Day-12 [linked list-1]

Data Structures

1) Arrays

2) Stacks

3) Queues

4) Linked List

→ Linear D.S

5) Trees

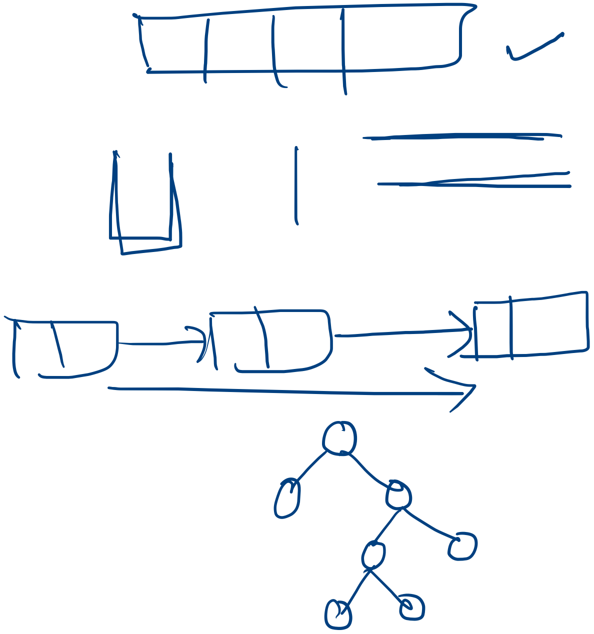
6) Graphs

7) Heaps

→ Non - Linear D.S

8) Hashing

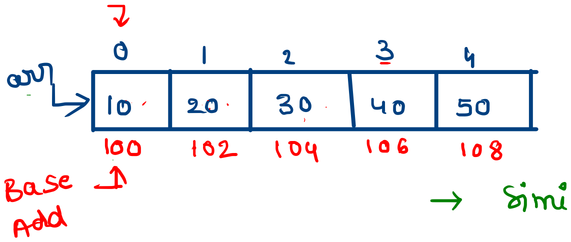
9) Trie , segment tree , etc....



Custom define

Why LL? if we have already Arrays

type of arr: integer [2 Bytes]



→ Similar Datatype

→ contigious memory loc

→ index starts from 0

$arr[3] = 40 \rightarrow O(1)$

$= 100 + (3-0) \times 2$

$= 100 + 6 = 106$ → 40

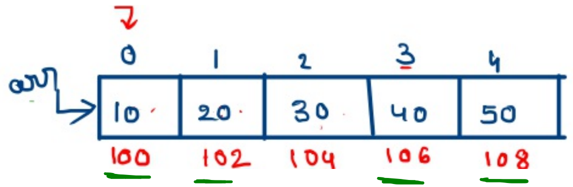
Random Access

= B.A of array + (# of elements to cross) * size of element

Note:- Array name will contain Base add. of array.

