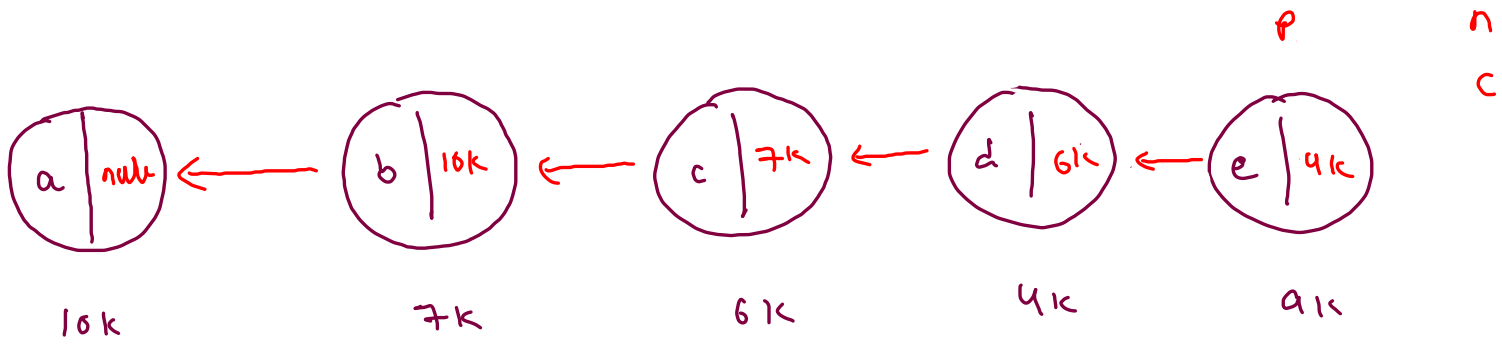
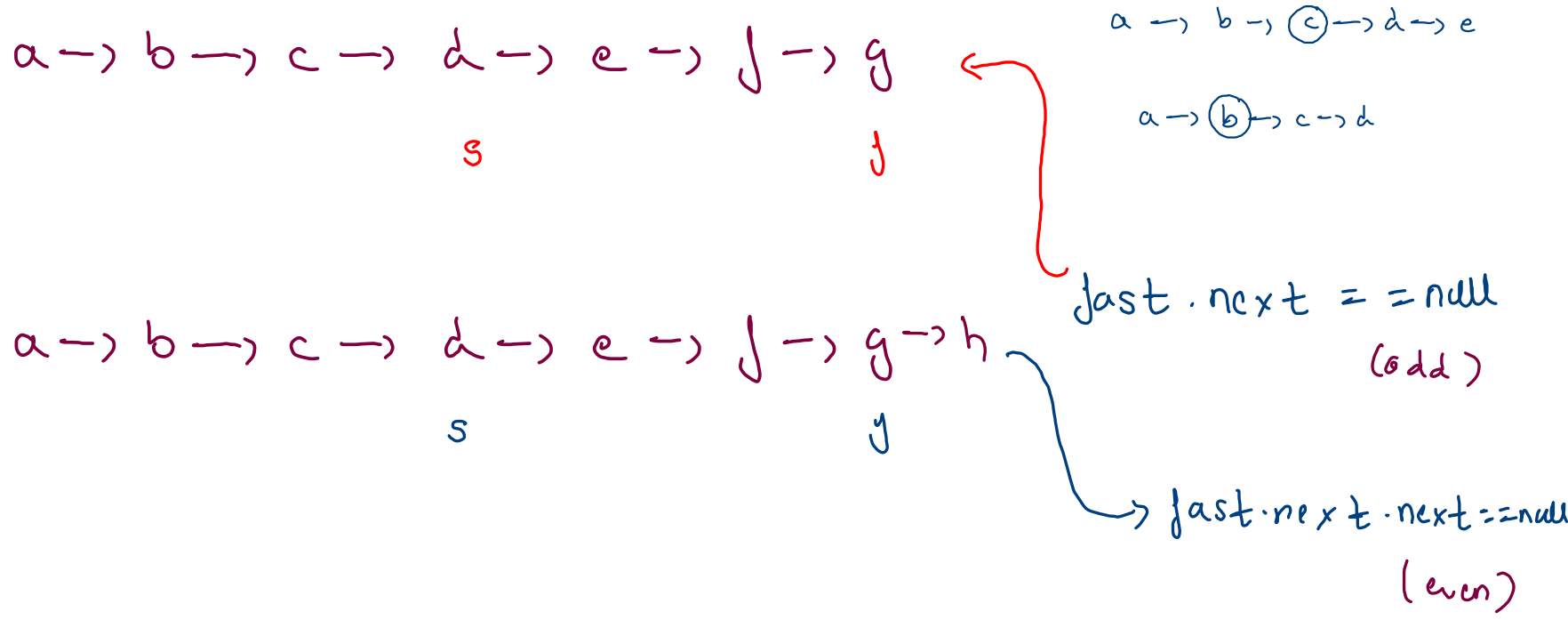
Reverse A Linkedlist

