$N1 \to N_2 \to N_3 \to \cdots \to N_{10086} \cdots \to NULL$

Physical Memory: (byte)

$0x\ FFFF0001$ $0x\ FFFF0002$ $0x\ FFFF0003$ $0x\ FFFF0004$

! make sure do not deference null pointer

1. How to reverse a linked list

prev head next

$N_1 \to N_2 \to N_3 \to \cdots N_n \to null$

$N_1 \leftarrow N_2 \leftarrow N_3 \leftarrow \_ \; N_n$

head

① Iterative:

```
next = head.next;
head.next = prev;
prev = head;          return prev
head = next.
```

② Recursive. (subproblem).

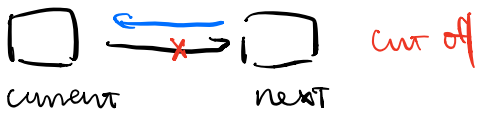
head

$N_1 \to$ [$N_2 \to N_3 \to \cdots \to N_n \to null$] before

$null \leftarrow N_1 \leftarrow$ [$N_2 \leftarrow N_3 \leftarrow \cdots \leftarrow N_n$] after

current's next's next = current.

current's next = null;



2. How to find the middle node of a linked list.

Fast & Slow pointer.

└ increment 2 times speed as slow pointer.

when fast reaches the end, slow will be at the middle.

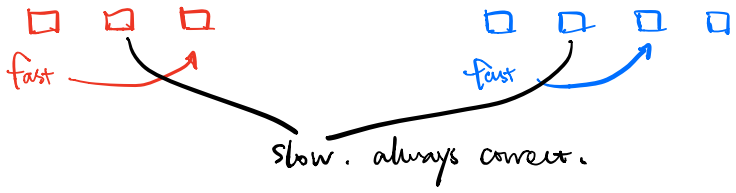
while loop condition: fast has next

fast's next has next.

So when we jump out of the loop, fast will be either the last node or the second last node

n = odd number

n = even number

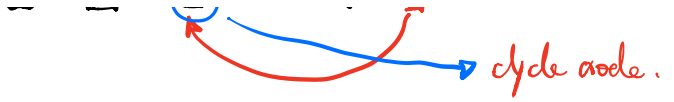


3. Check if a linked list has a cycle. (no duplicate val)

Fast & slow pointer.

if cycle, fast & slow will eventually have the same value.

Extra: Cycle Node in Linked List.

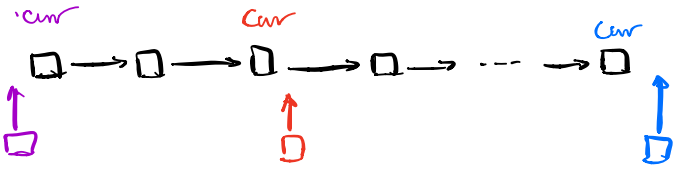


HashSet to check.

add visited node (next) to map.

If found contains - cycle node (where cycle starts)

4. Insert a node in a Sorted linked list.



case # 1: insert midde : cur < target ≤ cur.next.

case # 2: insert end : cur < target.

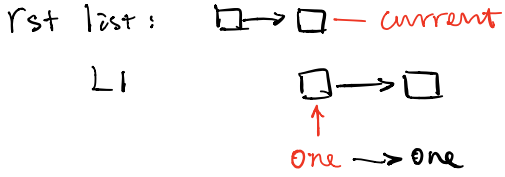
Case # 3 : insert head : cur ≥ target

J. Merge two sorted linkedlist into a longer Linked list.

use two pointers & move the smaller pointer.

use dummy head and return dummy's next.

Note : when append to current.



current.next = one.

one = one.next;

Then: cur = cur.next

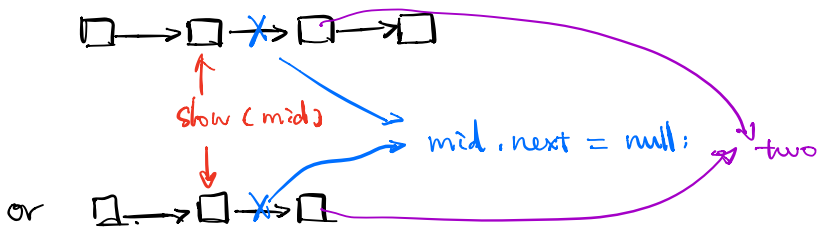
6. Reorder Linked List.

$N_1 \to N_2 \to N_3 \to N_4 \to \cdots \to N_n \to null$

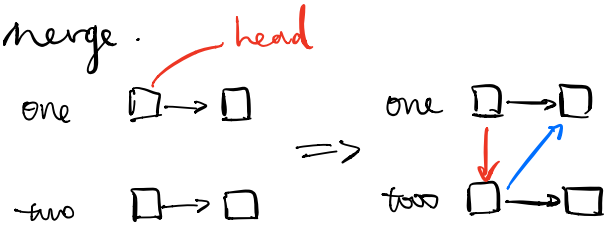
$\Rightarrow (N_1 \to N_n) \to (N_2 \to N_{n-1}) \to (N_3 - N_{n-2}) \to \cdots$

Step 1: Find the middle and split to 2 lists.
Step 2: Reverse the 2nd list.
Step 3: Merge 2 list.

Note: after find mid. need to cut off.



Note: merge.



one.next = two.

two.next = one's next.

update one & two

7. Partition linked list.

Given a list & target x.

partition it s.t all nodes less than x are listed before nodes larger than or equal to x.

(Keep the original relative order).

Step 1: create two dummys for small & Large values

Step 2: iterate over the list. & compare values. add to small & large tails.

Step 3: Concat small & large.

small Tail's next = largeDummy's next.

Step 4: Terminate cycle: !

largeTail.next = null;

8. Merge Sort Linked list.

Step 1: find Middle (slow & fast pointers) & split to two halves.

└ mid.next = null;

Step 2: Recursion

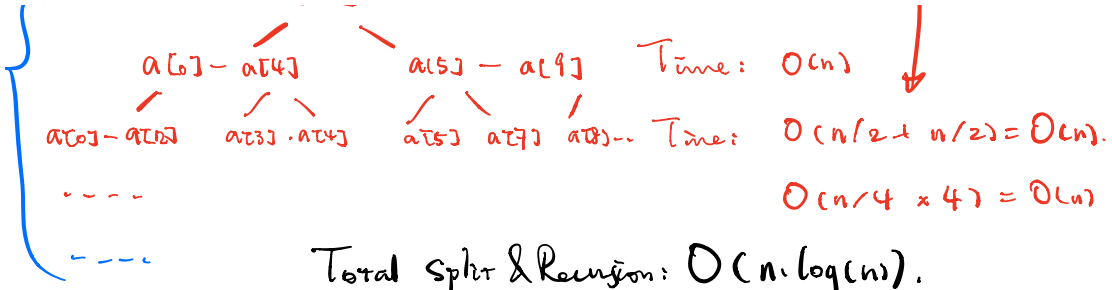
Step 3: Merge 2 halves.

Analysis:

Split part: find mid = O(n) for each layer.

Total: (log n) layers.

A[10]



Total Split & Recursion: $O(n \cdot \log(n))$.

Merge Part: $O(n)$ for each layer.

$O(\log n)$ layers.

Total Merge $O(n \cdot \log(n))$.

Total Time: $O(n \log n + n \log n) = O(n \log n)$.

Total Space: all nodes in a path: $O(\log n)$.

9. Add Two numbers:

Given 2 non-negative #s. digits stored in reverse order add them up & return as a linked list.

&10. Remove Nodes. based on target value