18 Given N Array elements, re-arrange the array such that

- → ar[0] should go to its sorted position
- → All Elements < ar[0] go to left side of ar[0]
- → All Elements > ar[0] go to right side of ar[0]

	0	1	2	3	4	5	6	7	8	9	10
ar[11] = {	10	3	8	15	6	12	2	18	7	15	14 }

App:1 // Sort : 2 3 6 7 8 10 12 14 15 15 18

Merge Sort

App2: tmp[11] =

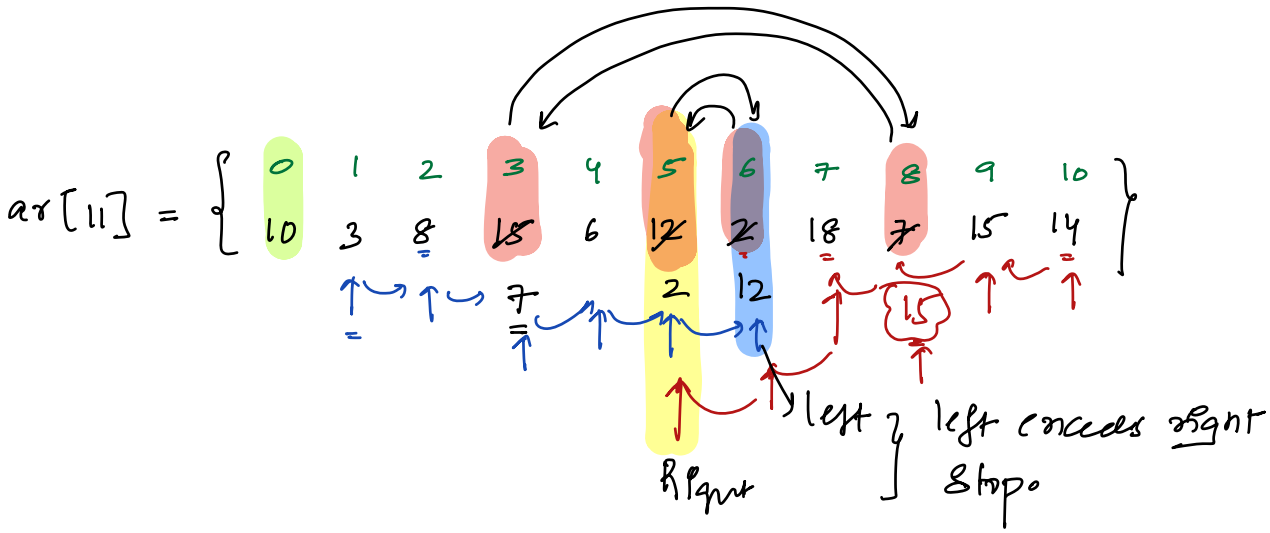
	0	1	2	3	4	5	6	7	8	9	10
	3	8	6	2	7	10	14	15	18	12	15