by pointer iterative




```
n = c.next; // backup
c.next = p; // connection
p = c;      // move
c = n;
return p;
```

Middle Of A Linked List



fast.next == null

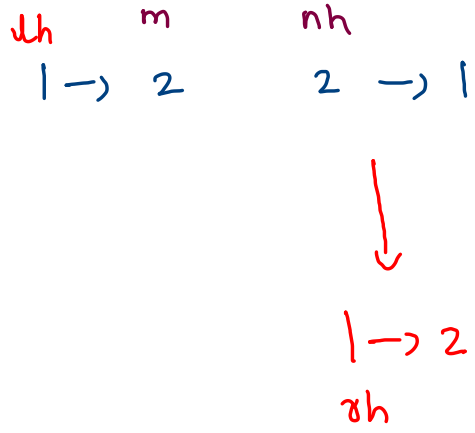
a -> b -> c -> d -> e -> f -> g -> h

s

f

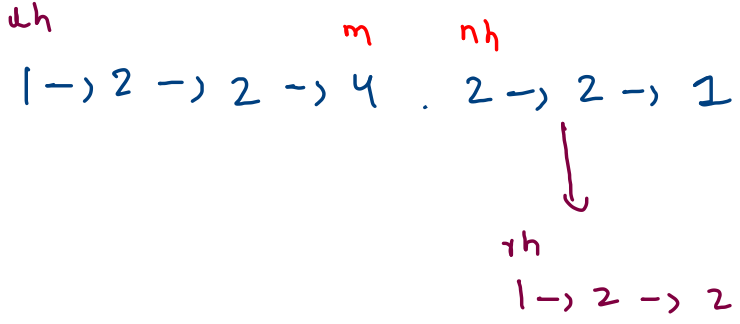
fast == null

Palindrome Linked List



T: O(n)

S: O(1)

