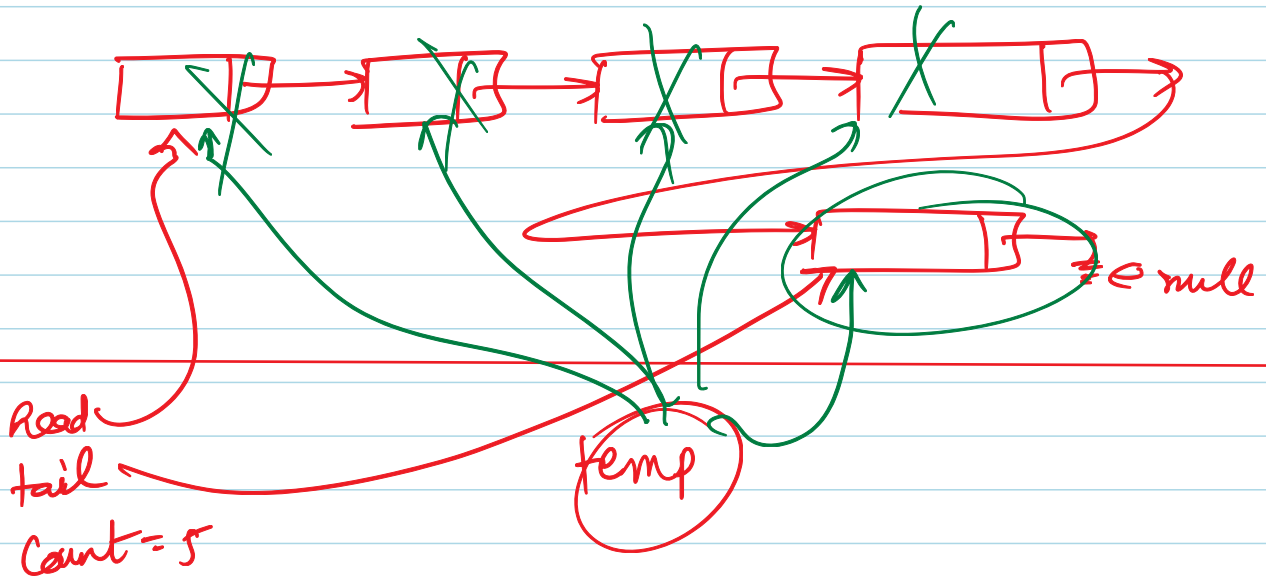
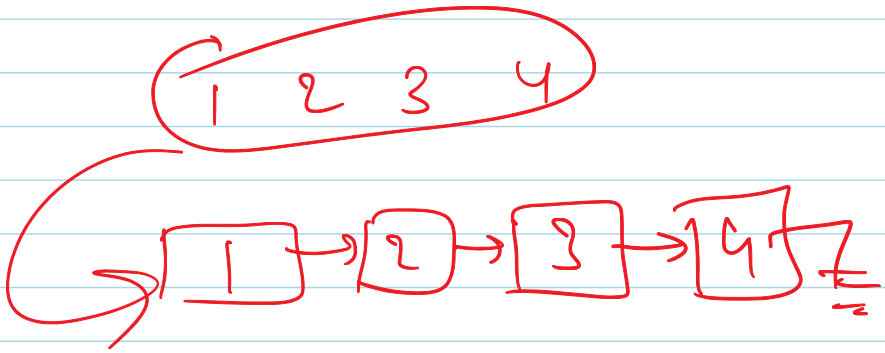
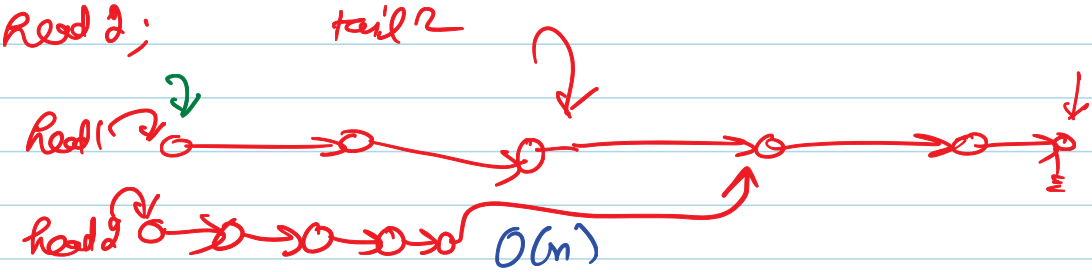
Heap -





