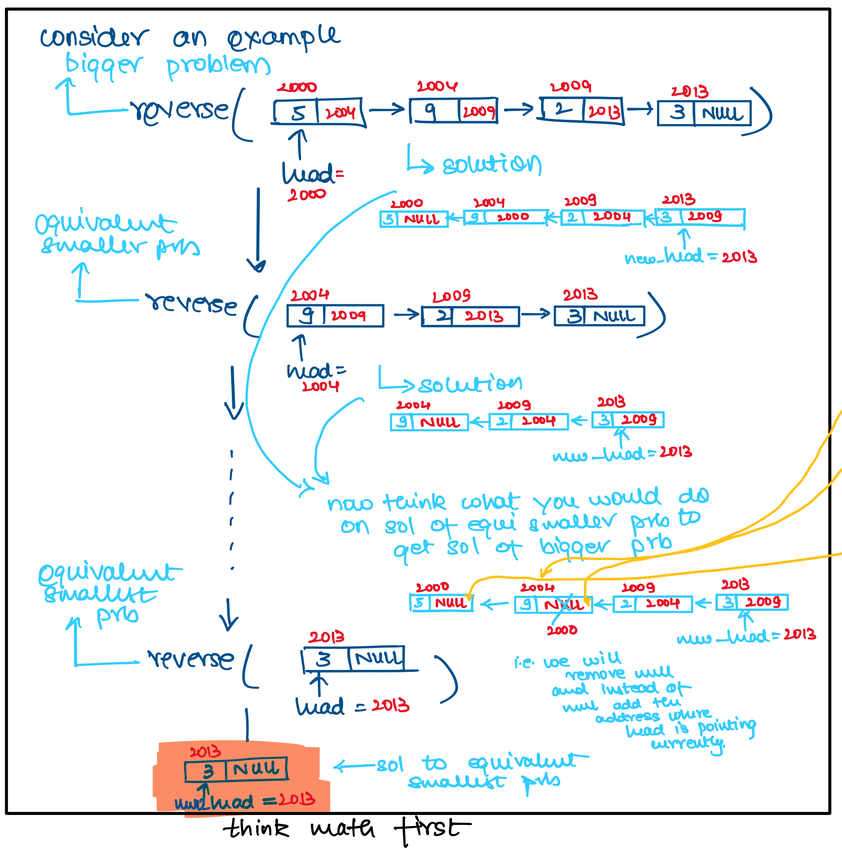
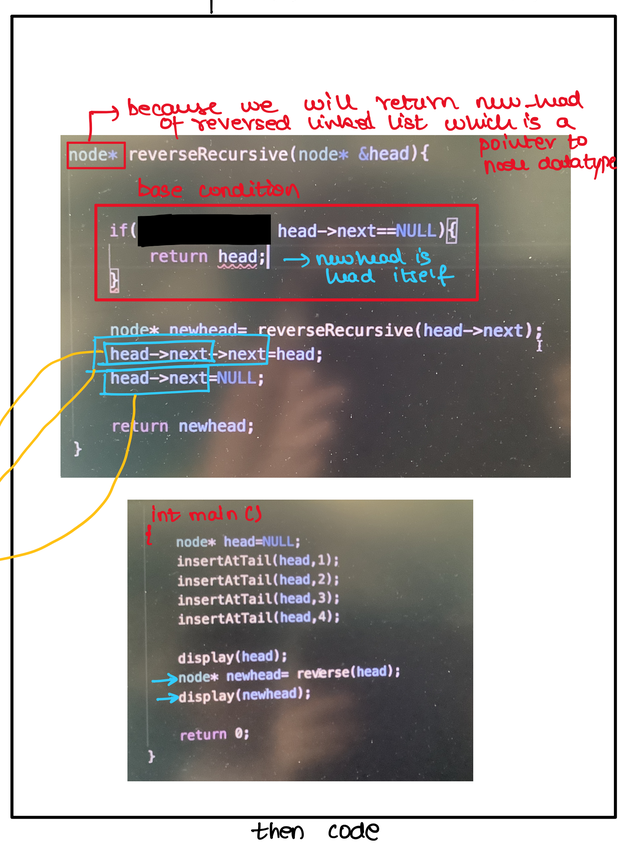
Recursion

