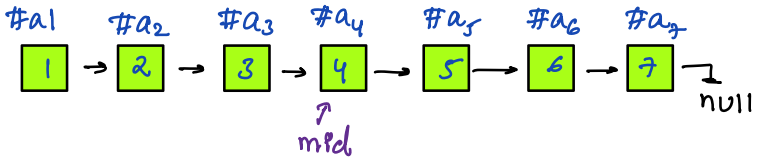
Todays Content:

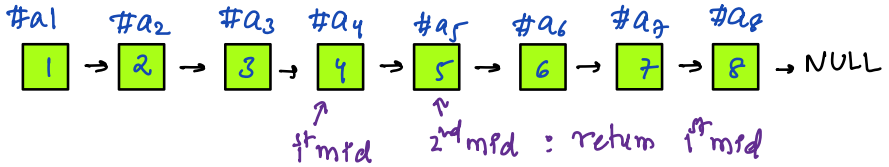
a) Merge
b) Merge sort
c) Re-arrange Linked list
d) Cycle detection
    i) Detect cycle
    ii) Find start of cycle
    iii) Remove cycle
e) Find Intersection

10) Given head node find mid of linkedlist

Ex1:



Ex2:



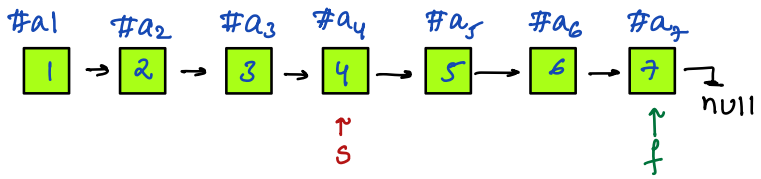
Idea: Iterate & get length, & goto to center node

TC: O(N) SC: O(1), Note: Traversing 2 times on linkedlist

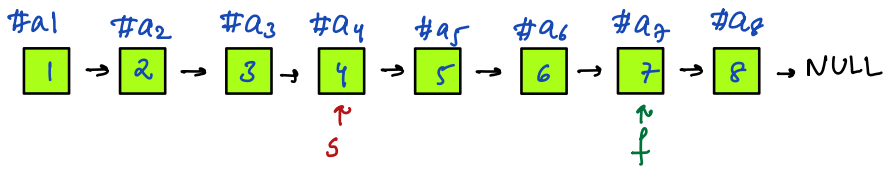
Idea2: Usage of slow and fast pointer

slow = slow.next

fast = fast.next.next

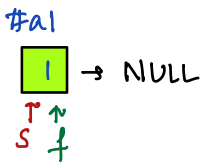


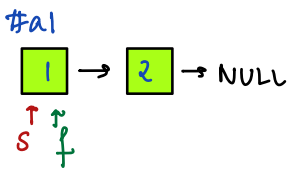
obs1: If f.next == null, s is at centre



obs2: If f.next.next == null, s is at centre

```
Node mid (Node h){ TC: O(N) SC: O(1)
    s = h, f = h
    if(h == NULL) { return h;
    while( f.next != NULL && f.next.next != NULL){
        s = s.next
        f = f.next.next
    }
    return s;
}
```

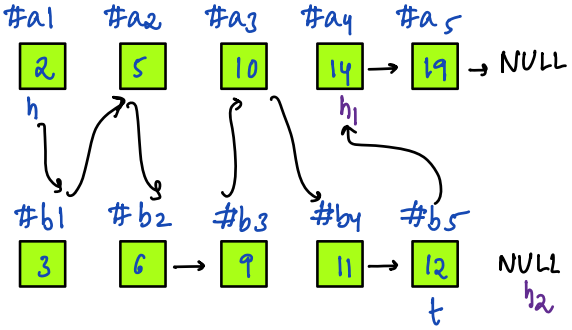




2Q) Given 2 sorted linked lists, Merge & get final sorted list

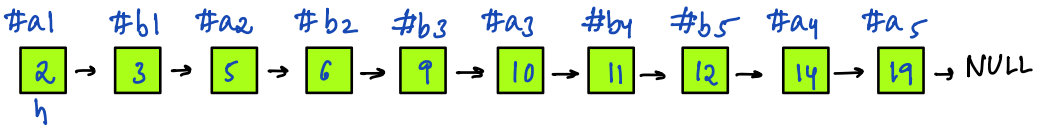
TC: O(N+M) SC: O(1)