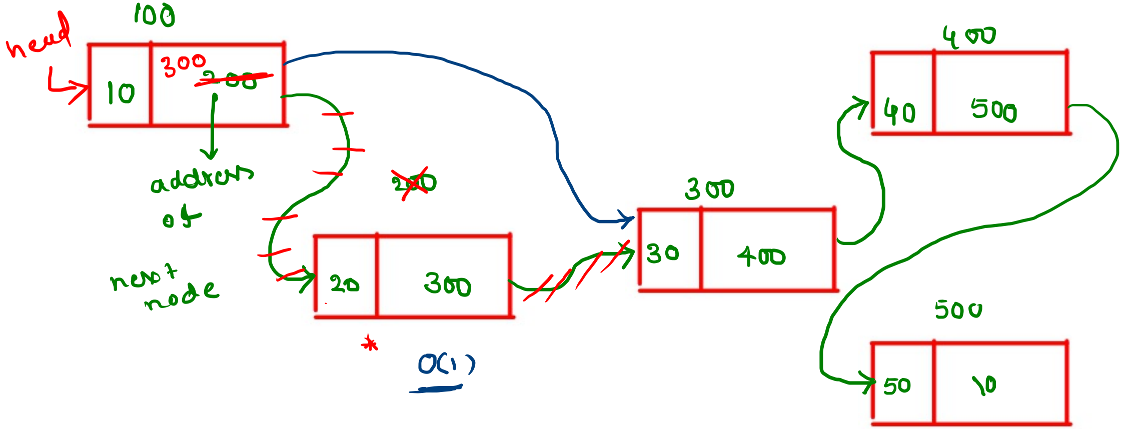
arr[5] = { 10, 20, 30, 40, 50}

\0 ⇒ NULL

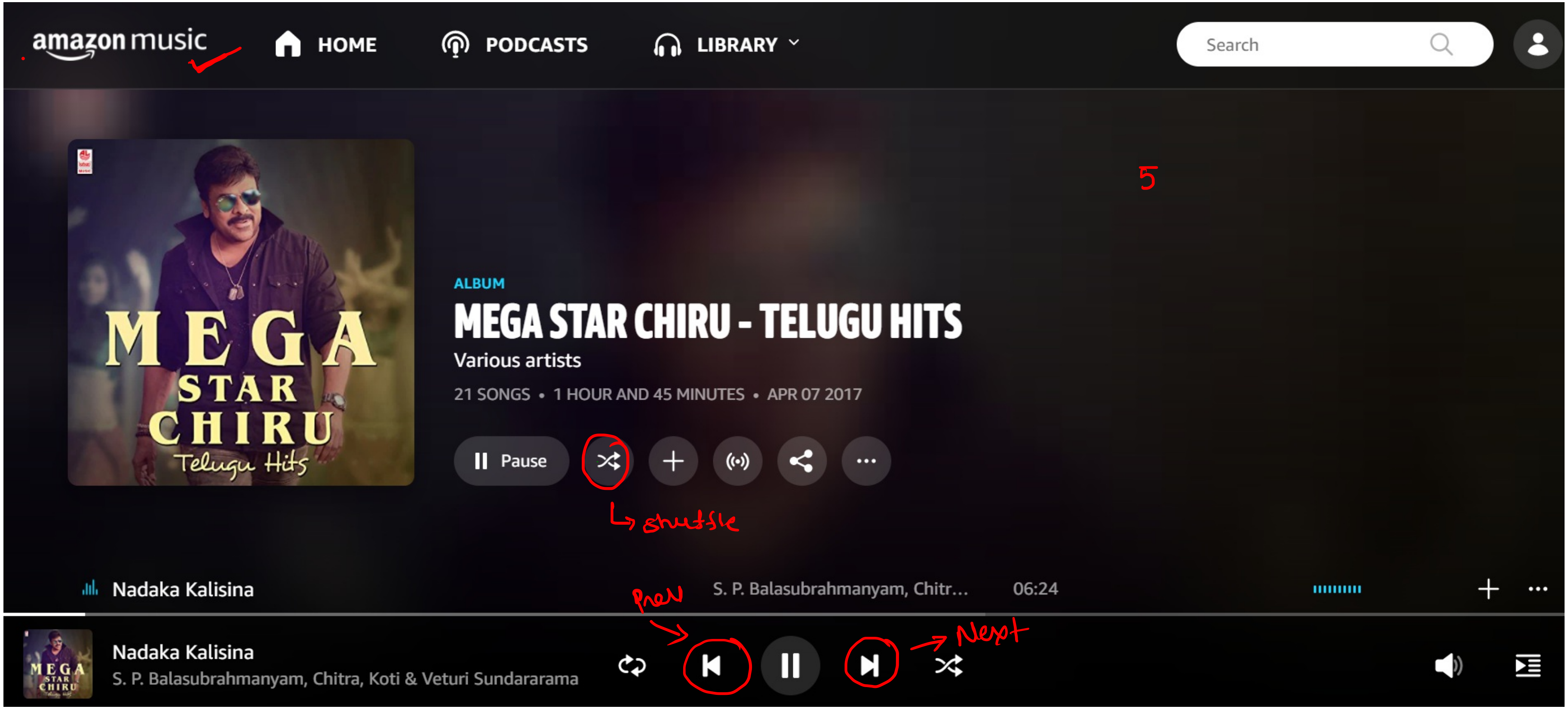


C1 :- delete 1st element

✓ (n-1) moments are required



O(1)