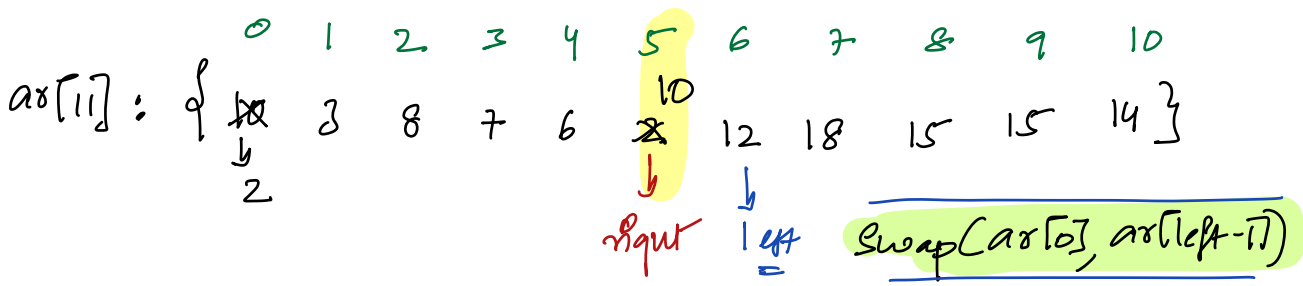
< 10 | > 10

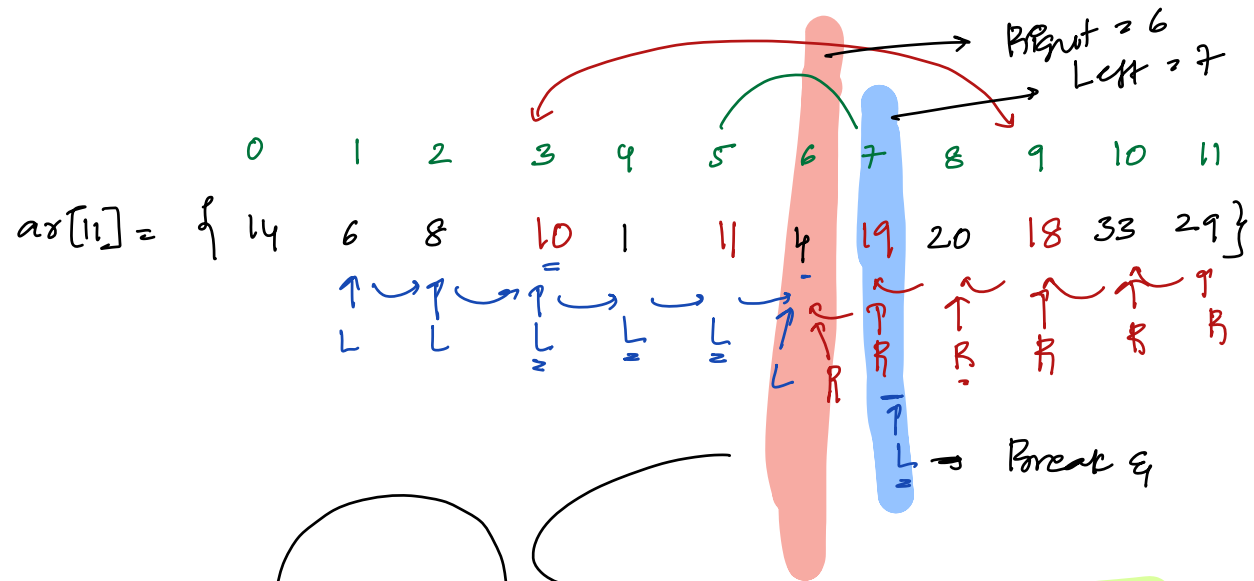
Pseudocode:

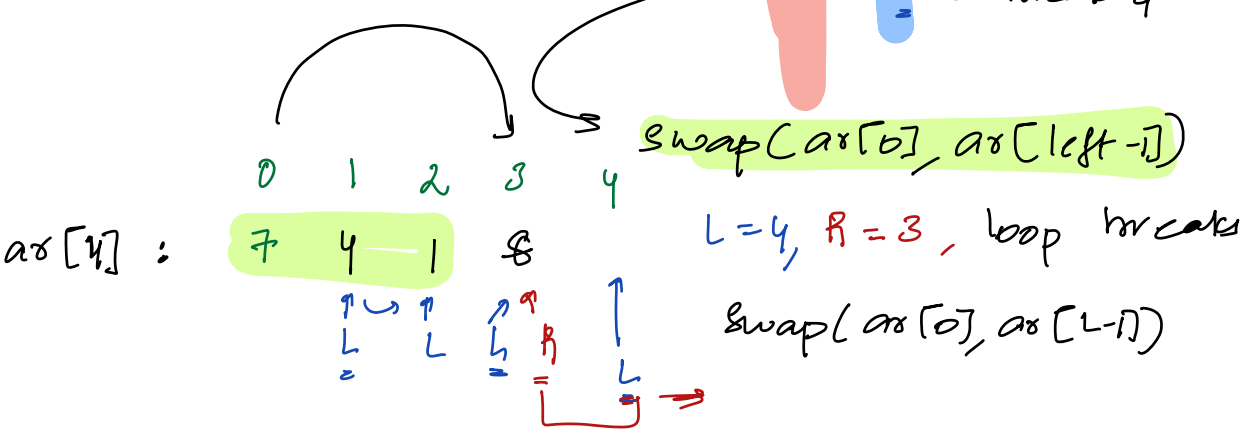
TC: O(N)
SC: O(N)

```
tmp[N], L = 0, R = N-1
i = 1; i < N; i++{
    if( ar[i] < ar[0]) { tmp[L] = ar[i]; L++ }
    else { tmp[R] = ar[i]; R-- }
}
tmp[L] = ar[0]
// copy tmp[] -> ar[]
```