Ex1:

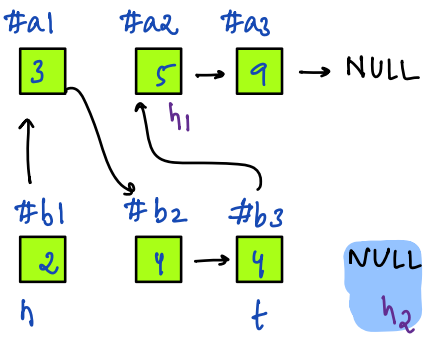


obs: if h_1 or h_2 == null stop comparison

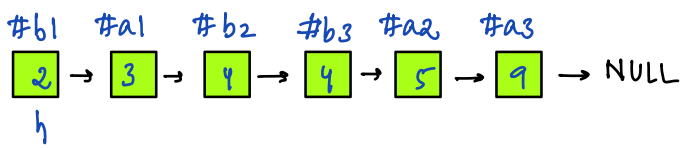
final:



Ex2:



final:

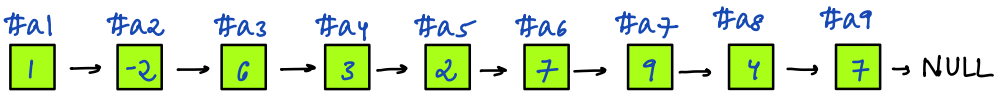


Psendo Code:

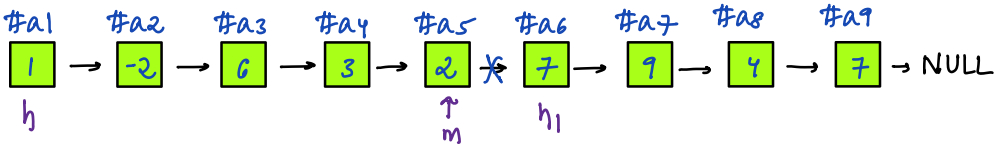
```
Node merge( Node h1, Node h2){ TC:O(N+M) SC:O(1)
                                 // assuming both linkedlists have diff length
    Node h = null, t = null
    if(h1 == null) {return h2}
    if(h2 == null) {return h1}
    if( h1.data < h2.data){ // h1 should be head node
        h = h1, t = h1, h1 = h1.next
    }
    else{ // h2 should be head node
        h = h2, t = h2, h2 = h2.next
    }
    while( h1 != NULL && h2 != NULL){
        if( h1.data < h2.data){
            t.next = h1,
            h1 = h1.next
            t = t.next
        }
        else{
            t.next = h2
            h2 = h2.next
            t = t.next
        }

    }
    // Add leftout part
    if(h1 != NULL) { t.next = h1}
    if(h2 != NULL) { t.next = h2}
    return h
}
```

38) Merge Sort on Linked list:

Eg1:

