7th August Morning

1. Multiplication → Addition
2. Merge k Sorted Divide and Conq.
3. Reverse of linked list using addFirst
4. Remove duplicates from sorted list → Add Last / Normal
5. Remove all duplicates
6. Clone a linkedlist with Random pointer
   ↳ O(1) → without using space

7th August Evening

1. Mathematical proof of cyclic linkedlist
2. Clone a linkedlist using Hash Map
3. Segregate odd-Even
4. segregate 01 → swap data / Nodes arrangement
5. segregate 012 → swap data / Nodes arrangement

8th August (Morning)

1. Segregate node of Linkedlist over last Index
2. Segregate Node of Linked list over pivot Index
3. quick sort in linkedlist

8th August (Evening) Doubly linkedlist

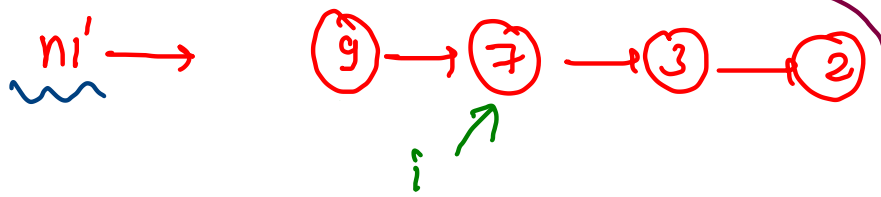
1. Get First + Get Last + Get At
2. Add At
3. Remove At
4. Add Before + Add After
5. Remove Before + Remove After
6. Remove Node in DLL.

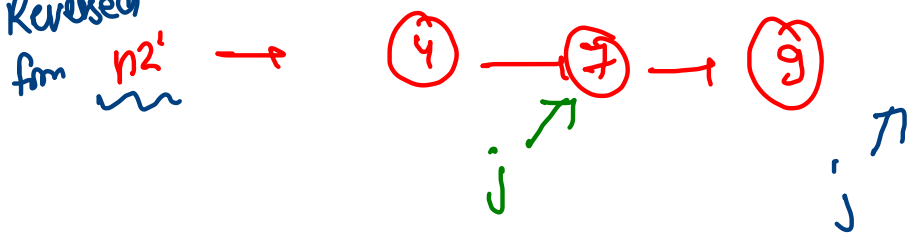
Creation of doubly linked list

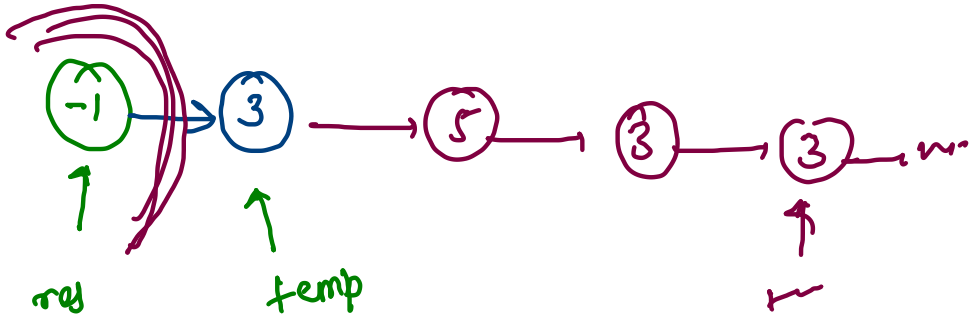
10th August

1. Display Forward and Backward
2. LRU cache → Application of DLL
3. Miscellaneous.

Addition Of Two LinkedList







ival = 9
jval = 4
sum = 13

carry != 0 || i != null || j != null

carry = 0 / 1

ival = i == null ? 0 : i.val;

jval = j == null ? 0 : j.val;