5

amazon music

HOME

PODCASTS

LIBRARY

Search

ALBUM

MEGA STAR CHIRU - TELUGU HITS

Various artists

21 SONGS • 1 HOUR AND 45 MINUTES • APR 07 2017

Pause

↳ shutfle

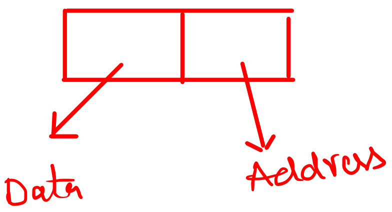
Nadaka Kalisina | S. P. Balasubrahmanyam, Chitr... | 06:24

Nadaka Kalisina

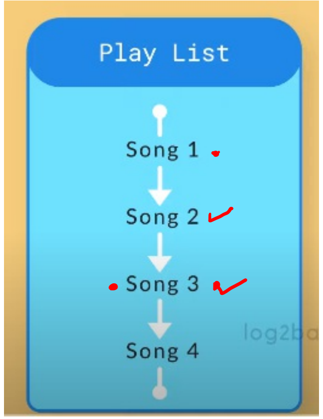
S. P. Balasubrahmanyam, Chitra, Koti & Veturi Sundararama

Prev →

→ Next

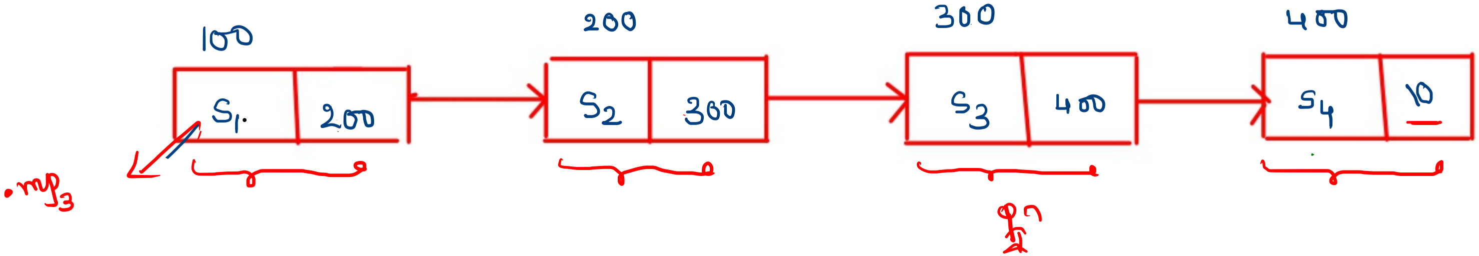