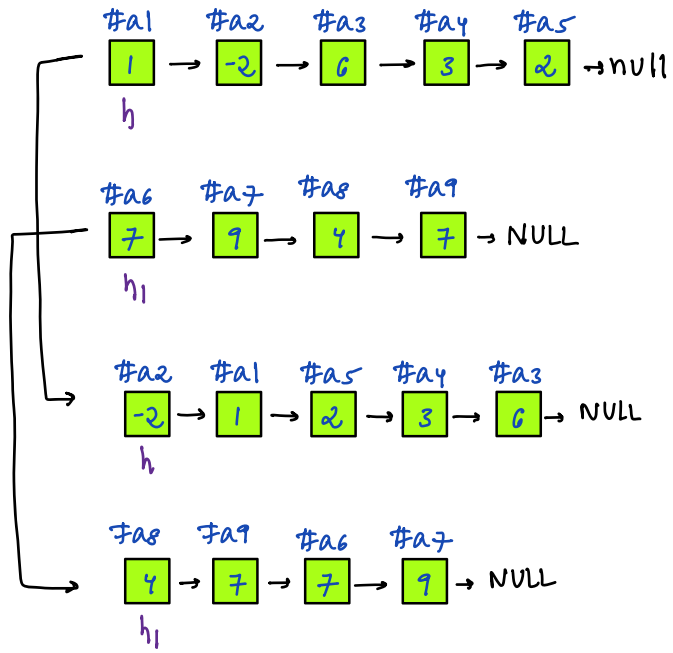


Step1: Calculate mid, & divide into 2 separate linked lists

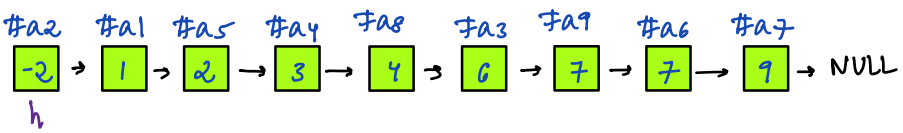


$h_1 = m.next$, $m.next = null$

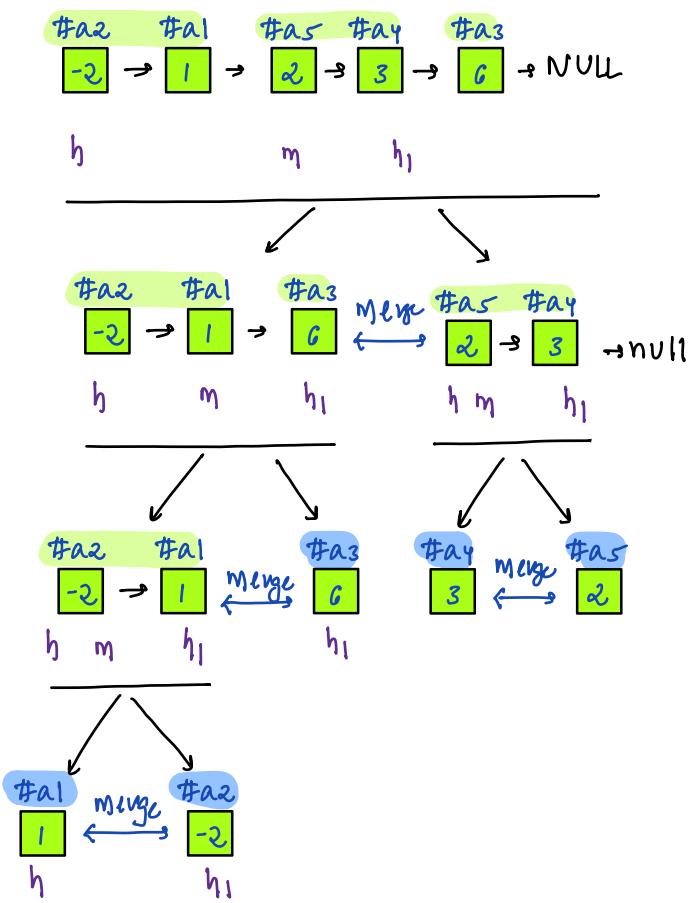
Step2: Sort both halves using merge sort