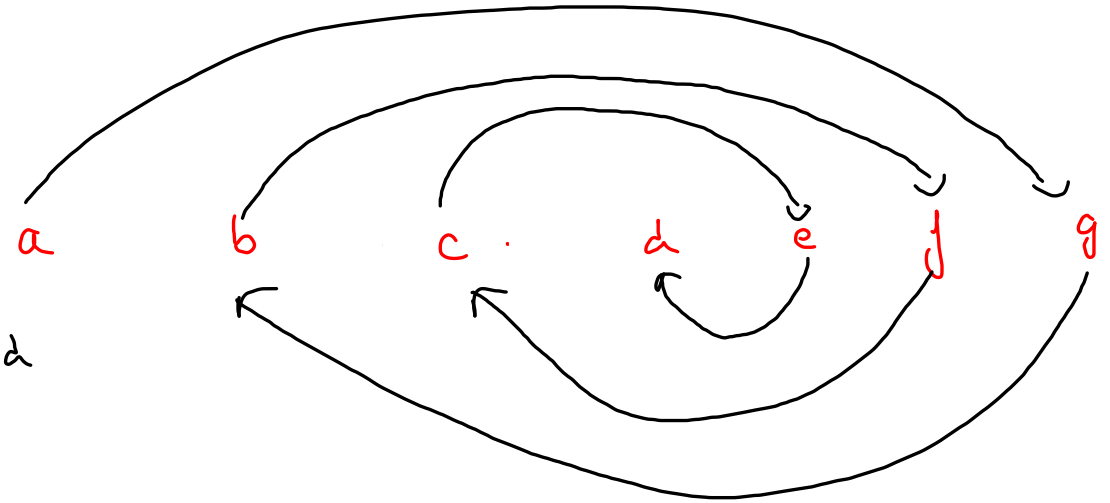


nh -> m.next

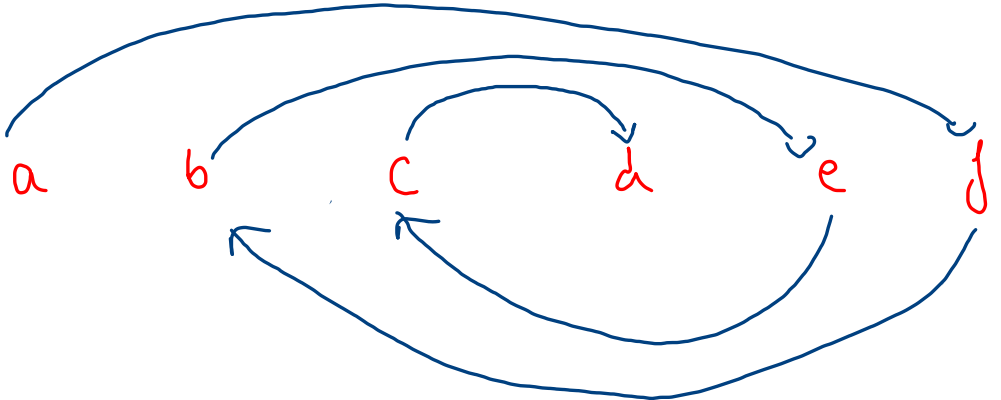
m.next = null

rh = rev(nh);

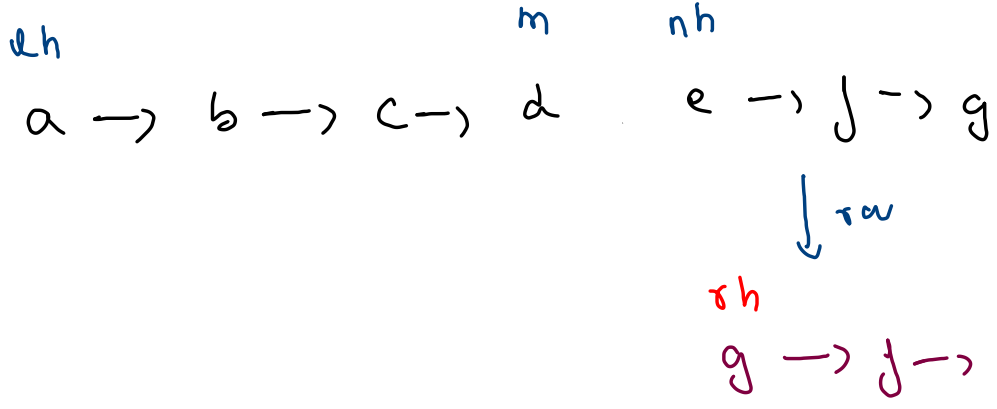
Reorder List (Fold)