ar[11] = { 0 1 2 3 4 5 6 7 8 9 10
10 3 8 15 6 12 2 18 7 15 14 }
7
2
12
15
left
Right
left crosses right
& stop.


ar[11] : { 0 1 2 3 4 5 6 7 8 9 10
3 8 7 6 2 10 12 18 15 15 14 }
2
Right
left
Swap(ar[0], ar[left-1])


Right = 6
Left = 7
ar[11] = { 0 1 2 3 4 5 6 7 8 9 10 11
14 6 8 10 1 11 4 19 20 18 33 29 }
L
R
Break &


Swap(ar[0], ar[left-1])
ar[4] : { 0 1 2 3 4
7 4 1 8 }
L=4, R=3, loop breaks
Swap(ar[0], ar[L-1])

```
// int re-arrange ( ar[], s, e) {
    L = s+1 , R = e
    while ( L <= R ) {
        if ( ar[L] <= ar[s] ) { L++ }
        else if ( ar[R] > ar[s] ) { R-- }
        else {
            swap ar[L] & ar[R]
            L++, R--
        }
    }
    // swap ar[s] & ar[L-1]
    // ar[s] is going to L-1
    return L-1
}
```

TC: O(N)

SC: O(1)

2) Given N array elements & subarray [s e].

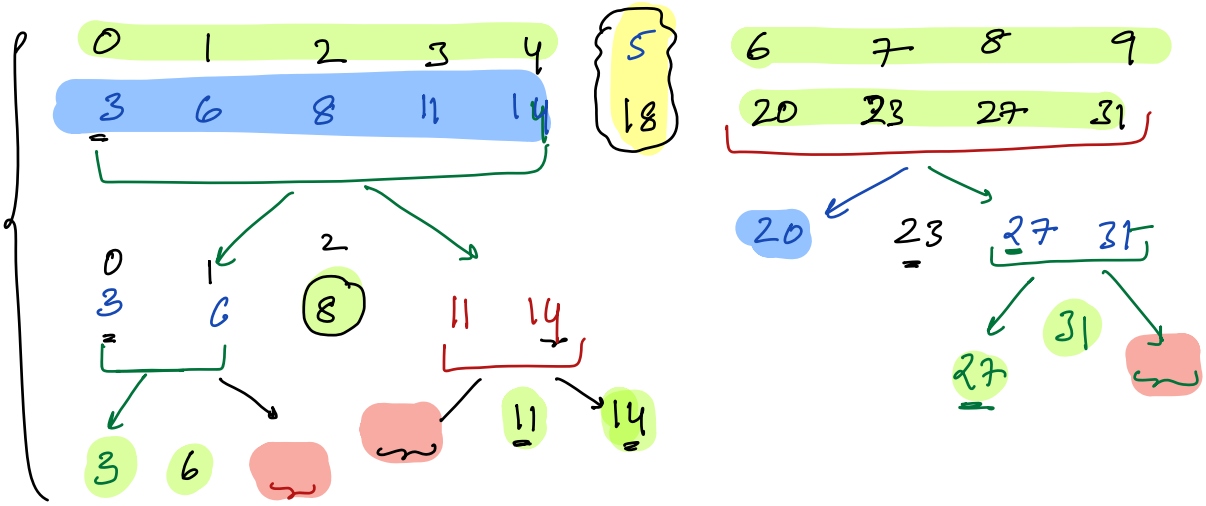
Re-arrange subarray [s e],

Ex:	0	1	2	3	4	5	6	7	8	9
[s e]	10	3	8	6	4	7	14	12	7	1
[2 7]				L	L	L R		L		

L=6, R=5

swap (ar[2], ar[5])