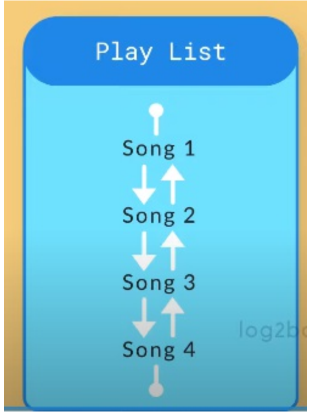
- → Integer
- → character
- → string



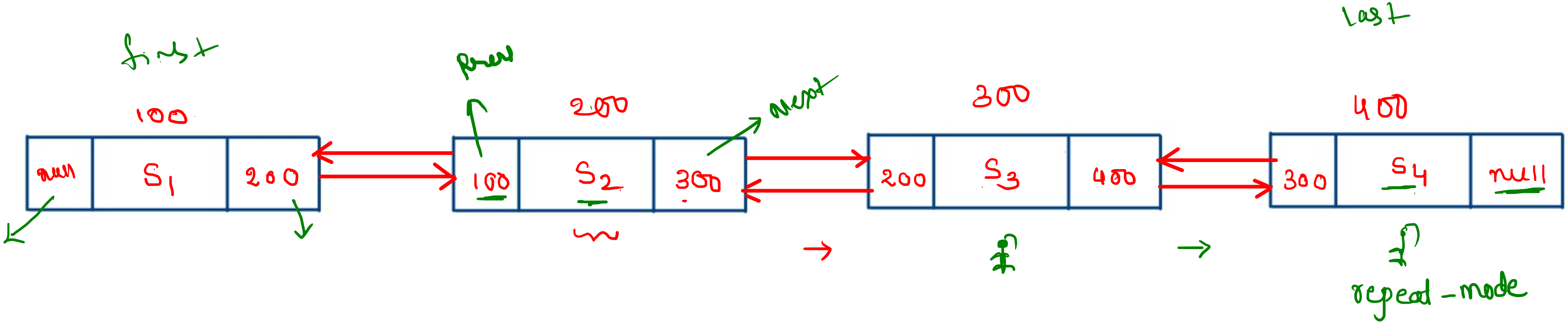
SLL → Traversing 1-way

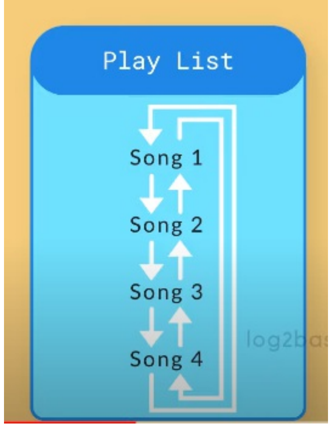
⇒ sequential Access




Play List
Song 1
Song 2
Song 3
Song 4
log2b

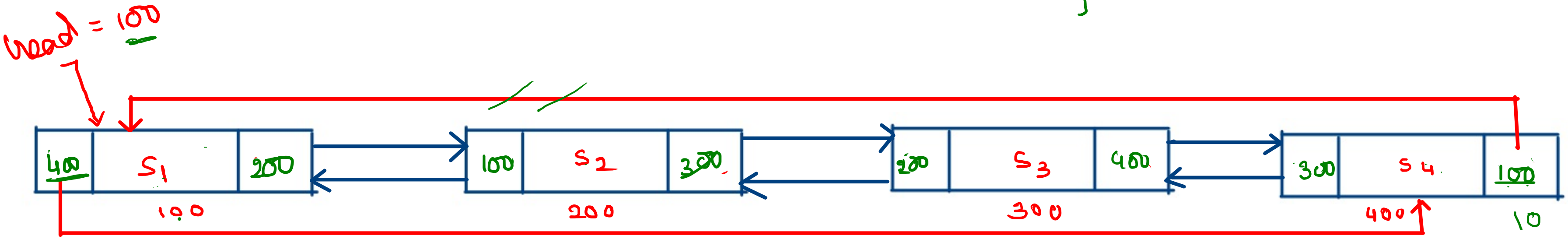
DLL

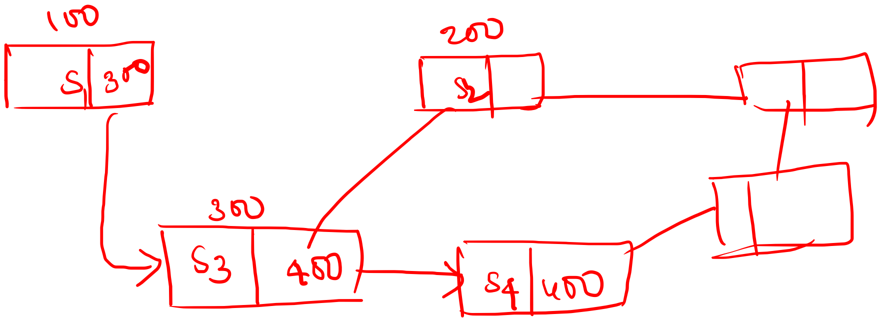

first
prev
next
last
100
200
300
400
null
S_1
200
100
S_2
300
200
S_3
400
300
S_4
null
repeat_mode



