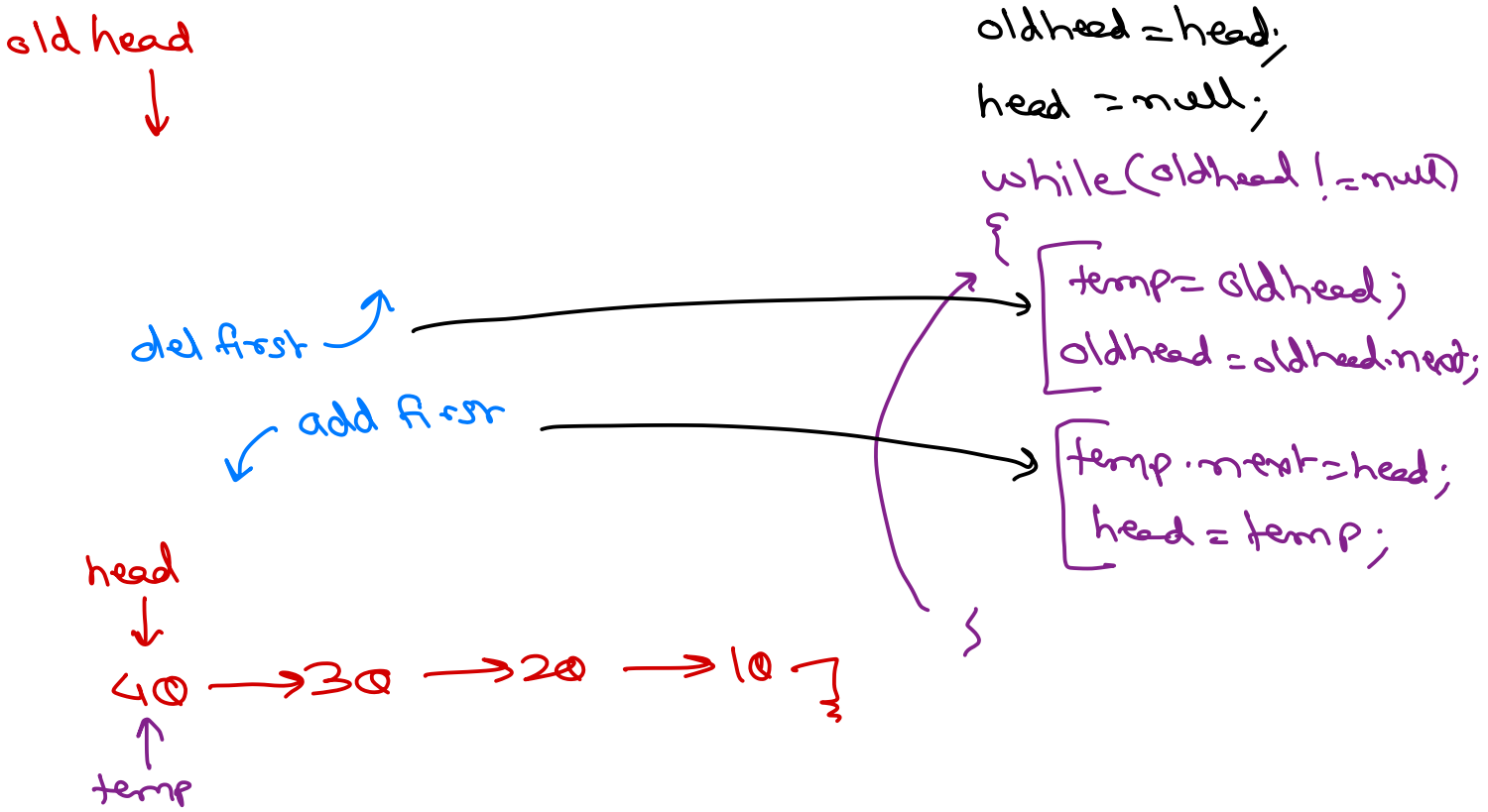


Selection sort

```
Node i,j; i.next!=null
for(i=head ; i !=null ; i=i.next) {
    for(j=i.next ; j !=null ; j=j.next) {
        if( i.data > j.data )
            swap(i.data, j.data);
    }
}
```