i = i == null ? null : i.next

j = j == null ? null : j.next

sum = ival + jval + carry

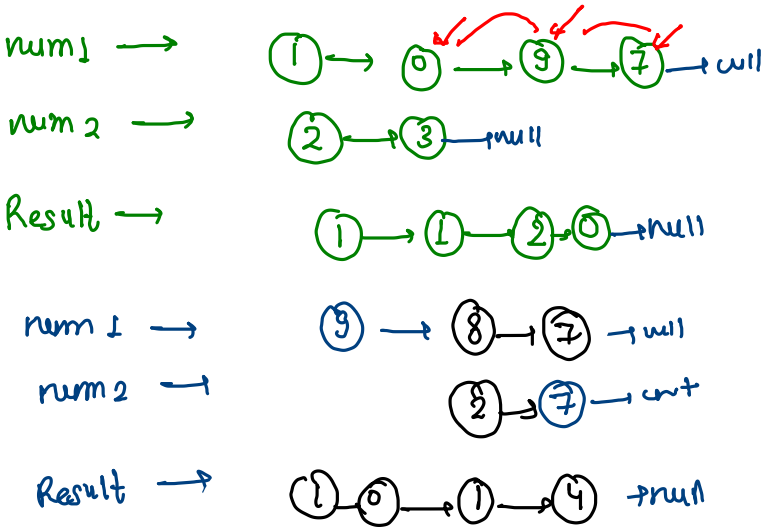
val = sum % 10 val = new ListNode(sum%10)

carry = sum / 10

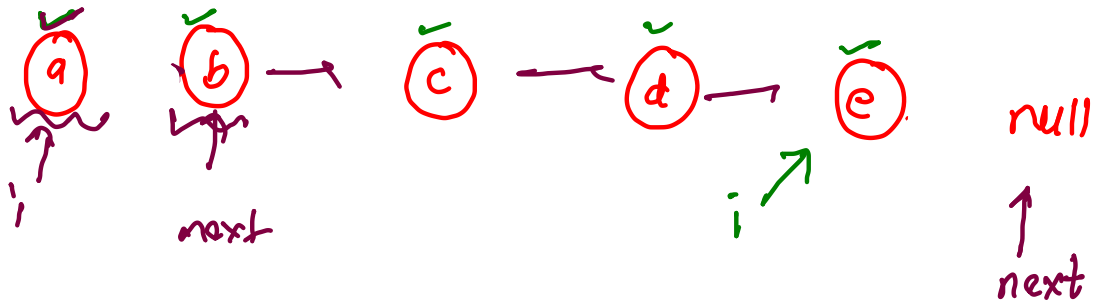
temp.next = val;

temp = val

```
  2379
   974
  3353
```

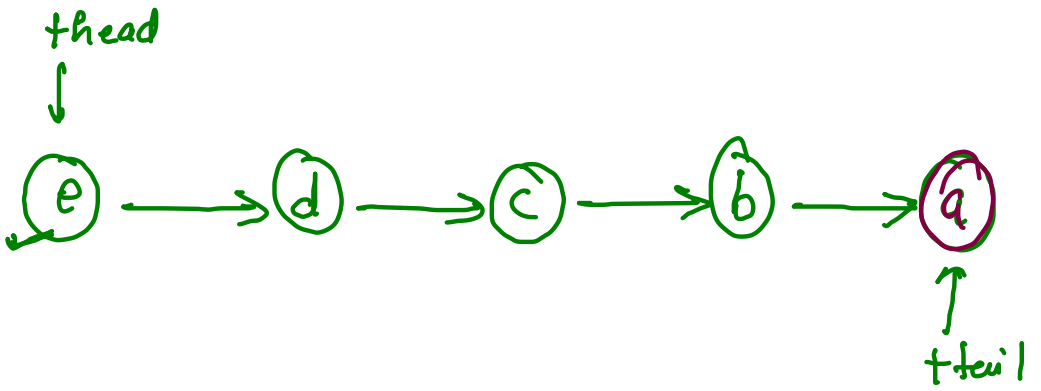


Reverse using addFirst



thead = ~~a~~ ~~b~~ ~~c~~ ~~d~~ e

ttail = a




```java
// reverse using addFirst Technique
static ListNode thead;
static ListNode ttail;

public static void addFirst(ListNode node) {
    if(thead == null) {
        thead = ttail = node;
    } else {
        node.next = thead;
        thead = node;
    }
}

public static ListNode reverseUsingAddFirst(Li
    thead = ttail = null;
    ListNode next = head;

    while(next != null) {
        ListNode i = next;
        next = i.next;
        i.next = null;
        addFirst(i);
    }
    return thead;
}
```