	0	1	2	3	4	5	6	7	8	9
ar[10] =	18	8	6	3	11	14	23	20	31	27

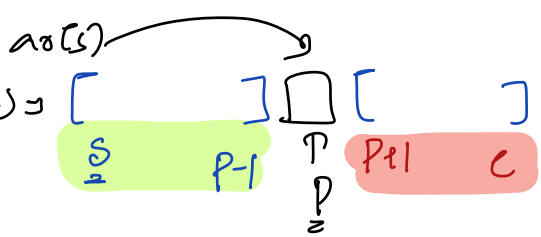



```
QuickSort(ar[], s, e) {
    if(s >= e) return;
    p = re-arrange(ar[], s, e)
    QuickSort(ar, s, p-1)
    QuickSort(ar, p+1, e)
}
```

[s p-1] p [p+1 e]

```
// int re-arrange ( ar[] , s, e) {
    L = s+1 , R = e
    while ( L <= R ) {
        if ( ar[L] <= ar[s] ) { L++ }
        else if ( ar[R] > ar[s] ) { R-- }
        else {
            swap ar[L] & ar[R]
            L++, R--
        }
    }
    // swap ar[s] & ar[l-1]
    // ar[s] is going to L-1
    return L-1
}

QuickSrt ( ar[], s, e) {
    if(s >= e) { return; }
    p = re-arrange (ar[], s, e);
    QuickSrt (ar, s, p-1)
    QuickSrt (ar, p+1, e)
}
```



// Best Case

$T(N) = N + T(N/2) + T(N/2)$

$T(N) = 2T(N/2) + N$

$T(N) = N \log N$, SC: $O(\log N)$

Worst Case

$T(N) = N + T(N-1)$

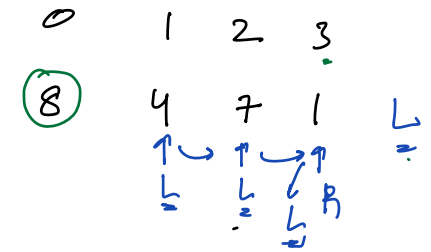
$= N + N-1 + T(N-2)$

$= N + N-1 + N-2 + T(N-3)$

$T(N) = O(N^2)$

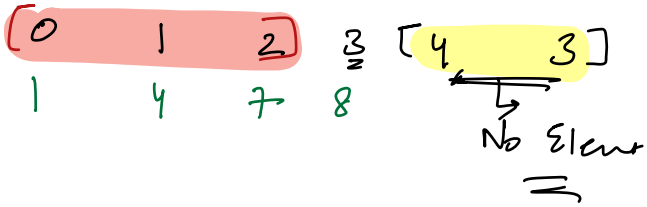
SC: $O(N)$

// TC



L=4, R=3 break

Swap (ar[0], ar[3])



	Best	Avg Case (end)	Worst
Quick Sort :	O(NlogN)	O(NlogN)	O(N²)

Even if we split a 90% & 10%

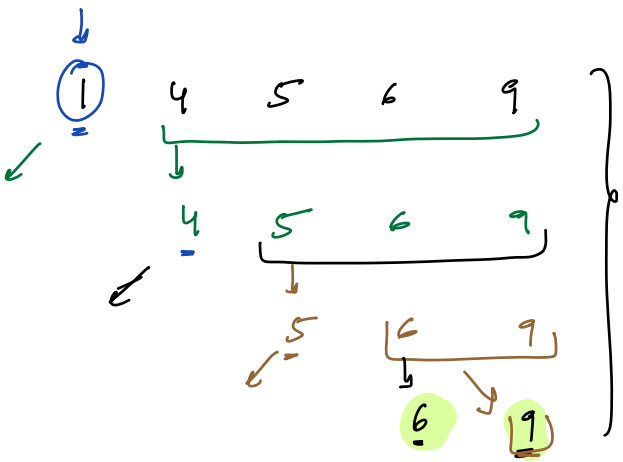
$T(N) = N + T(9N/10) + T(N/10)$ → we cannot use masters Theorem