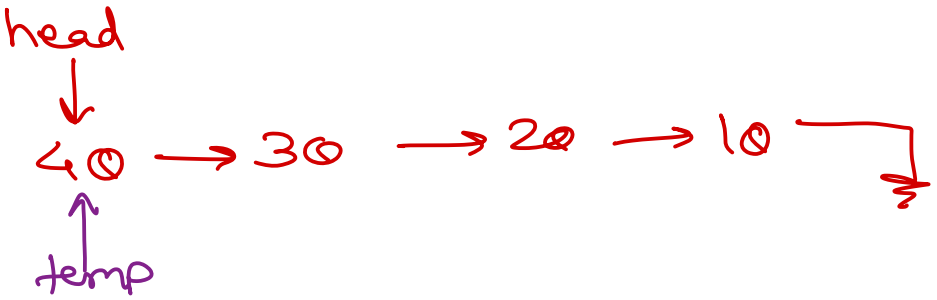
Linked List – Competitive programming

- Reverse singly linked list.



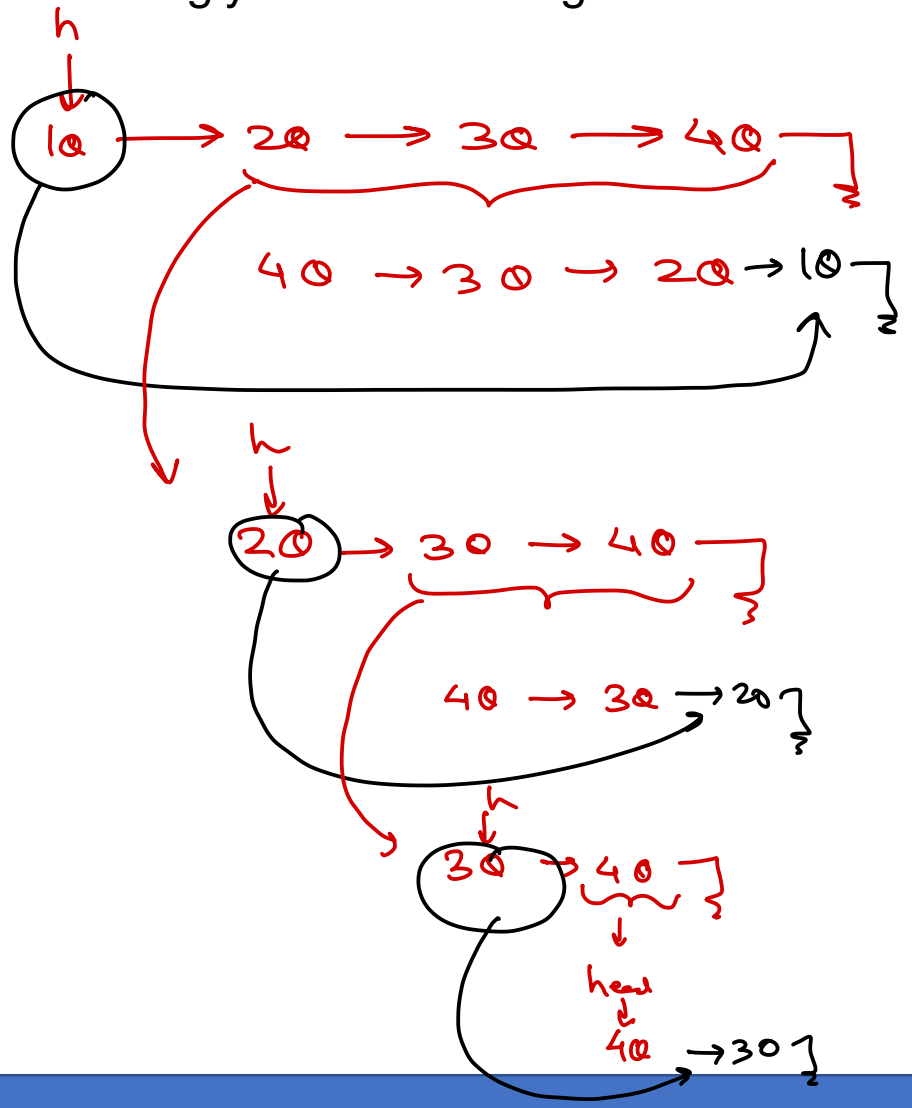
Linked List – Competitive programming

- Reverse singly linked list.



