Q1 Given N Array elements, rearrange the array such that:

- arr[0] should go to the correct sorted position.
- All elements $\leq$ arr[0] go to it's left in any order.
- All elements $>$ arr[0] go to it's right in any order

Ex1 arr[] =

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

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

Approach 1 : Sort it : Tc: O(nlogn)
Sc: O(n)

Approach 2 :

arr[] =

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

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

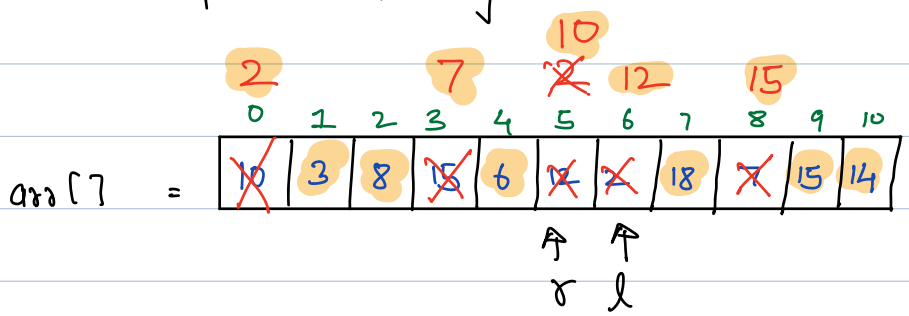
l

r

Tc: O(n)
Sc: O(n)

Approach 3

Can space complexity be reduced.



1st swap (3, 8)

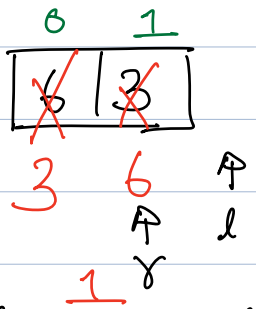
2nd swap (5, 6)

After all operations correct pos for arr[0] = r / l-1

TC: O(n)

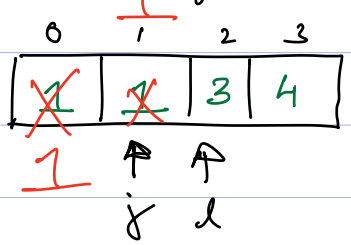
SC: O(1).

Ex1



Swap (arr[0], arr[l-i])

Ex2

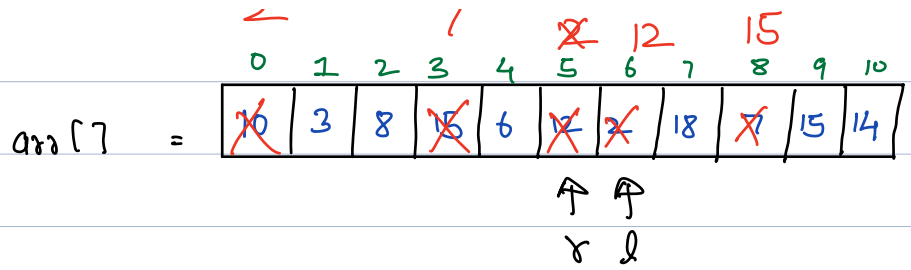


Pseudo Code

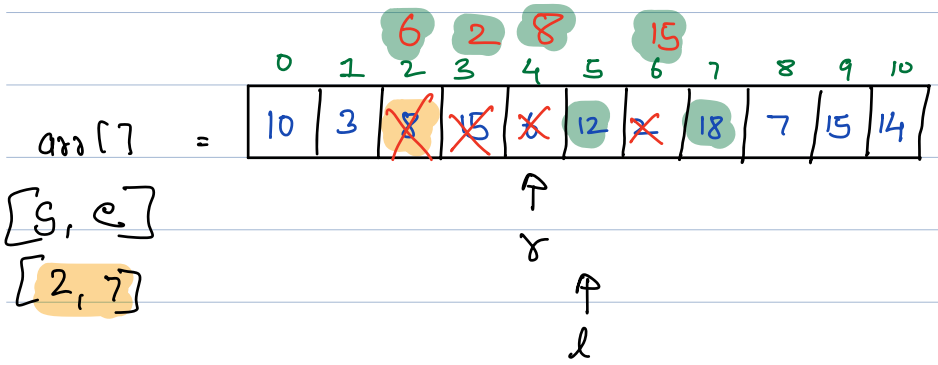
```
void rearrange (int arr[], int n) {
    if (n == 1)
        return;
    int l => 1
    int r => n-1

    while (l ≤ r) {
        if (arr[l] ≤ arr[0]) {
            l++;
        } else if (arr[r] > arr[0]) {
            r--;
        } else {
            swap (arr[l], arr[r]);
            l++; , r--;
        }
    }

    swap (arr[0], arr[l-1]);
}
```