Circular DLL

Loop





100
S
300
200
S2
300
S3
400
S4
400

* Arrays v/s Linked List

elements present cont. mem. loc

Arrays:-

← 1)Random Access ✓

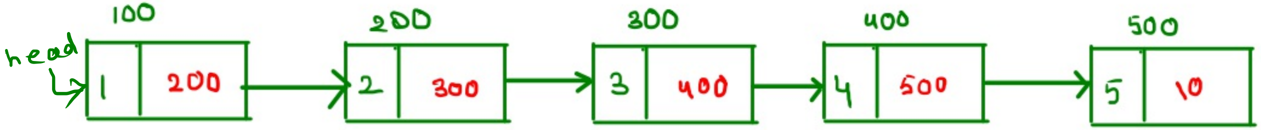
2)elements will prsent at consecutive mem loc ✓

3)Size is always fixed, [Dynamic Arrays]

* Linked List :-

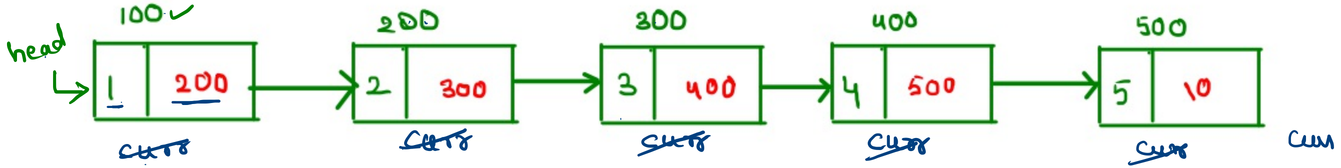
1) Random access not possible x

→ 2) Ease of Insertion and deletion



head
100
1
200
200
2
300
300
3
400
400
4
500
500
5
10

* 1) Print the elements of linked list



100

```
printList(Node head)
{
    if(head==null)
    {
        return
    }
  → Node curr=head
  x while(curr!=null)
    {
        print(curr.data)
        curr=curr.next
    }


}
```

- (dot) operator

curr = ~~100~~ ~~200~~ ~~300~~ ~~400~~ ~~500~~ 10

$\underbrace{1, 2, 3, 4, 5}_{o/p}$

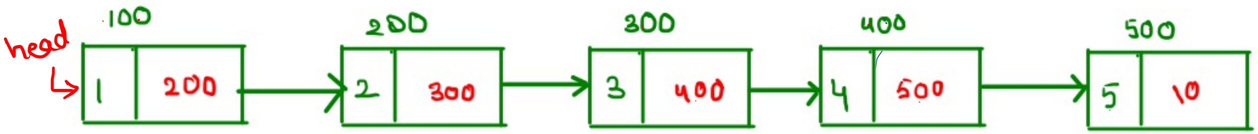
2) Print the elements of SLL in reverse order, without reversing it.