Linked List – Competitive programming

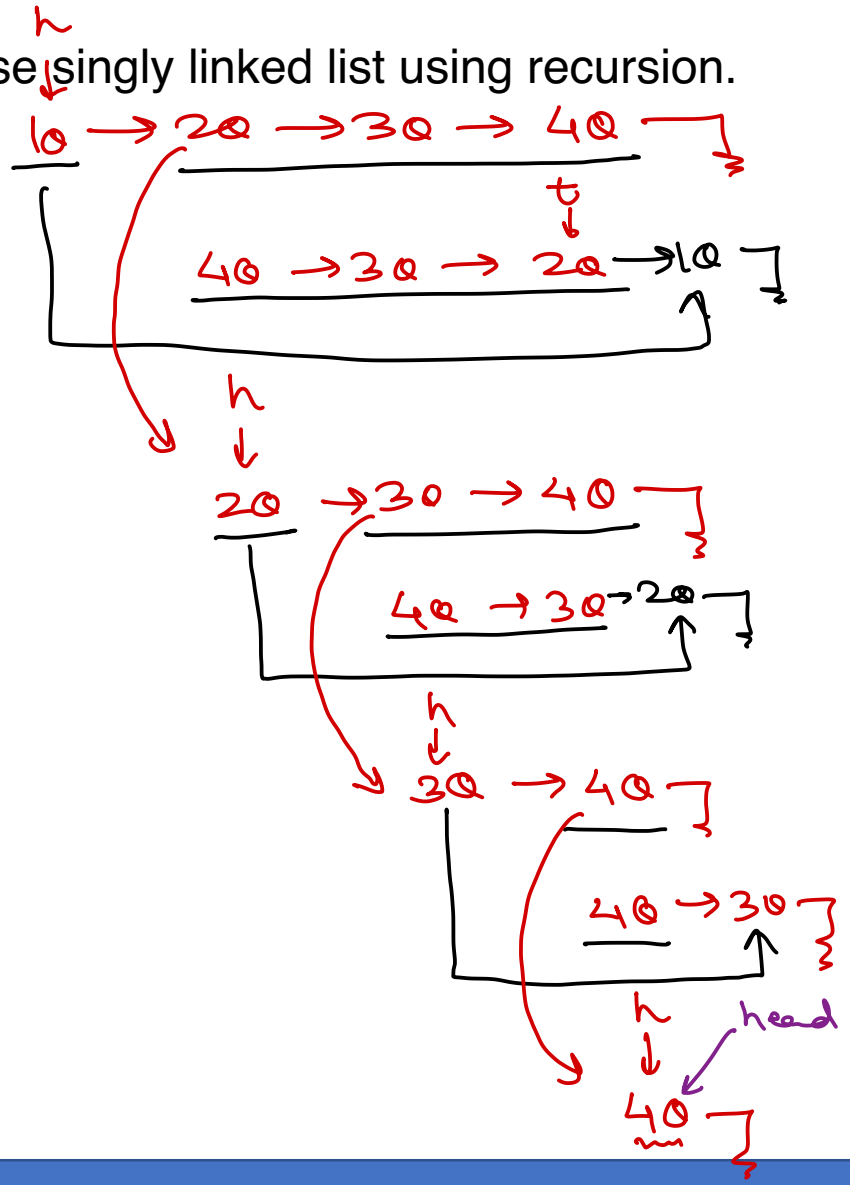
- Reverse singly linked list using recursion.




```
Node recRev(Node h) {
    if(h.next == null) {
        head = h;
        return h;
    }
    t = recRev(h.next);
    t.next = h;
    h.next = null;
    return h;
}
```

Linked List – Competitive programming

- Reverse singly linked list using recursion.