Step3: Merge 2 sorted linked lists

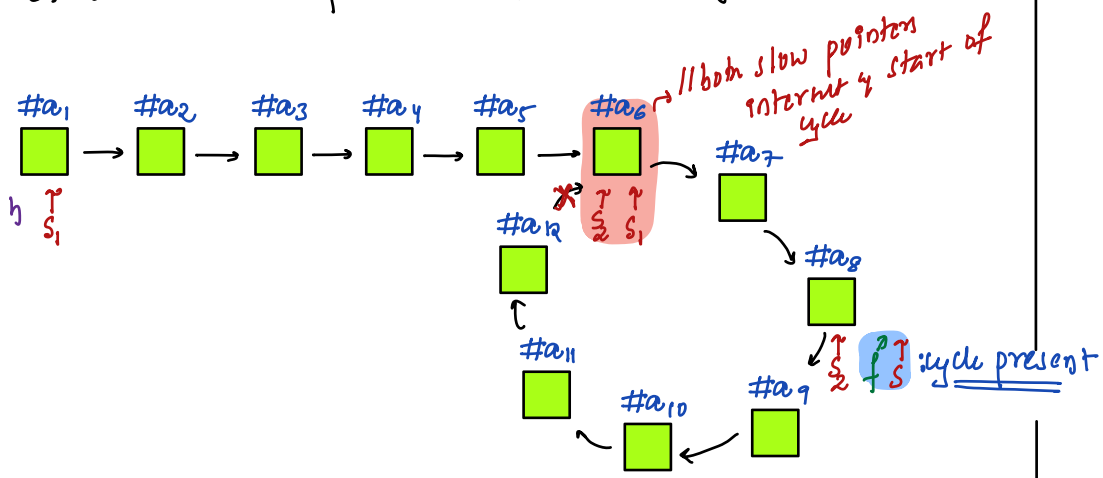


```
Node mergeSort(Node h){   TC: O(NlogN)  SC: O(1+logN) ≈ SC: O(logN)

    if(h==null || h.next==null){return h;}   // if only single element, already return it

    Node m = mid(h)
    Node h1 = m.next
    m.next = null                            // Step-1, breaking into 2 linked lists

    Node t1 = mergeSort(h)
    Node t2 = mergeSort(h1)                  // Step-2, sorting 2 linkedlist & storing head nodes in t1 & t2

    Node t3 = merge(t1,t2)                   // Step-3, merge 2 sorted lists & store head node in t3

    return t3
}
```

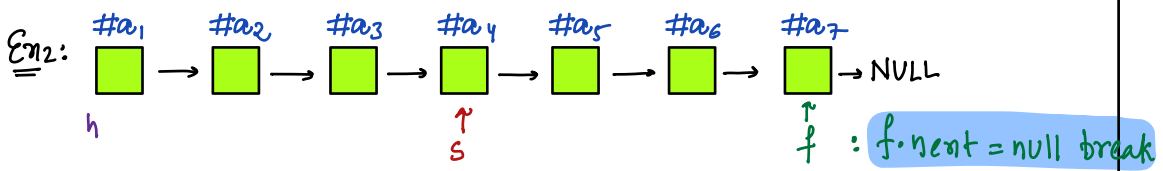


4Q) Given a head node of linkedlist, check for cycle detection?

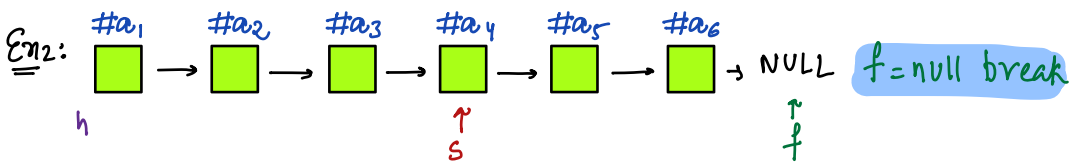
Ex1:



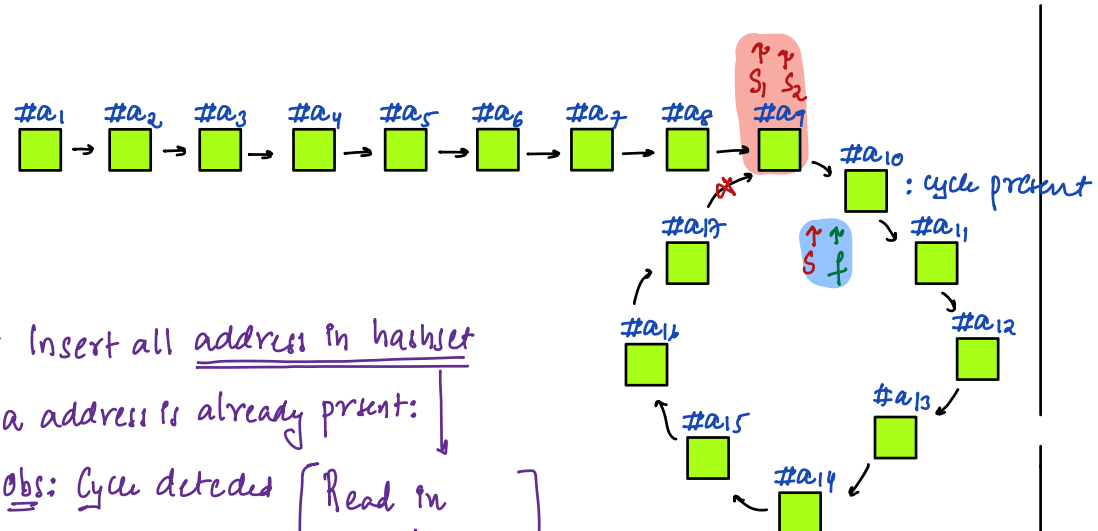
Ex2:



Ex2:



Ex3:



Idea: Insert all address in hashset

a) If a address is already present:

Obs: Cycle detected

b) If we reach null

Obs: No cycle

[Read in your language of choice TODO]

TC: O(N) SC: O(N)

obs: s = h, f = h, TC: O(N) SC: O(1)

s = s.next

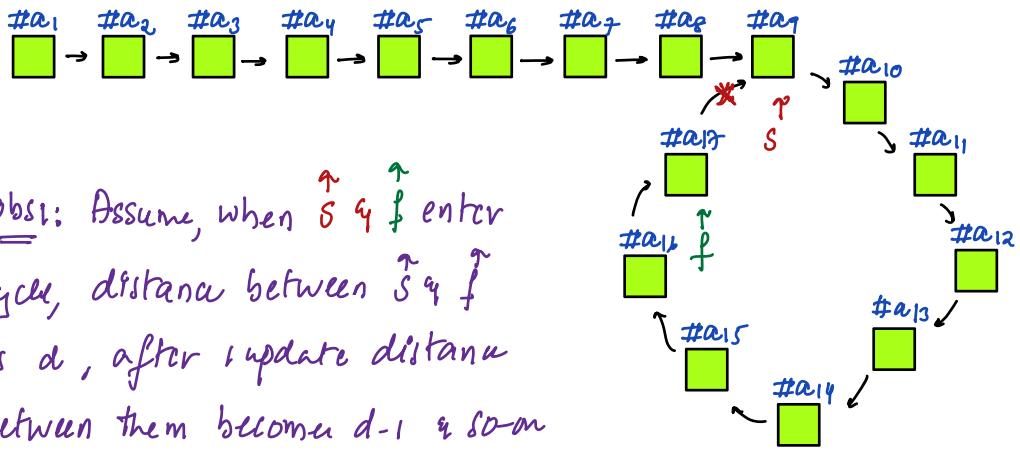
f = f.next.next

After few iterations

a) s == f : cycle present

b) f = null or f.next = null : cycle not present

Why will s == f if there is a cycle?



obs1: Assume, when s & f enter cycle, distance between s & f is d, after 1 update distance between them becomes d-1 & so-on