a -> g -> b -> f -> c -> e -> d

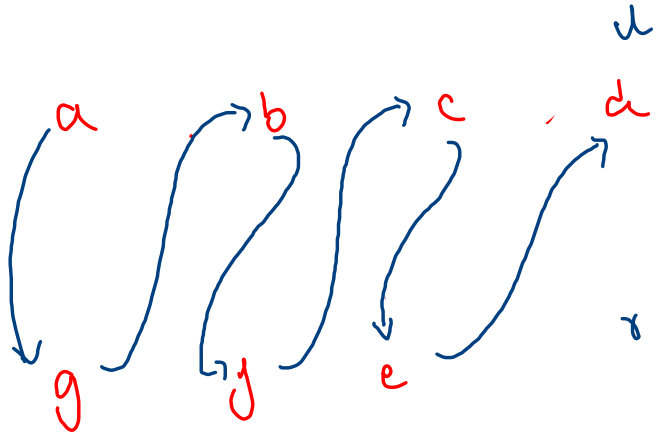


a -> f -> b -> e -> c -> d

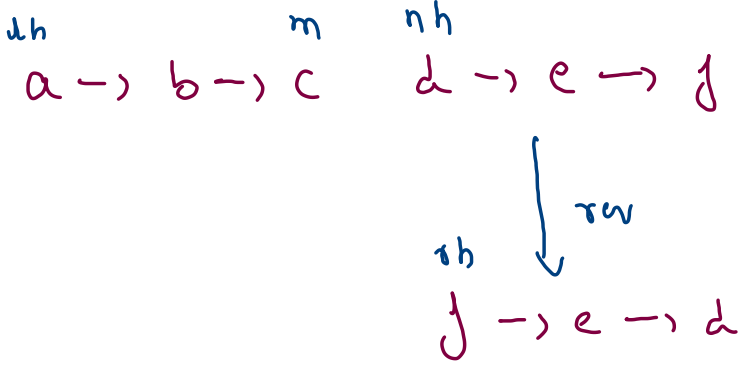


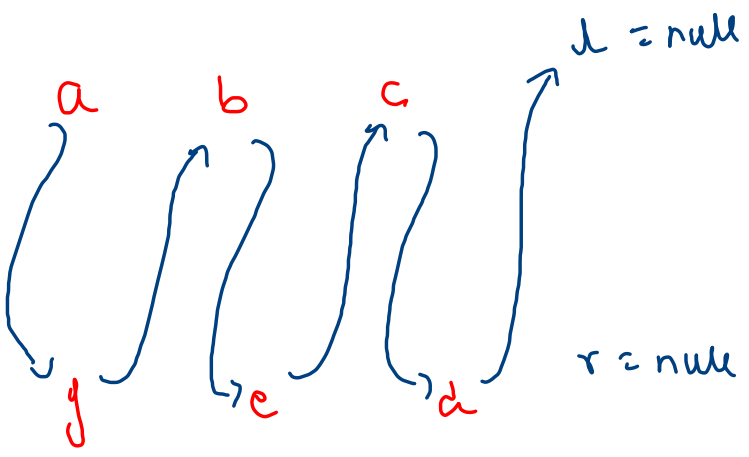
T : o(n)

S : o(1)




```
while (r != null)
{
    ln = l.next;
    rn = r.next;

    l.next = r;
    r.next = ln;

    l = ln;
    r = rn;
}
```



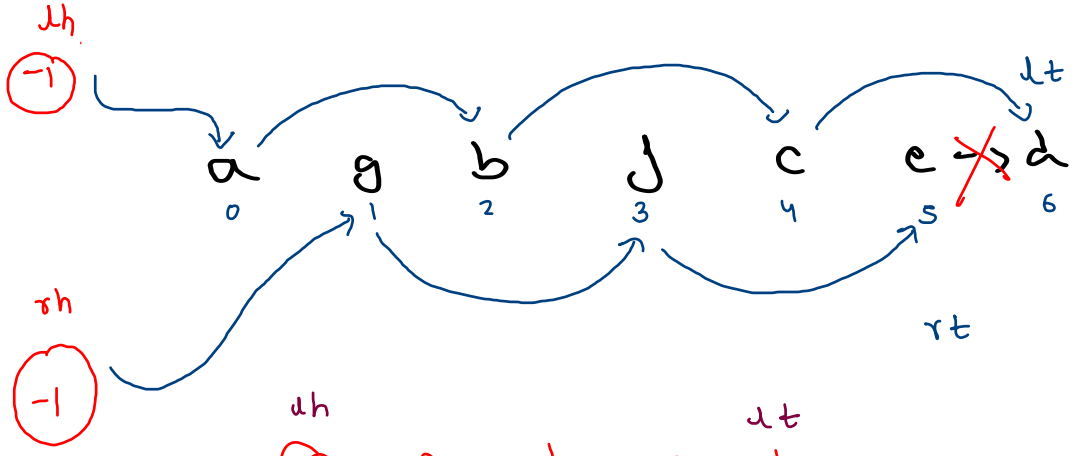



```
while (r != null)
    dn = d.next;
    rn = r.next;
    d.next = r;
    r.next = dn;
    d = dn;
    r = rn;
```