## (Q) Reverse a Linked List (V.V.Imp)

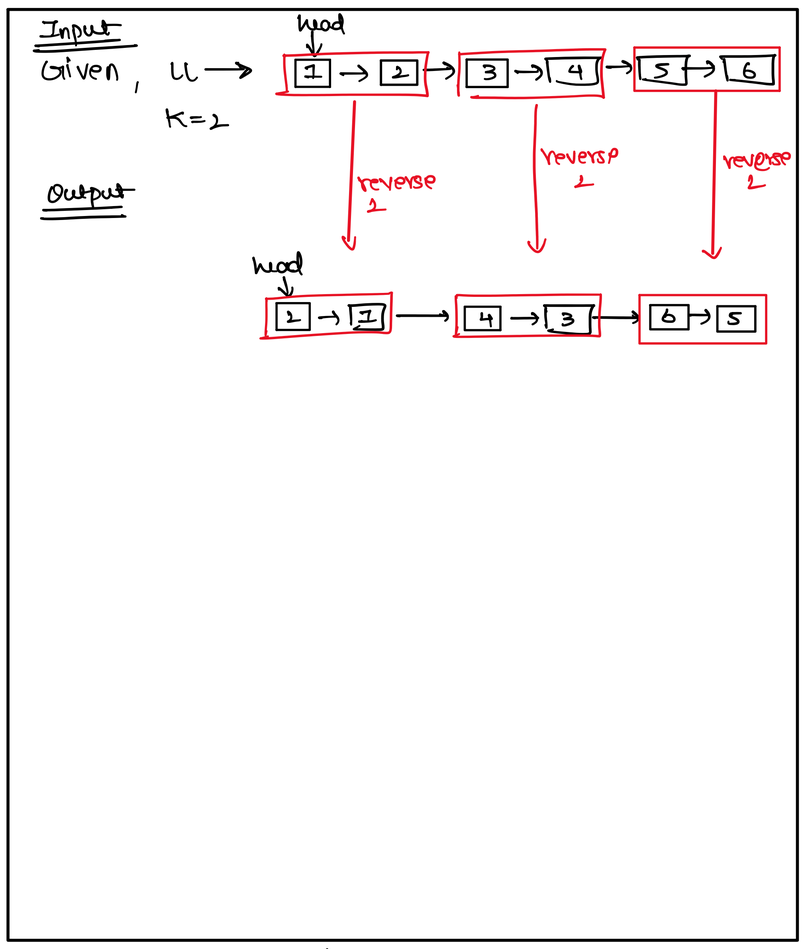
→ O(n) time complexity, O(n) space comp (recursive stack)