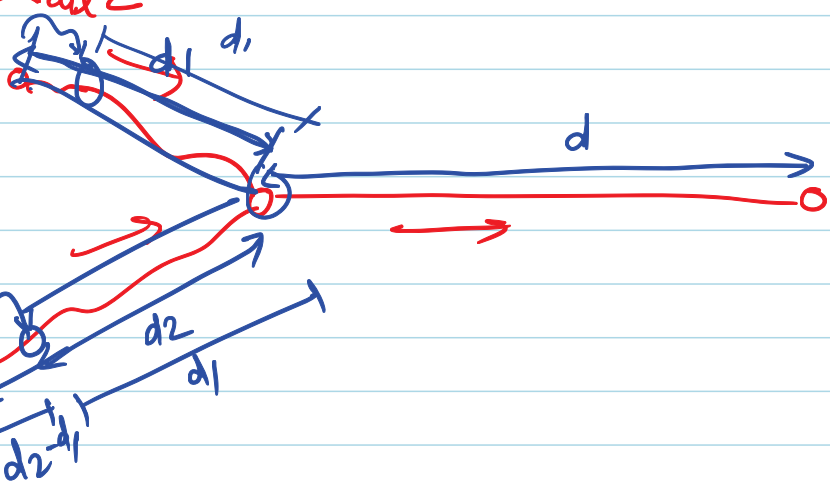
ListNode* Head1 ; ← tail1
→ Head2; tail2



it is guaranteed

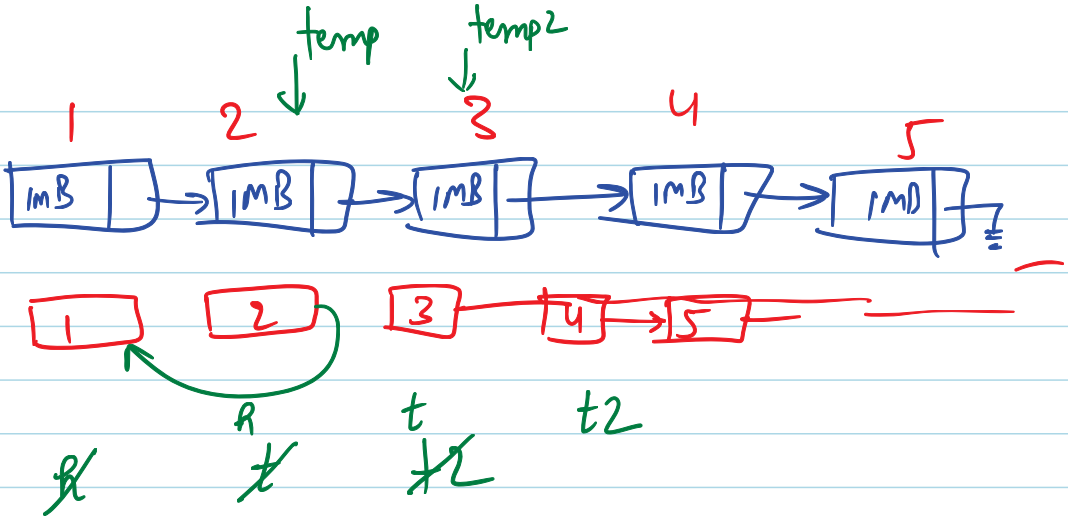
Head1 != null
Head2 != null
tail1 = tail2