TODO

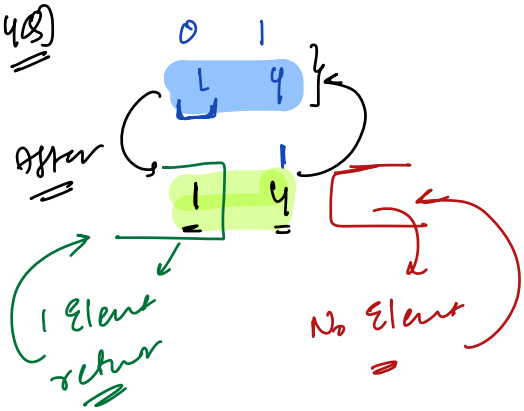
$T(N) = \sim O(N \log_{10} N)$

We will have a constant factor

Ex:



10:45 pm break


0
1
L
4
1
4
1 Element
No Element

5ⓑ

ຂໍ້1: 3 5 ~~5~~ 7 ~~5~~ 9 10 14

3 5 7 ~~9~~ ~~10~~ ~~14~~ ~~8~~ 14

8 ~~8~~ ~~8~~

ຂໍ້2: 3 5 7 8 9 10 14

0 1 2 3

ຂໍ້3: 1 4 6 8

0 1 2 3 4 5 6

ຂໍ້4: {3} 1 5 -2 4 7 2

{1 3} 5 -2 4 7 2

1 3 5 -2 4 7 2

-2 1 3 5 4 7 2

-2 1 3 4 5 7 2

-2 1 3 4 5 7 2

-2 1 2 3 4 5 7

2 [illegible]

// Pseudocode :

i=1 : [0 0]

```
for(i = 1; i < N; i++){
    // Trying to insert i^th
    [0, i-1] data is sorted
    j = i-1;
    while (j >= 0 && ar[j] > ar[j+1]){
        swap ar[j] & ar[j+1]
        j--
    }
}
```

TC: $O(N^2)$

SC: $O(1)$

$\sum_{i=0}^{n-1} i$

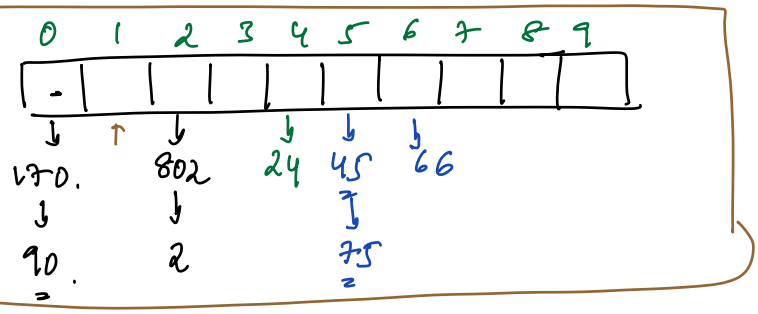
(Radix Sort)

Idea :

170 45 75 90 802 24 2 66

// Sort data Based on 0^{th} digit

lptr[10]:

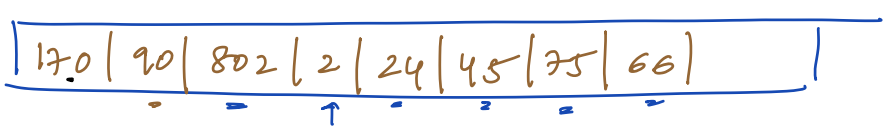


using lptr[] data modify original array.