① if list has single node, mark it as head & return it (it is last node)

① reverse rest of list (from next)

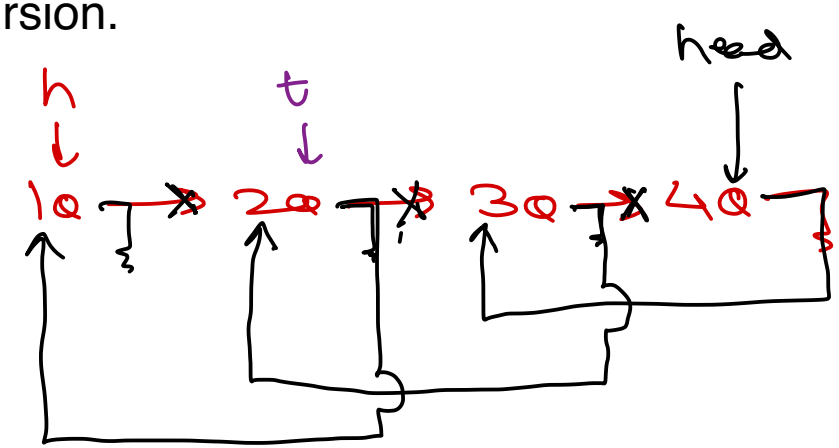
② add cur node (h) after last node (t)

③ make cur node next as null.
(now cur node became last node).

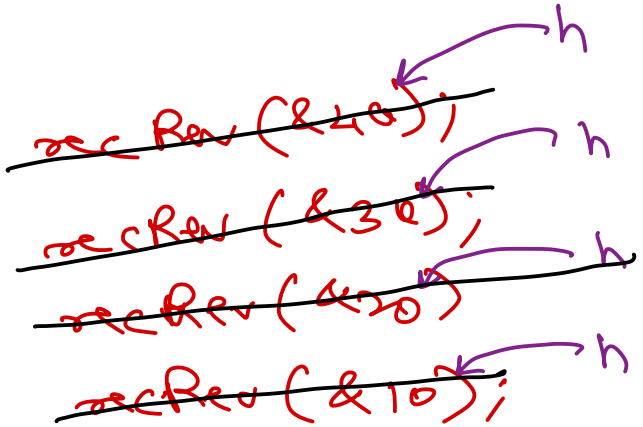
④ return cur node (last node);

Linked List – Competitive programming

- Reverse singly linked list using recursion.



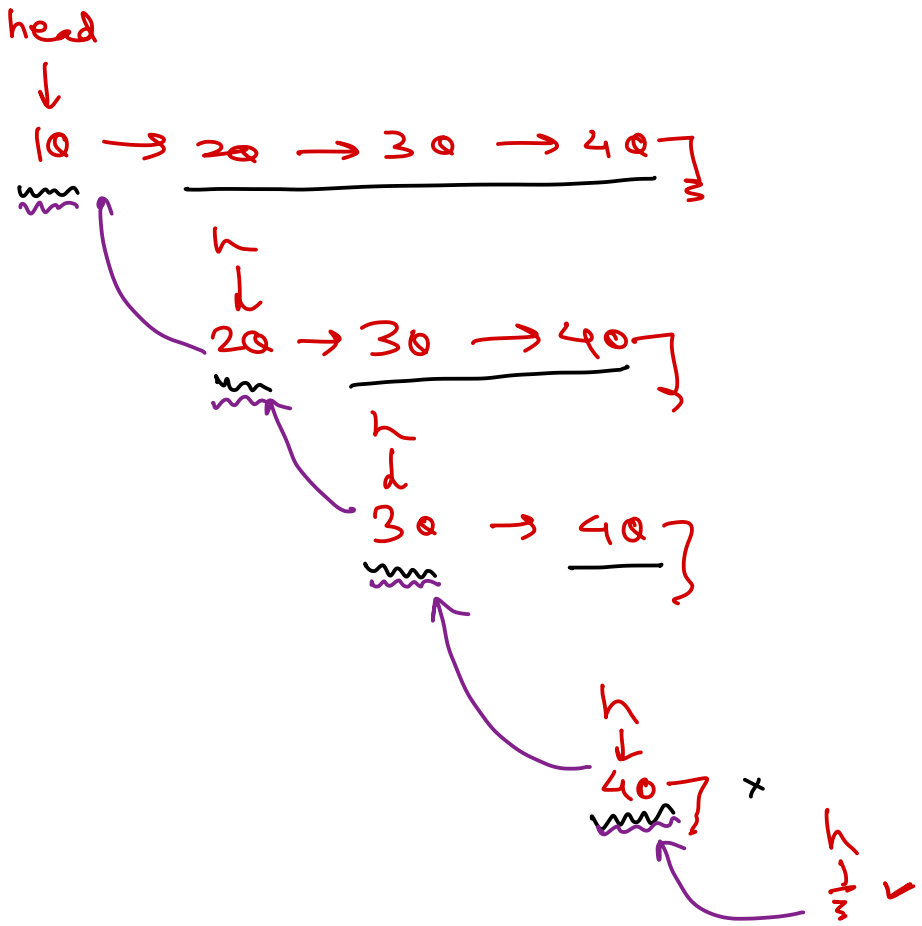

```
Node recRev(Node h){
    if(h.next==null){
        head = h;
        return h;
    }
    Node t= recRev(h.next);
    t.next = h;
    h.next=null;
    return h;
}
```