Unfold Of Linkedlist

O(n) → T

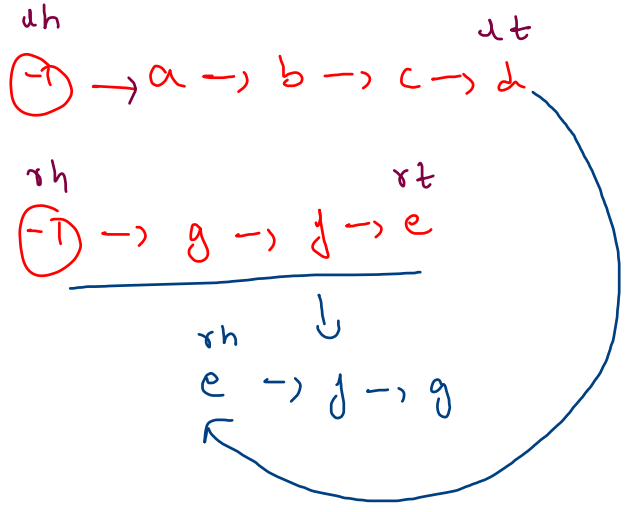
O(1) → S



(folded)

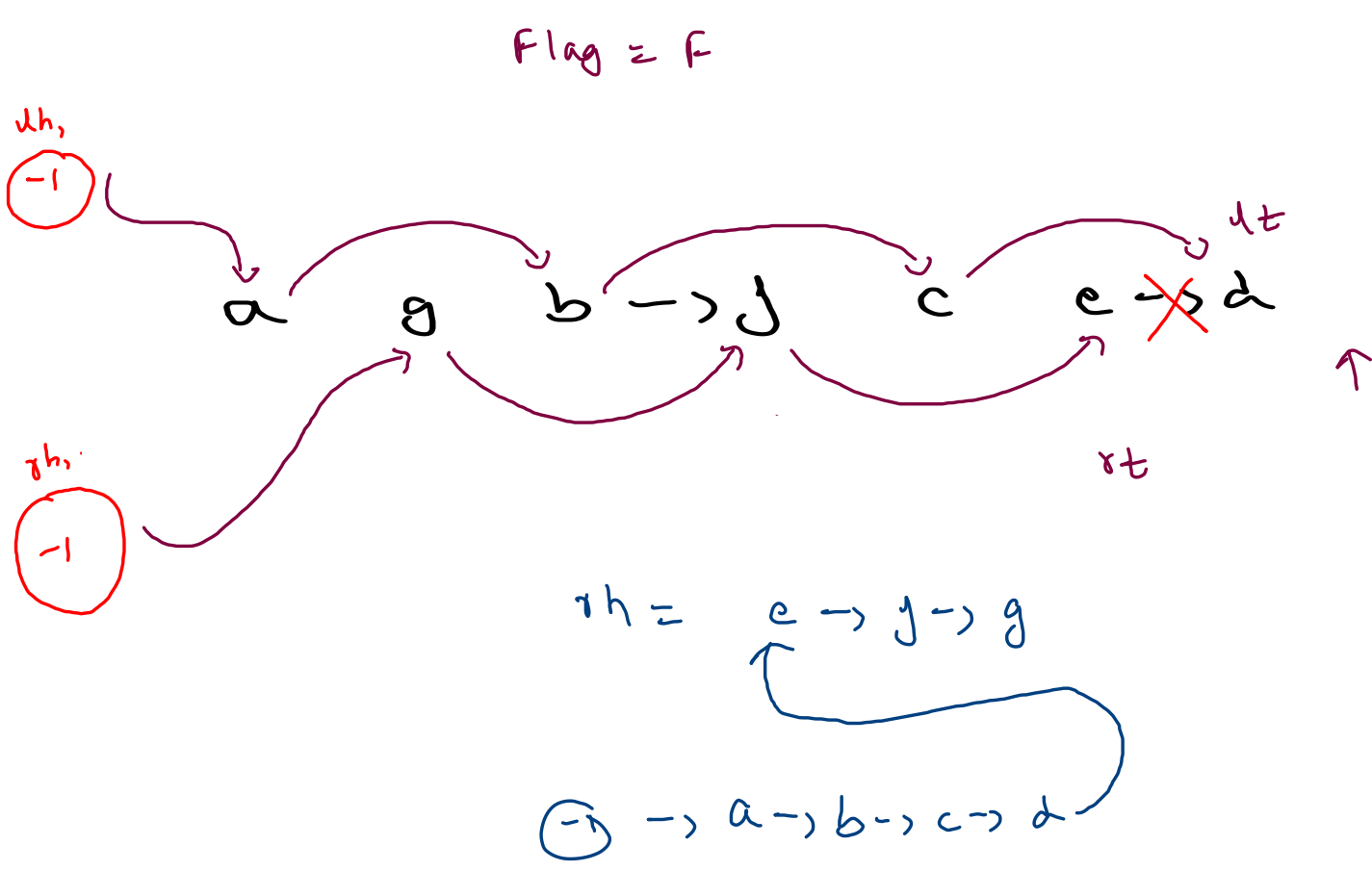
lt.next = null

rt.next = null

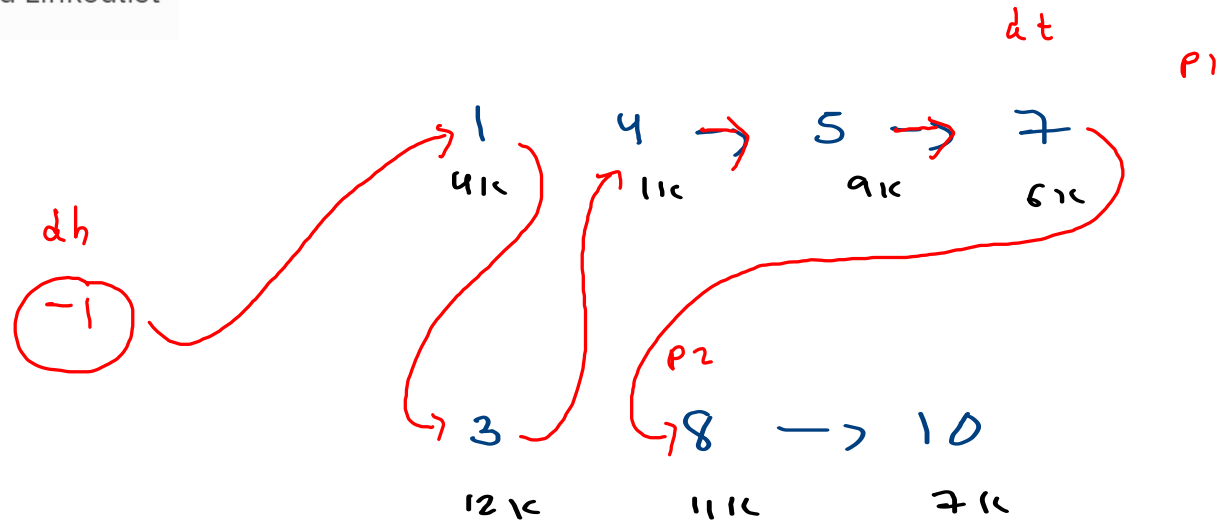



```
rh = rev(rh.next);

lt.next = rh;

return lh.next;
```

```java
public static void unfold(ListNode head) {
    ListNode lh = new ListNode(-1); //dummy
    ListNode lt = lh;

    ListNode rh = new ListNode(-1); //dummy
    ListNode rt = rh;

    ListNode temp = head;
    boolean flag = true;

    while(temp != null) {
        if(flag == true) {
            //add in left part
            lt.next = temp;
            lt = lt.next;
        }
        else {
            //add in right part
            rt.next = temp;
            rt = rt.next;
        }

        flag = !flag; //tweak
        temp = temp.next;
    }

    rt.next = null;

    rh = reverse(rh.next);
    lt.next = rh;

}
```