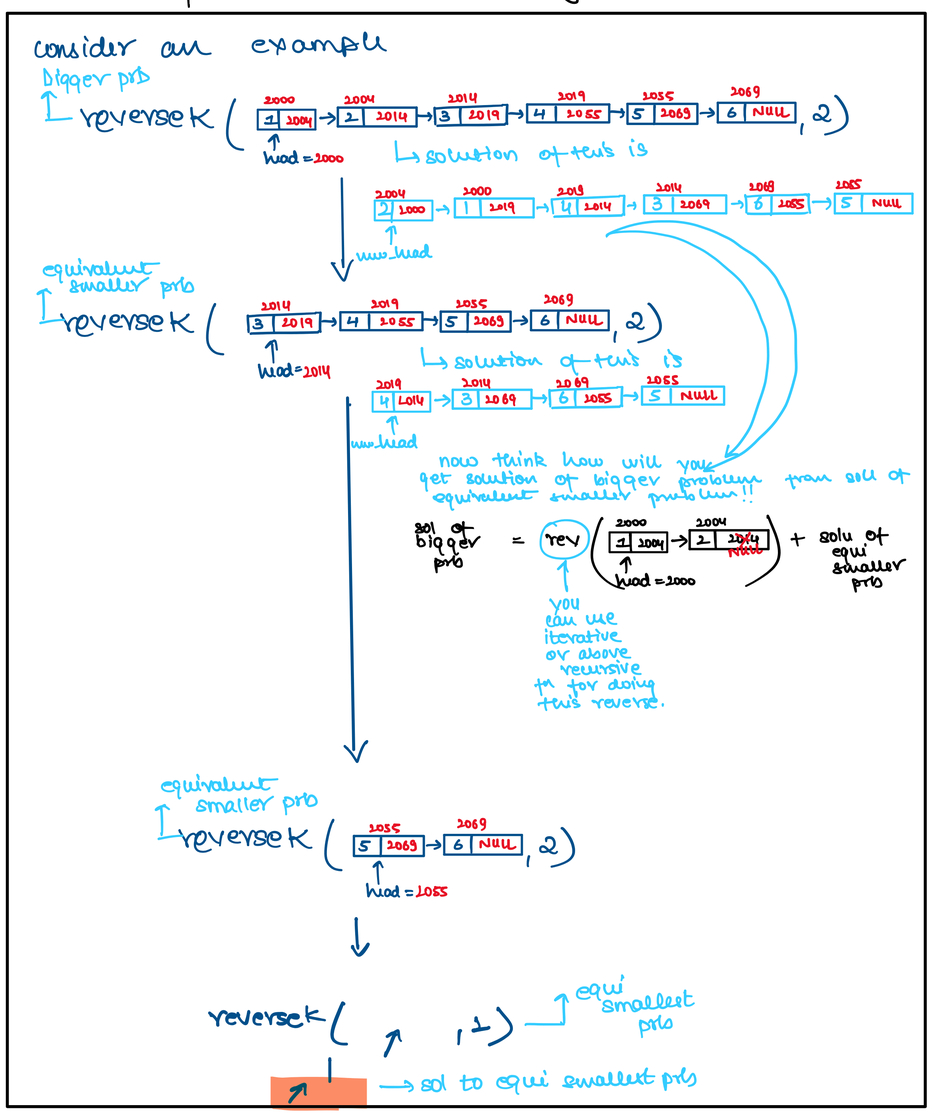


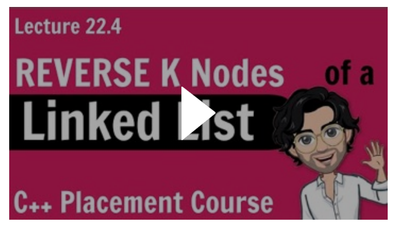

## (Q) Reverse k nodes in a Linked List (V.V.V.Imp)

→ O(n) time complexity




[Reverse K Nodes in a Linked List | C++ Placement Course | Lecture 22.4](#)




then code

## (Q) Sort a Linked List using Merge Sort

similarly logic, what we saw in array, usually interviewers never ask this question.

all sorting techniques we saw in array can be applied here too

1. Bubble Sort O(n²)
2. Insertion Sort O(n²)
3. Selection Sort O(n²)
4. Heap sort O(nlogn)
5. Merge sort O(nlogn)
6. Quick Sort O(nlogn)
7. Shell sort O(n^3/2)

→ comparision based sort algos

8. Count Sort O(n)
9. Bucket/Bin Sort O(n)
10. Radix Sort O(n)

→ index based sort algos.

→ why quick sort is prefereed for arrays over linked list??

- Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires O(N) extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm.
- Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.
- Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.
- Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.
- Quick Sort is also tail recursive, therefore tail call optimizations is done.

→ why merge sort is preferred for linked list over arrays??

- In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory.
- Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time if we are given reference/pointer to the previous node. Therefore merge operation of merge sort can be implemented without extra space for linked lists.
- In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at (x + i*4). Unlike arrays, we can not do random access in linked list.
- Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

- Insertion sort is more friendly to linkedlist compared to array cause in linkedlist to insert an element we need not shift element, here we just need to attach a node.
- Similarly merge sort is also more friendly to linked list compared to array.

think math first