ListNode thead = n
ListNode n.

Multiplication Of Two LinkedList

result = null

result = 0

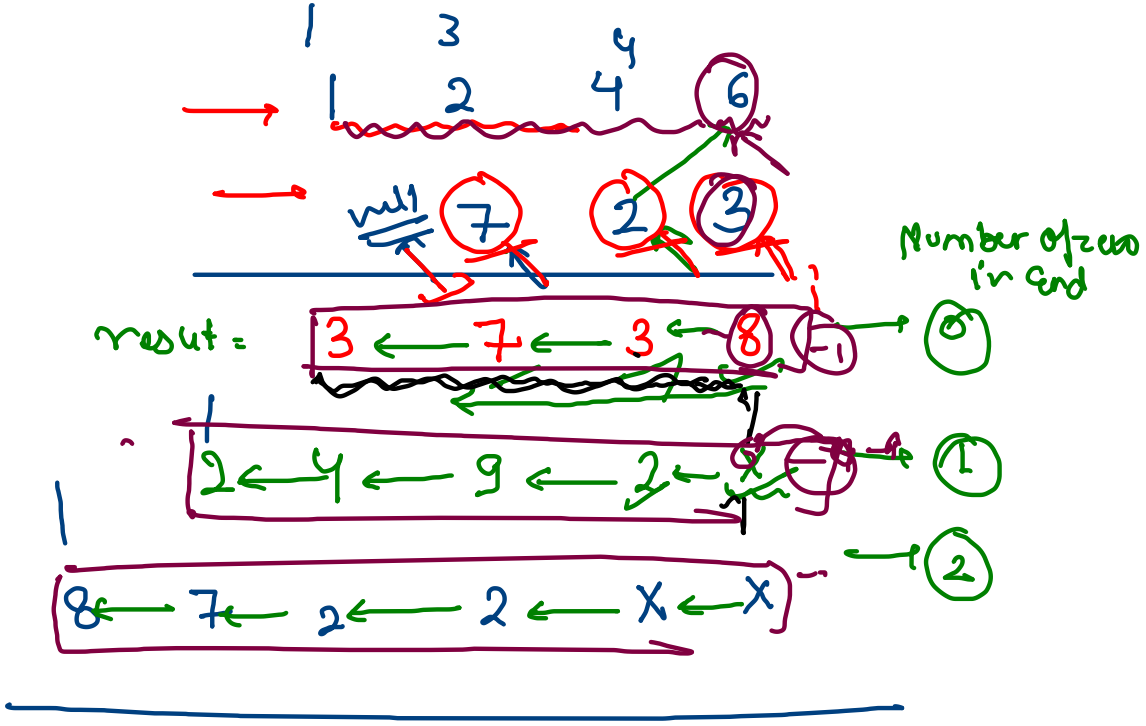
+ 34738

+ 24920

+ 872200

result = 900858

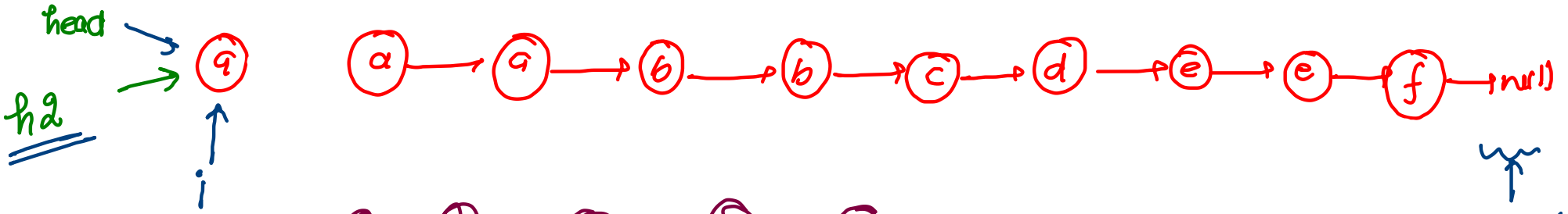
Add first

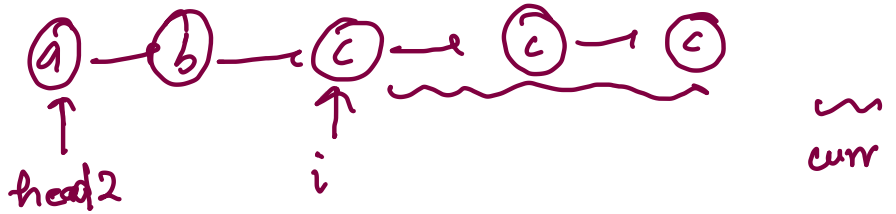


Result ← 9 ← 0 ← 0 ← 8 ← 5 ← 8