Linked List – Competitive programming

- ~~Reverse singly linked list using recursion.~~

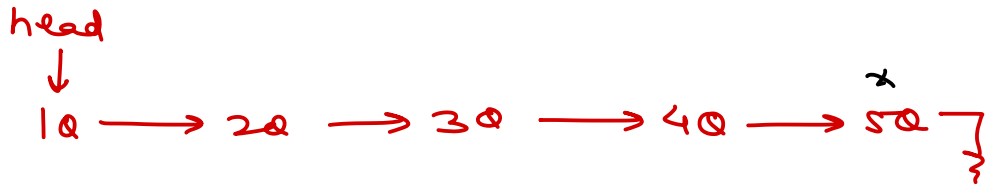
Display singly list in reverse order.

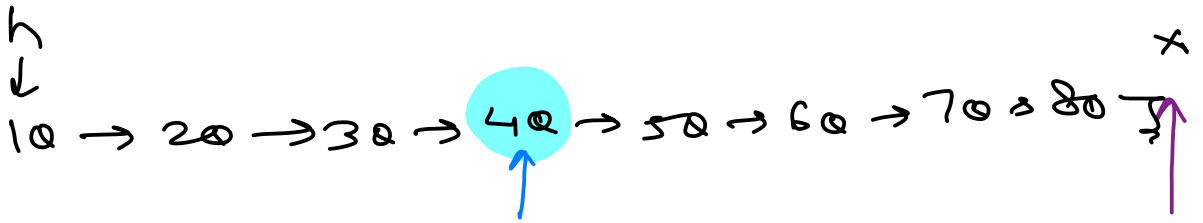



```
rev Display( h) {
→ if list is empty, return;
→ display rest of list (after recur);
→ display cur node;
}
```

Linked List – Competitive programming

- Find middle of singly linear linked list.





approach 1

1. traverse list & count num of nodes.
2. traverse till count/2.

approach 2

1. take two pointers - fast & slow.
2. traverse fast pointer: fast = fast.next.next;
3. traverse slow pointer: slow = slow.next;

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

even nodes

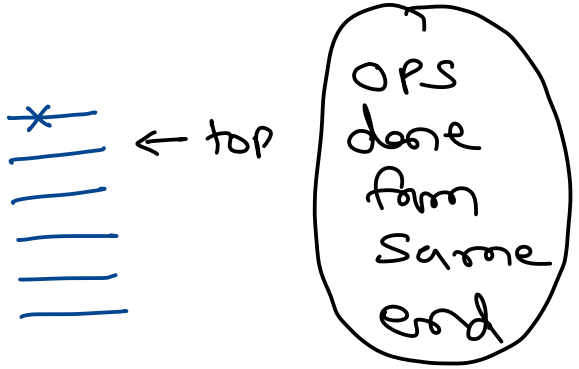
odd nodes

?