Sol 1: using inbuilt comparator

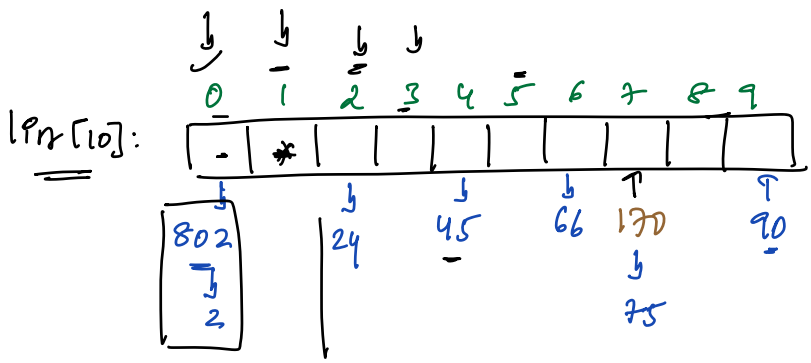
TC: O(NlogN)

SC: O(N)

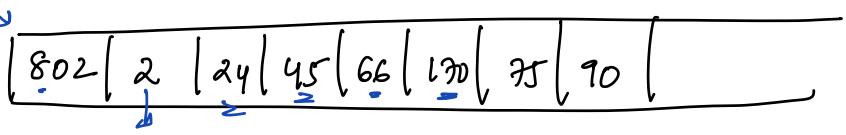


Sort data based on 1^{st} digit

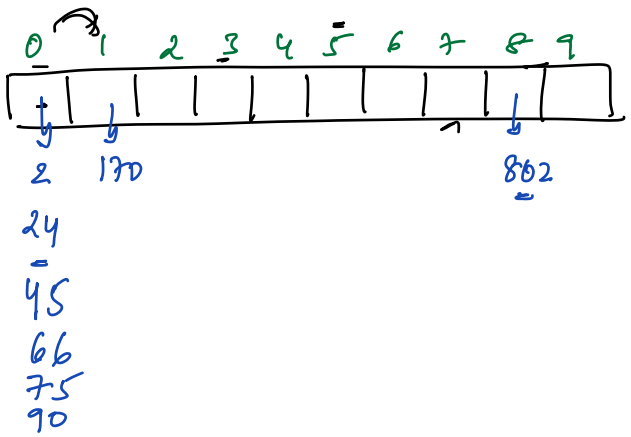
lptr[10]:

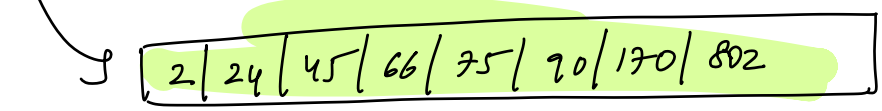


Sort data based on 2^{nd} digit



lptr[10]:





Idea :

- Sort data based on 0^{th} digit
- On above data sort 1^{st} digit
- On above data sort 2^{nd} digit
- On above data sort n^{th} digit → {number of digits in max, max value}