Reverse in Result →

Remove Duplicates from Sorted LinkedList

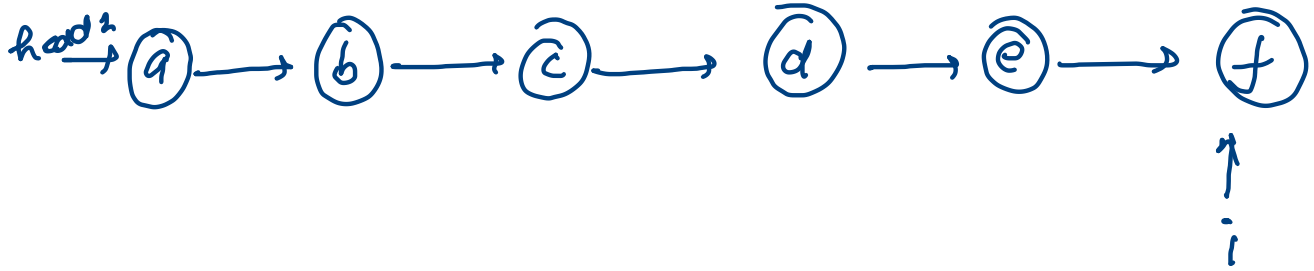




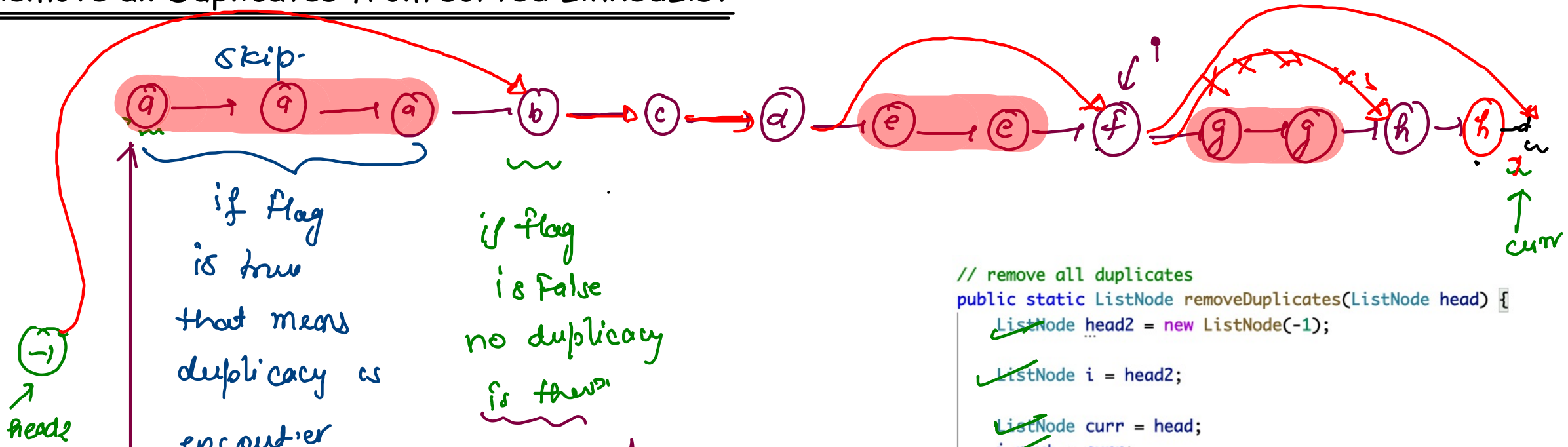

```
while(curr != null) {
    if(i.val != curr.val){
        i.next = curr;
        i = curr;
    }
    curr = curr.next
}
```