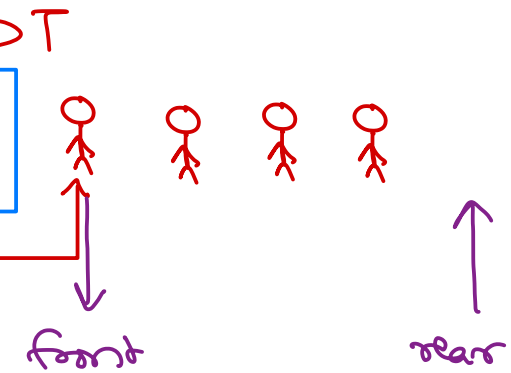
```
slow.data → return;
```

Stack and Queue

- Stack & Queue are utility data structures. → temp storage during process.
- Can be implemented using array or linked lists.
- Usually time complexity of stack & queue operations is O(1).
- Stack is Last-In-First-Out structure.
- Stack operations ↔ ADT
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()*



- Simple queue is First-In-First-Out structure.
- Queue operations – ADT
  - push() / enque()
  - pop() / deque()
  - peek()
  - isEmpty()
  - isFull()*

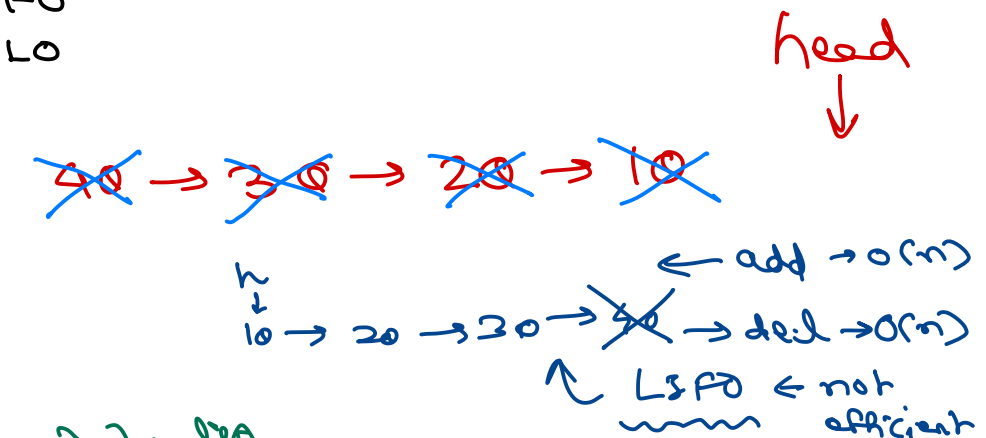


- Queue types
  - Linear queue
  - Circular queue
  - Deque
  - Priority queue

Stack / Queue using Linked List

- Stack can be implemented using linked list.
  - add first → push() → O(1)
  - delete first → pop() → O(1)
  - is empty → isEmpty() → O(1)
  - peek() → return head.data; → O(1)

LIFO
FILO



- Queue can be implemented using linked list.
  - add last → push()
  - delete first → pop()
  - is empty → isEmpty()
  - peek() → return head.data;

Singly list with head
inefficient → O(n)
O(1)
O(1)
O(1)

Singly list with head & tail ✓
O(1)
O(1)
O(1)
O(1)

FIFO
LILO

Stack - using array

push:

```
top++;
arr[top] = val;
```

peek:

```
return arr[top]
```

pop:

```
top--;
```

isFull:

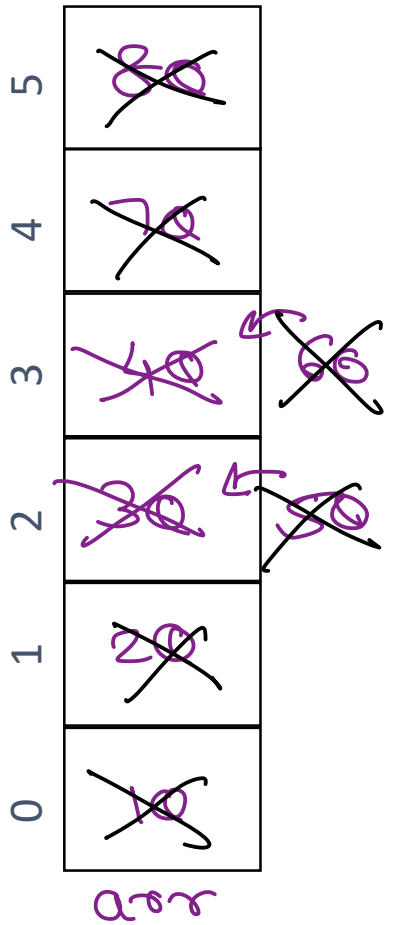
```
top == max-1
```

isEmpty:

```
top == -1
```

init:

```
top = -1;
arr = new int[];
```



top → -1

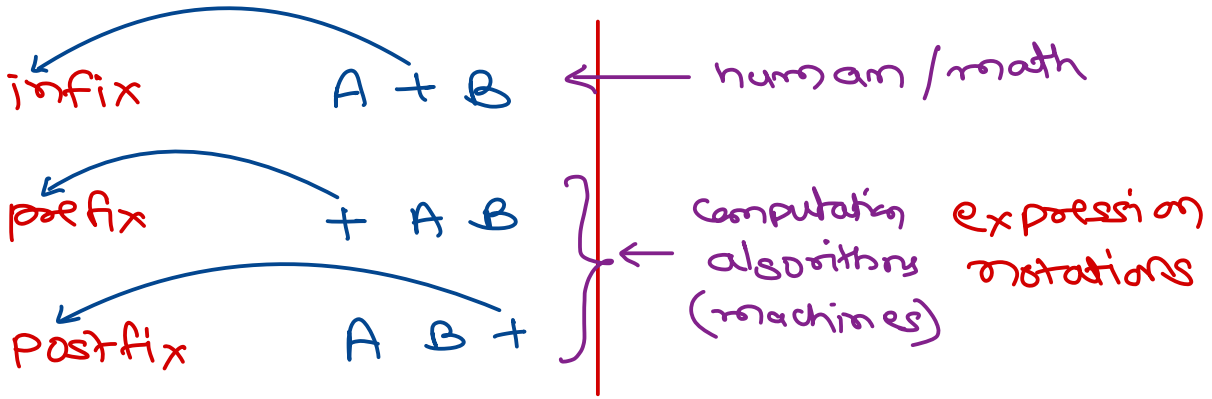
Stack / Queue in Java collections

- class java.util.Stack<E>
  - E push(E);
  - E pop();
  - E peek();
  - boolean isEmpty();

- interface java.util.Queue<E>
  - boolean offer(E e); - push
  - E poll(); - pop
  - E peek();
  - boolean isEmpty();

ArrayDeque<>
LinkedList<>

Expression notations.



infix → result ✓

OR

infix → postfix → result

infix → prefix → result

precedence

( )

$

* / %

+ −

← higher
←
←
← lowest

⑥ ⑦ ⑤ ② ① ⑧ ④ ③

5 + 9 − 4 * (8 − 6 / 2) + 1 $ (7 − 3)

5 + 9 − 4 * (8 − 6 2 /) + 1 $ (7 − 3)

5 + 9 − 4 * 8 6 2 / − + 1 $ (7 − 3)

5 + 9 − 4 * 8 6 2 / − + 1 $ 7 3 −

5 + 9 − 4 * 8 6 2 / − + 1 7 3 − $

5 + 9 − 4 8 6 2 / − * + 1 7 3 − $

5 9 + − 4 8 6 2 / − * + 1 7 3 − $

5 9 + 4 8 6 2 / − * − + 1 7 3 − $

5 9 + 4 8 6 2 / − * − 1 7 3 − $ +

⑥ ⑦ ⑤ ② ① ⑧ ④ ③

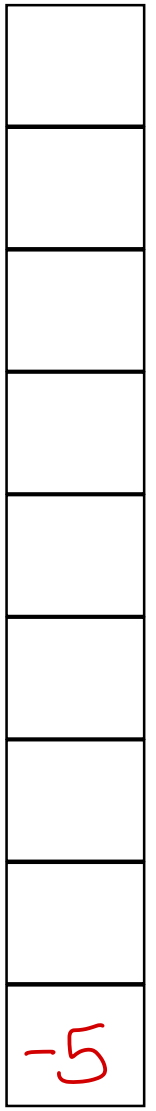
5 + 9 − 4 * (8 − 6 / 2) + 1 $ (7 − 3)

+ − + 5 9 * 4 − 8 / 6 2 $ 1 − 7 3

Postfix Evaluation

operands stack

- 5 9 + 4 8 6 2 / - * - 1 7 3 - $ + ↑



① traverse postfix from left to right.
② if sym is operand, push on stack.
③ if sym is operator, pop two args from stack, calc result & push on stack.
first poped - 2nd op / second poped - 1st op.
④ repeat 2 & 3 until all values from expr are done.
⑤ pop the final result from stack.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>