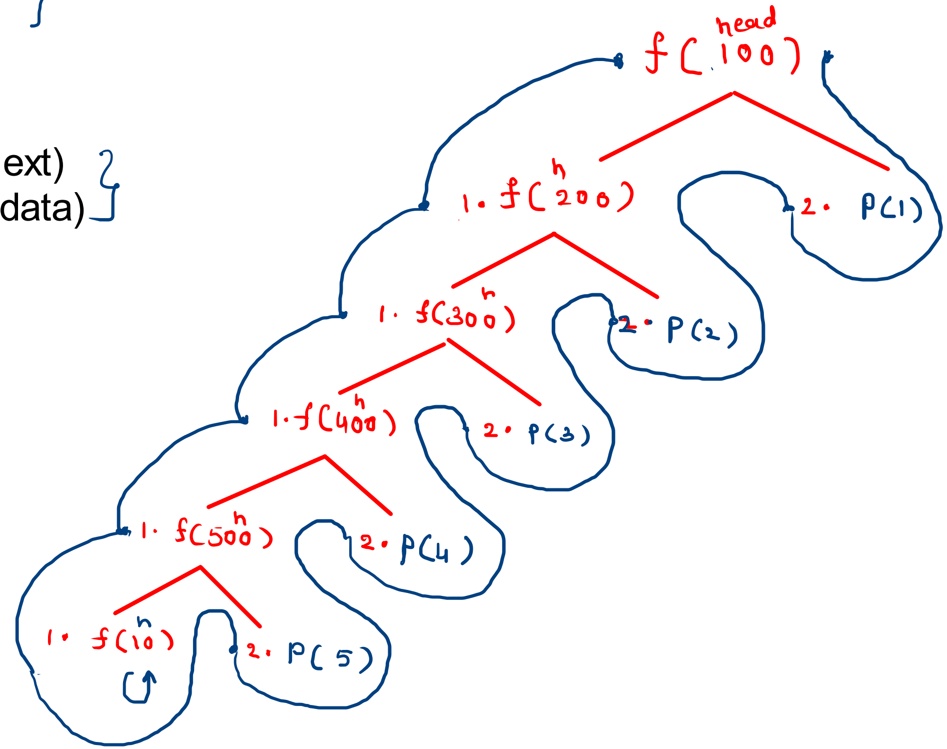
o|p 5, 4, 3, 2, 1

```
fun(Node head)
{
    if(head==null)
        return
    else
    {
      1. fun(head.next)
      2. print(head.data)
    }
}
```