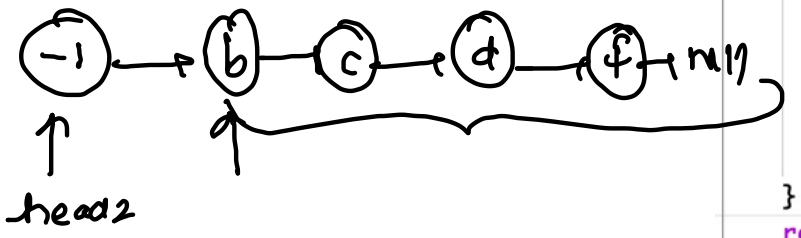
head2 = head

curr = head.next



Remove all Duplicates from sorted LinkedList





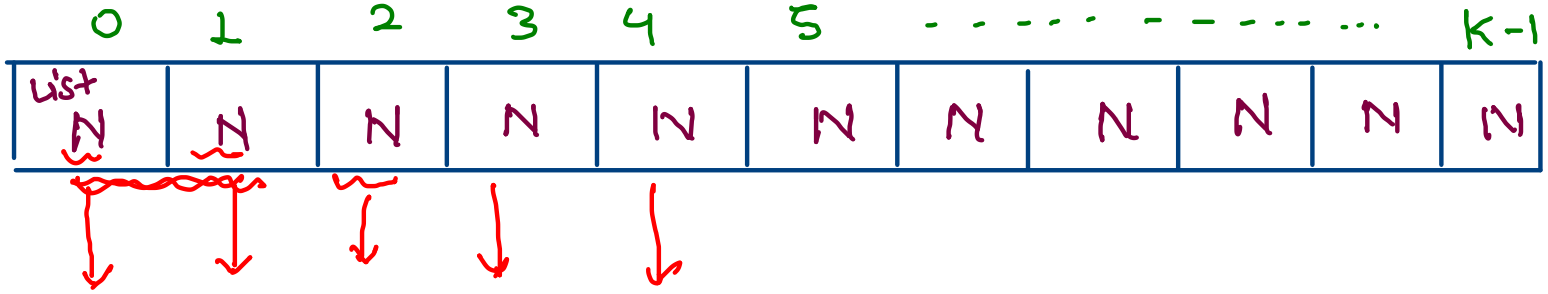

```java
// remove all duplicates
public static ListNode removeDuplicates(ListNode head) {
    ListNode head2 = new ListNode(-1);

    ListNode i = head2;

    ListNode curr = head;
    i.next = curr;
    while(curr != null) {
        curr = curr.next;
        boolean flag = false;
        while(curr != null && i.next.val == curr.val) {
            flag = true;
            curr = curr.next;
        }

        if(flag == true) {
            i.next = curr;
        } else {
            i = i.next;
        }
    }
    return head2.next;
}
```