After d updates: d → d-1 → d-2 → d-3 → ... 0, match

Q) Given head node of linked list

a) Check if there exists a cycle

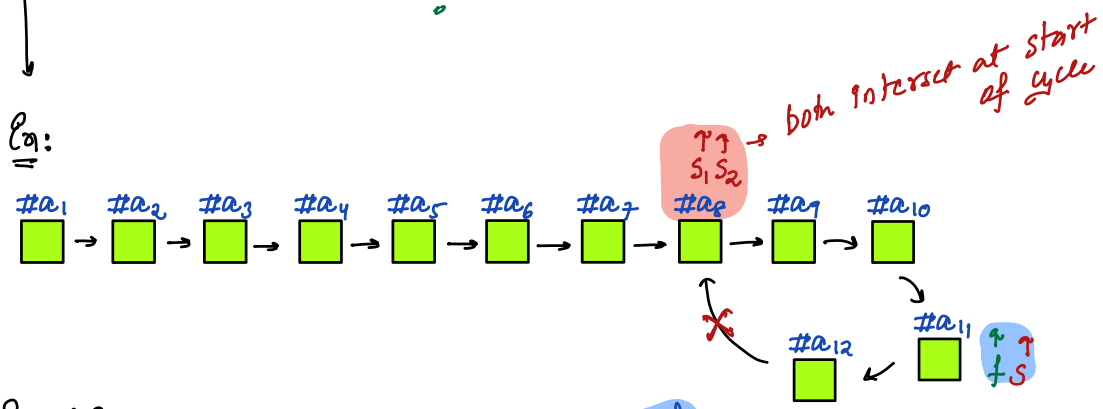
→ If no cycle return null ✓

b) If cycle :

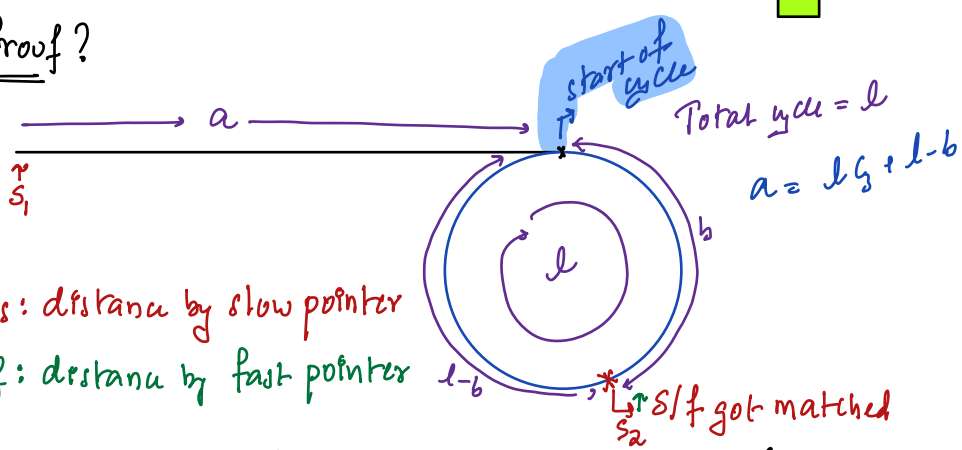
→ Return start of the cycle, after removing cycle ?

?

Eg:



Proof ?



d_s : distance by slow pointer

d_f : distance by fast pointer

$d_s = a + b + c_1 * l$

$d_f = a + b + c_2 * l$

obs: distance travelled by fast is twice as much as distance travelled by slow

$d_f = 2 d_s$

$a + b + c_2 * l = 2[a + b + c_1 * l]$

$a + b + c_2 * l = 2a + 2b + 2 c_1 l$

$c_2 l - 2 c_1 l = a + b$

$l[c_2 - 2c_1] = a + b$

$l[c_2 - 2c_1] = a + b$

$l[c_2 - 2c_1 - 1] + l = a + b$ → $\approx c_3$

$a = l[c_2 - 2c_1 - 1] + l - b$

$a = l c_3 + l - b$

Node removeCycle(Node h){ TC:O(N) SC:O(1)

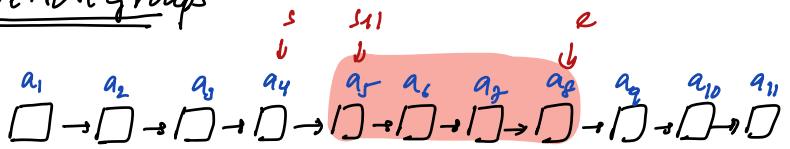
```
Node s = h, f = h
bool isCyl = false;
while( f != null && f.next != NULL){
    s = s.next
    f = f.next.next
    if (s == f){ // cycle present
        isCyl = true break;
    }
}
if ( isCyl == false){
    return null
}
Node s1 = h, s2 = f; prev = s2
while( s1 != s2 ){
    prev = s2 // updating prev node
    s1 = s1.next
    s2 = s2.next
}
prev.next = null // breaking cycle
return s1 // start of cycle is s1
```