~~200~~ ~~100~~ ~~300~~ ~~400~~ ~~500~~ 10

h

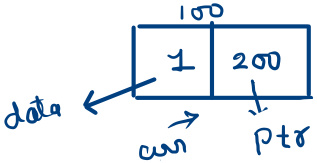




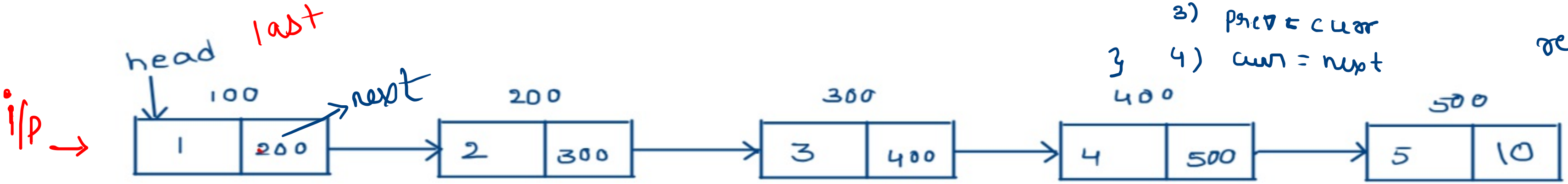
o|p

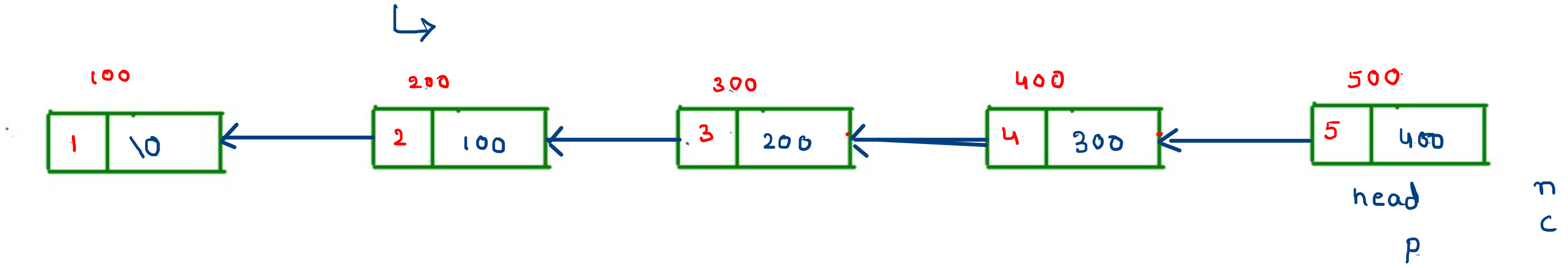
5, 4, 3, 2, 1

** Reverse the SLL



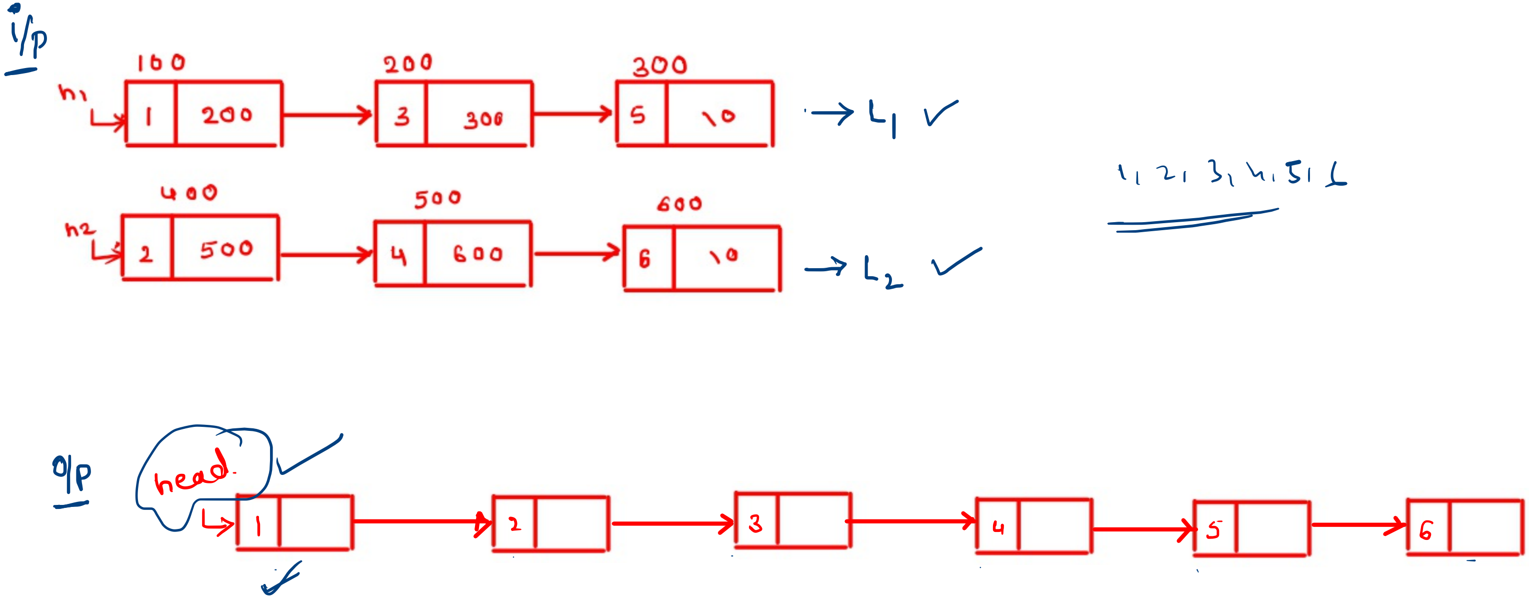
```
Prev = null
curr = head
next = null

x while ( curr != null )
{
    1) next = curr. ptr
    2) curr. ptr = prev
    3) prev = curr
    4) curr = next
}
. head=prev
return head
```