SC : O(N) TC : $n * N$ → $(\log_{10}^{(max)})$

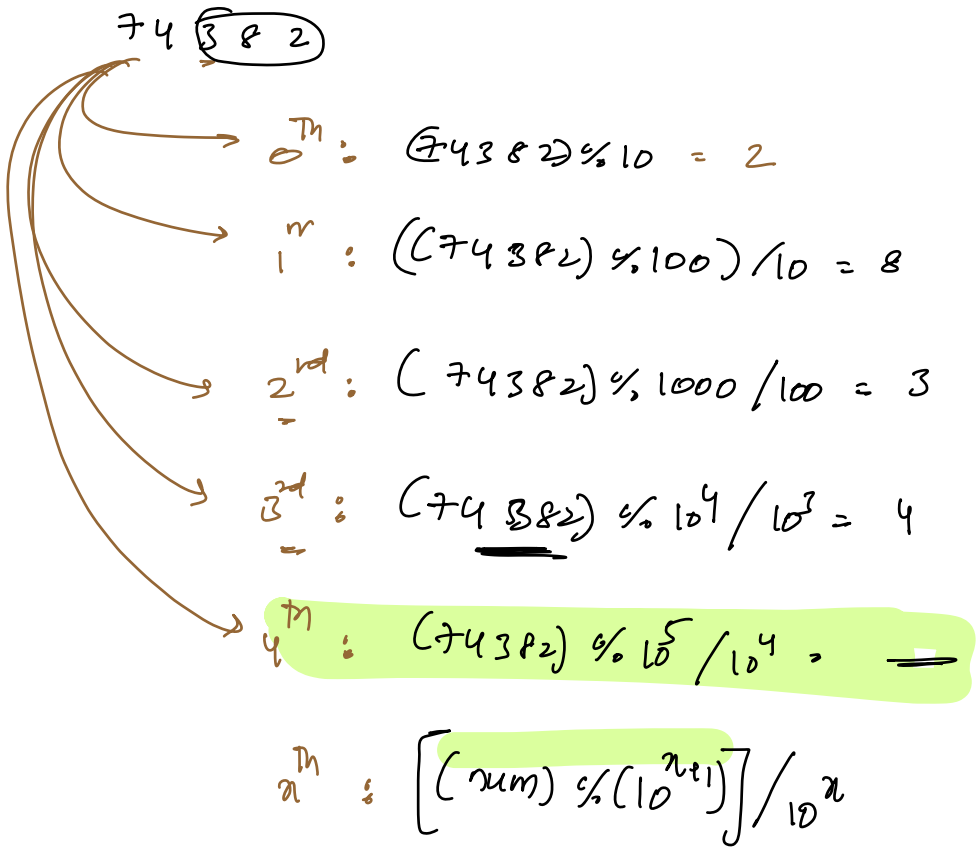
SC : O(N) TC : $(\log_{10}^{(max)} * N)$

$\log_{10}^{(max)}$ = $\frac{2 * 10^9}{10}$ = digits * 10 digits

TC = $10 * N$

TC = O(N)

Doubt :




```
digit(N){
    c = 0;
    while(N>0)
    {
        N = N/10
        c++
    }
}
```

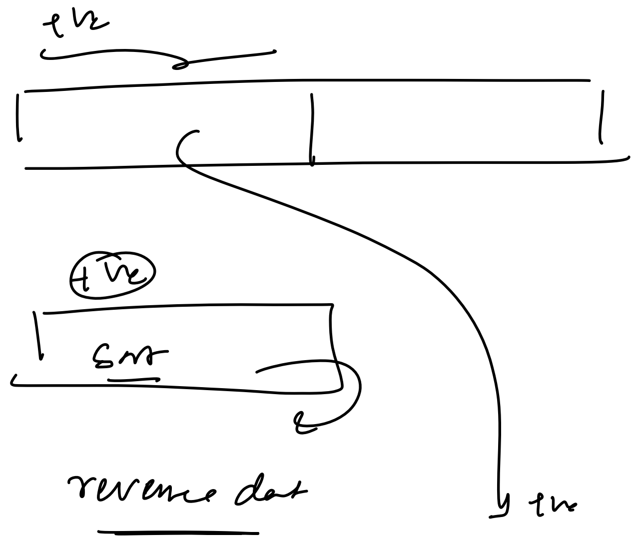
Step 1 : max array element get number of digits → digit = d

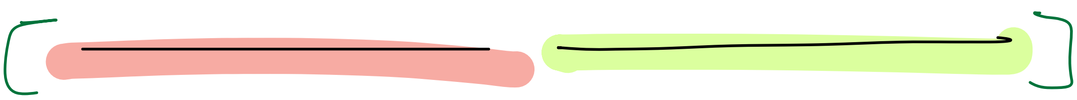
```
for(i=0; i<d; i++){
    every number extract ith digit
    // array of List ; List dig[10];
    for(j=0; j<N; j++) {
        deg = [ar[j] % (10^(i+1))] / 10^i
        dig[deg].insert(ar[j])
    }

    k=0
    for(j=0; j<10; j++) {
        for(s=0; s<dig[j].size(); s++){
            ar[k] = dig[j][s]
            k++
        }
    }
    // ar() modified
}
// ar[] Sorted
```

array

+ve / -ve





// Doubts