Merge K Sorted LinkedList Using Divide and Conq.



res = 0 + N + N + N + N + N + ... + N ⇒ overall space = kN

size

Cost of Merging ⟶ N + 2N + 3N + 4N + 5N + ... + kN

Time = N + 2N + 3N + 4N + ... + kN

= N(1 + 2 + 3 + ... + k)

= $N\left(\frac{k(k+1)}{2}\right)$

⇒ $\boxed{\text{Time} = O(Nk^2)}$

list → m size

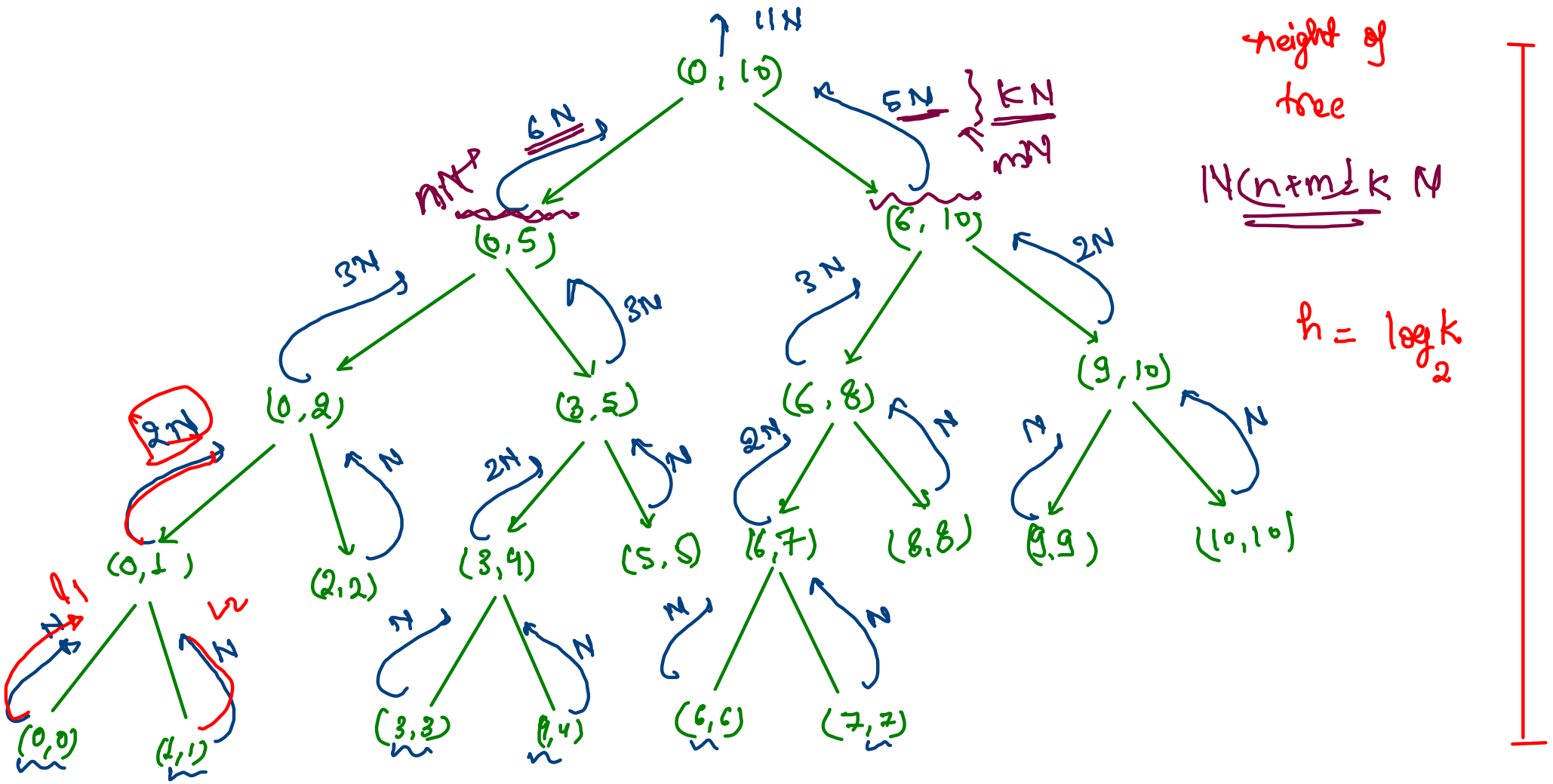
list → n size

↳ cost of Merging → (m+n)

N ⇒ max. no of elements in a list, k ⇒ No of lists.

k=11

0 to 10 ⇒ k=11 , 11 different list



height of tree

$h = \log_2 k$

$N(n+m) \leq k\ N$

Time ⇒ T

$T(k) = T(k/2) + T(k/2) + kN$

$K \longrightarrow \frac{K}{2} \longrightarrow \frac{K}{4} \longrightarrow \frac{K}{8} \longrightarrow \cdots\cdots \quad 8 \to 4 \to 2 \to 1$

No. of steps.

p^{th} term of G.P. $= a_0 r^{p-1}$

$a_0 \to$ first term

$r \to$ common ratio's

p^{th} term $\boxed{a_p = 1}$, $p = ?$ (No. of steps)

$a_p = a_0 r^{p-1}$, $a_0 = K$, $r = \frac{1}{2}$

$1 = K\left(\frac{1}{2}\right)^{p-1}$

$2^{p-1} = K$, take $\log_2$ both side

$\log_2 2^{p-1} = \log_2 K$. $\boxed{\log_a a = 1}$

$p - 1 = \log_2 K$

$\boxed{p = \log_2 K + 1}$

$T(K) = T\left(\frac{K}{2}\right) + T\left(\frac{K}{2}\right) + KN$