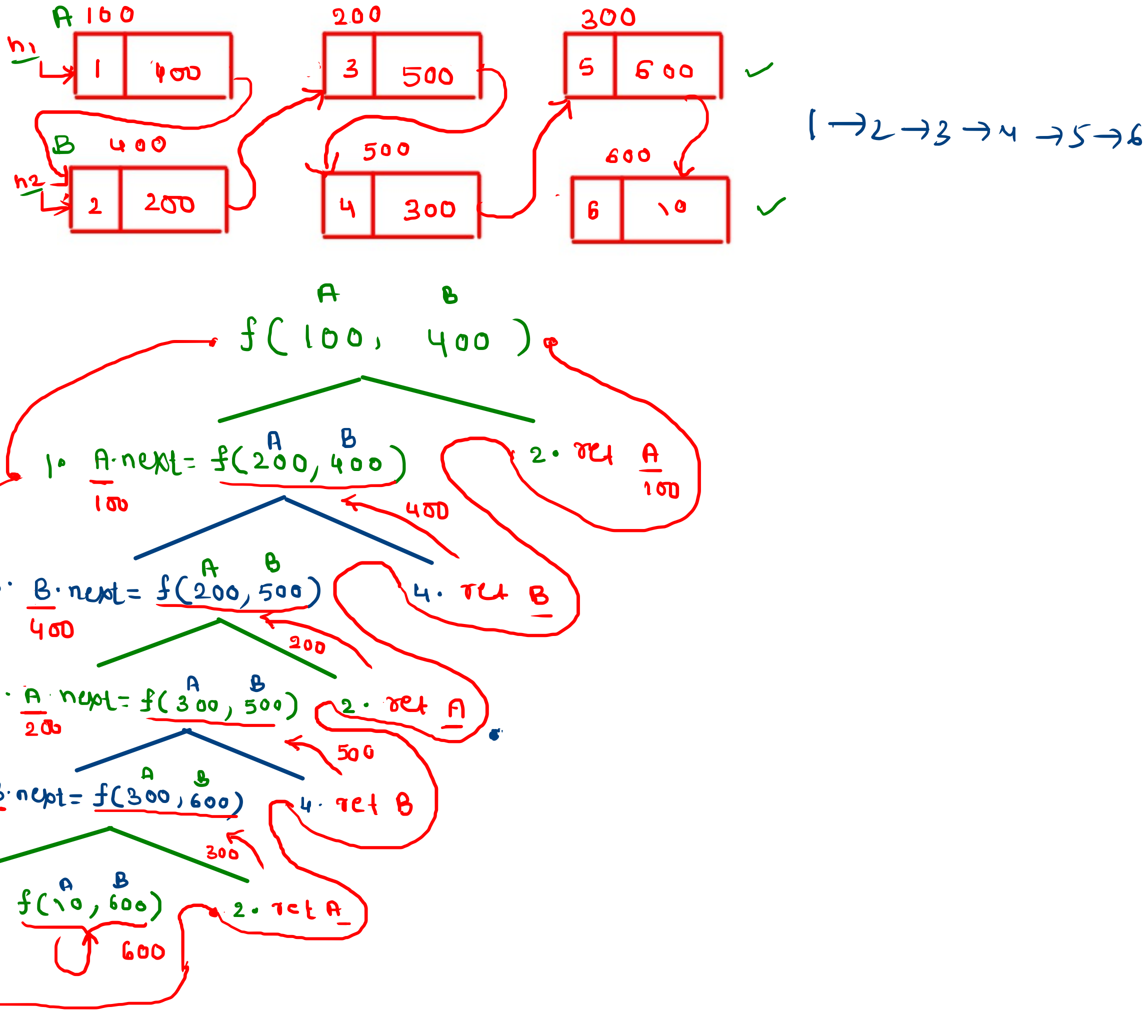
5, 4, 3, 2, 1,

* Merge two sorted SLL into a Single list



Merge two sorted SLL into a Single list

```
         10       600
Node fun(Node A, Node B)
{
    if(A==null) return B
    if(B==null) return A

    if(A.data<B.data)
    {
      1 A.next=fun(A.next,B)
      2 return A
    }
    else
    {
      3 B.next=fun(A,B.next)
      4 return B
    }
}
```



A 100
200
300
h1
1
400
3
500
5
600
B 400
500
600
h2
2
200
4
300
6
10
1→2→3→4→5→6
A B
f(100, 400)
1. A.next= f(200, 400)
100
400
2. ret A
100
3. B.next= f(200, 500)
400
4. ret B
200
1. A.next= f(300, 500)
200
2. ret A
500
3. B.next= f(300, 600)
4. ret B
300
1. A.next= f(10, 600)
600
2. ret A