// Detect cycle
TC: N

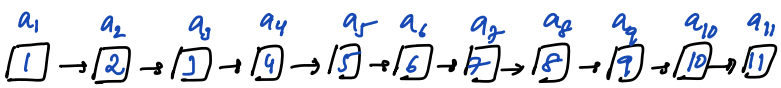
TC: N

}

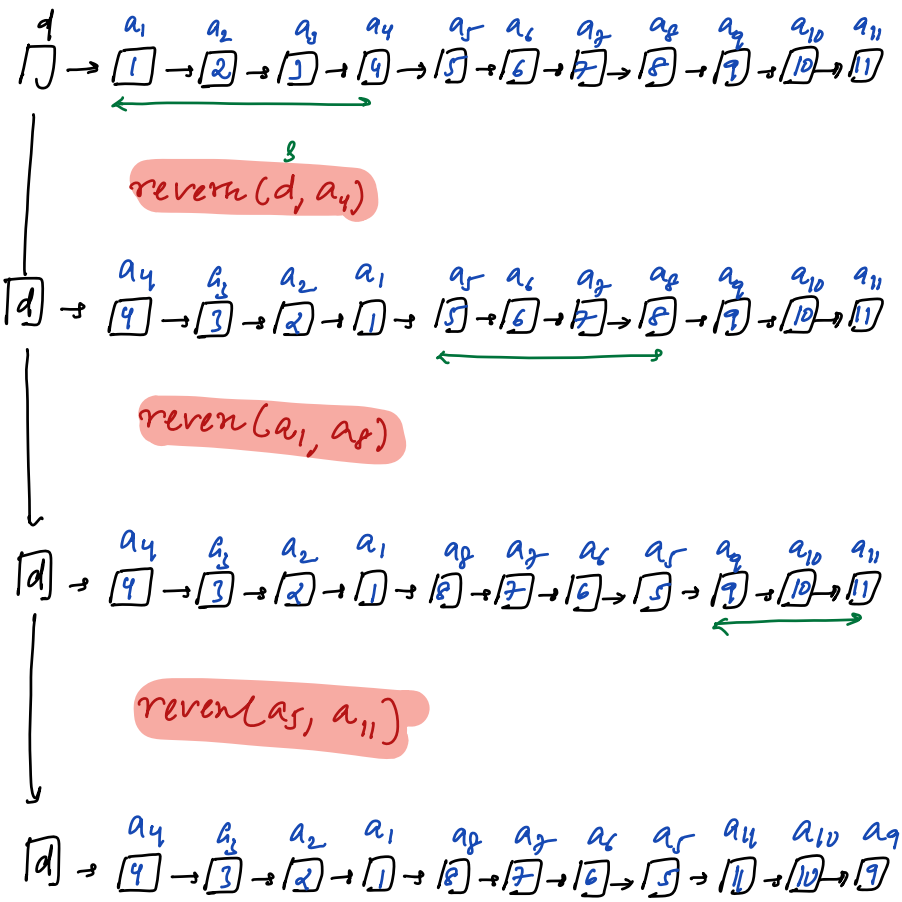
Reverse k groups




```
reverse(Node s, Node e){ TODO
    // reverse from [s.next -> e]
}
```



Step:1 Insert a dummy node at start, k = 4



```
void reverseInRange(Node s, Node e)
    // reverse all nodes from [s.next to e]
    // TODO
}

Node reverseKgroup(Node h, int k){
    Node d = new Node(-1);  // Insert dummy at front
    d.next = h, h = d
    Node s = d
    while(s.next != NULL){
        t = s.next
        e = s;
        int i = 1
        while(i<=k && e.next != NULL){
            e = e.next
        }
        reverseInRange(s, e)
    }  s = t
    return d.next
}
```