$T(K) = 2T\left(\frac{K}{2}\right) + KN \longrightarrow (1)$

eq(1) $\times 2$, $K \to \frac{K}{2}$: $2T\left(\frac{K}{2}\right) = 2^2 T\left(\frac{K}{2^2}\right) + 2\frac{K}{2}N \longrightarrow (2)$

eq(2) $\times 2$, $K \to \frac{K}{2}$: $2^2 T\left(\frac{K}{2^2}\right) = 2^3 T\left(\frac{K}{2^3}\right) + 2^2\frac{K}{2^2}N \longrightarrow (3)$

eq(3) $\times 2$, $K \to \frac{K}{2}$: $2^3 T\left(\frac{K}{2^3}\right) = 2^4 T\left(\frac{K}{2^4}\right) + 2^3\frac{K}{2^3}N \longrightarrow (4)$

$\vdots \qquad \vdots$

$2^{p-1} T\left(\frac{K}{2^{p-1}}\right) = 2^p T\left(\frac{K}{2^p}\right) + 2^{p-1}\frac{K}{2^{p-1}}N \longrightarrow (p)$

p steps.

Add all Eq^{n}s $\rightarrow$ $T(K) = 2^p T(1) + \underbrace{KN + KN + KN + \cdots}_{p \text{ times}}\ KN$

Add all Eq^{n}s → $T(k) = 2^{p} T(1) + \underbrace{kN + kN + kN + \cdots}_{p \text{ times}} kN$

$\underbrace{p = \log_2 k + 1}_{\log_2 k}$ $\quad \underbrace{T(1) = 1}$

$$T(k) = 2^{\log_2 k} \cdot 2 + p\,kN$$

$$= 2k + (\log_2 k + 1)\, kN$$

$$T(k) = 2k + \underbrace{Nk \log_2 k} + kN$$

$$\boxed{T(k) = O(Nk \log_2 k)}$$

↳ Divide and Conq. have better complexity for merging as compare from iterative merging]

NOTE: comparison is not for Priority Queue approach.