Merge Two Sorted Linkedlist



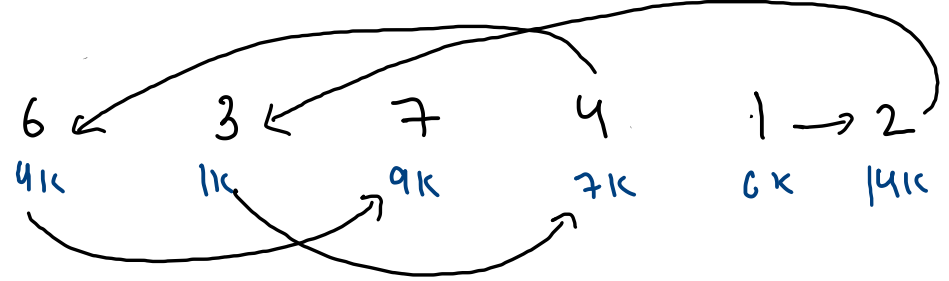
(i) O(n+m)

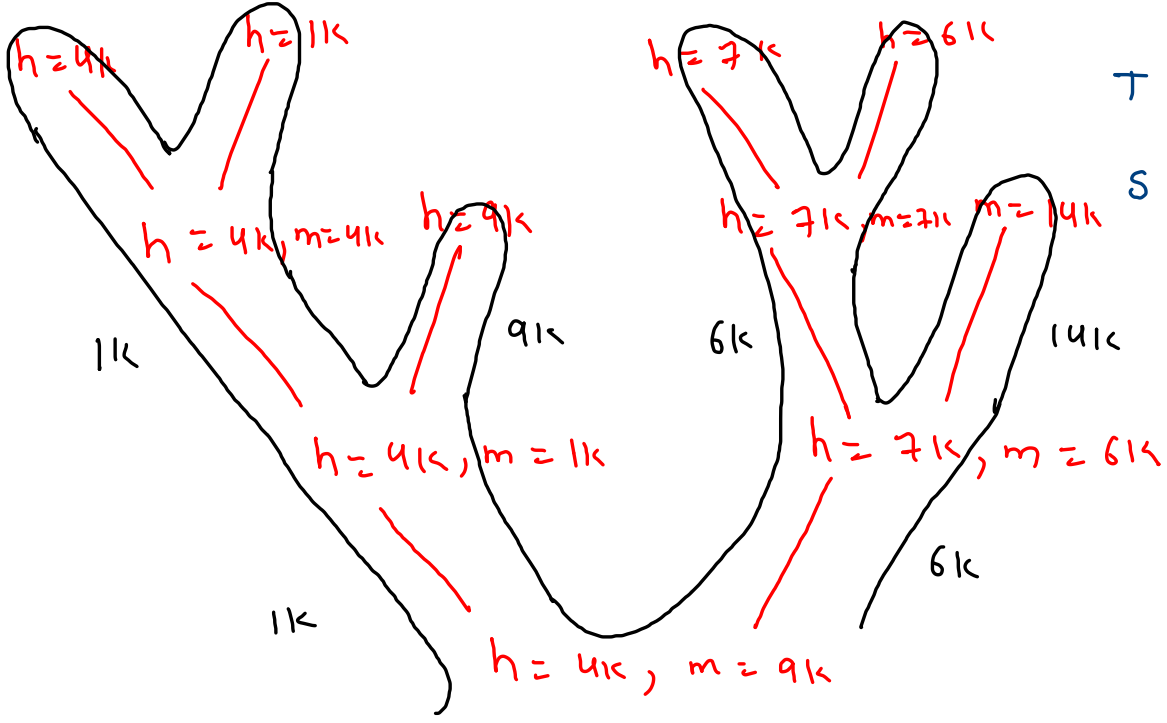
(ii) O(1)

(iii) without using new node multiple times.

return dh.next

Mergesort Linkedlist





T : nlogn

S : no extra space

(recursive space is allowed)

recursion space: logn

6k