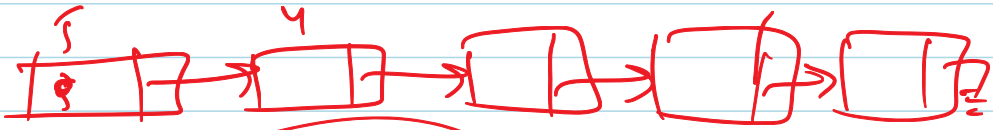
O(1) space



d2 > d1

d2 < d1

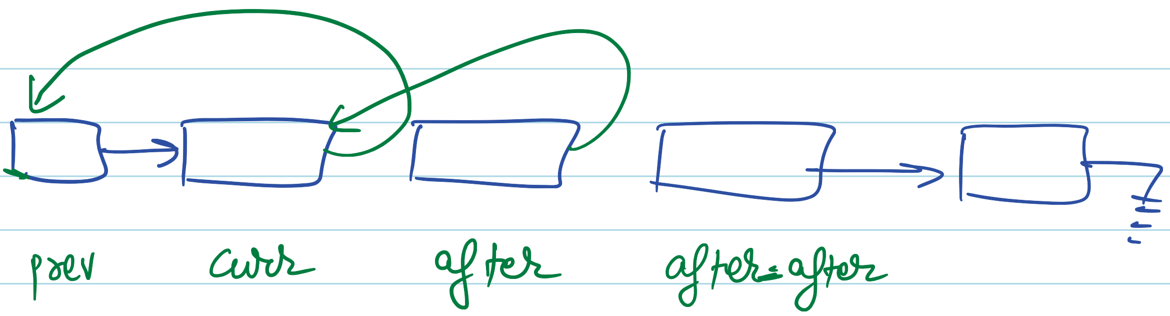


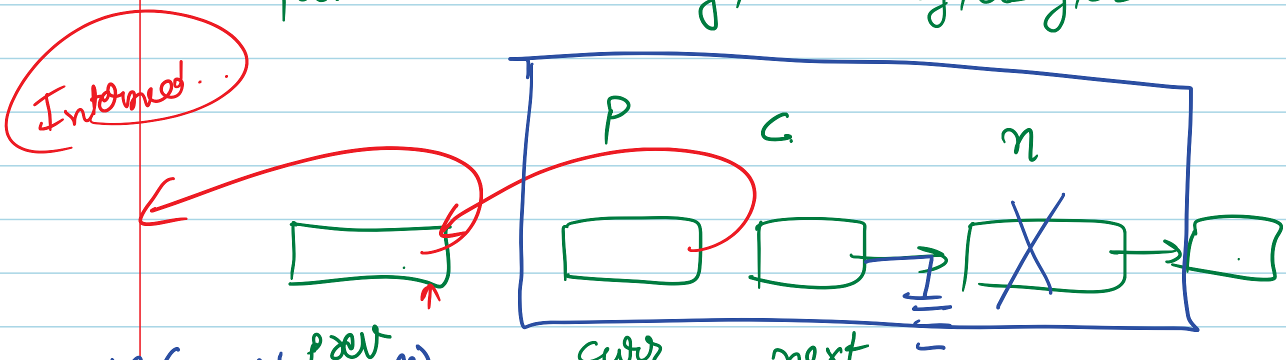


O(1) additional space
O(n) ⟶ time

```
Void Reverse ( Node* Head)
{
    // Initialize

    // Iterate

    // Terminate

}
```

```
// Base Case
if (!Head || !Head->next)
    return Head;
if (!Head->next->next)
{
    Head->next->next =
                    Head;
    auto temp = Head->next
    Head->next = nullptr;
    return temp;
}
```





while(next != null)
{
1. Curr->next = prev
2. prev = curr
3. Curr = next
4. next = next->next

}

1. next != null
2. curr != next
3. p != null
4. next->next != null

Curr->next = prev;
return curr;

Theory

Node ↓ (16 bytes)

8 byte

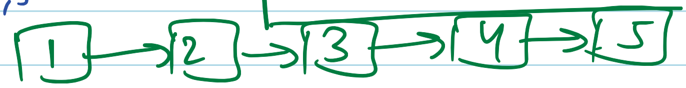
O(1)

n items

24n

arbitrary no. of insertion ✓

Time Space
O(n) O(1) 1. First -> end -> 1,2,3,4,5
O(n) O(1) 2. ~~Firs~~ 5,4,3,2,1



long long int

a

16n + C space

c < 32 bytes

exact ↓

Heap -> 16n

Stack -> c bytes

c < 64 bytes

Insertion at end → O(1) time
Iteration from start to end → O(n) time, O(1) space
end to start → O(n) time, O(1) space

16 n in heap
Exact → 16 n
O(1) in stack

```
class Arr list
{
private:
    -
    -
    -
    -
    -
public:
    List() : {   }
    void insert(int64_t data)
    void PrintForward() {   }
    void PrintBackward() {   }
}
```

O(1) space (private members)
O(1) time ← insert
O(n) time ← PrintForward
O(n) time ← PrintBackward