arr[7] =
0 1 2 3 4 5 6 7 8 9 10
10 3 8 15 6 12 2 18 7 15 14
r
l

Q2 Given N array elements & subarray [s, e].
Rearrange subarray [s, e]. ⇒ putting arr[s] in the correct position relative to the subarray.



[s, e]
[2, 7]

Pseudo Code

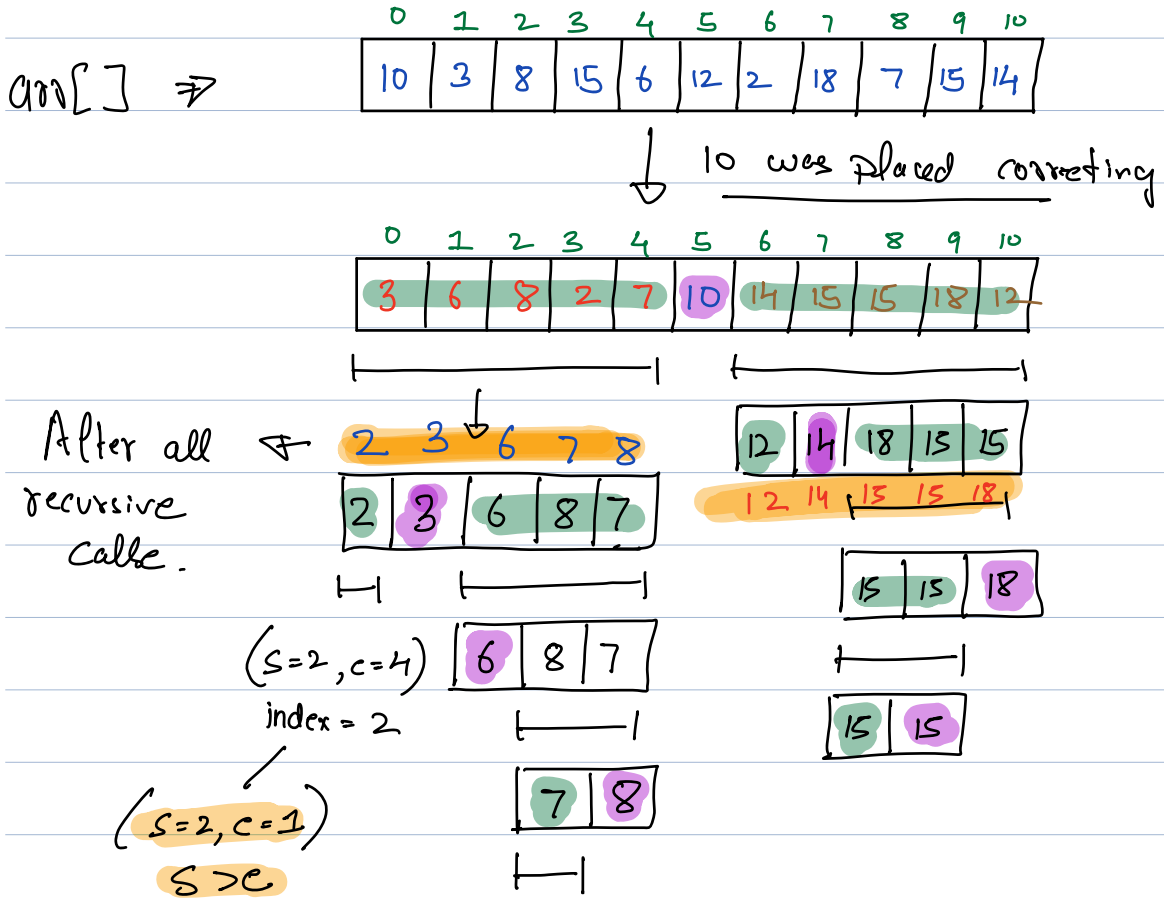
```
int rearrange (int arr[], int s, int e) {
    if (n == 1)
        return;
    int l = s+1
    int r = e

    while (l ≤ r) {
        if (arr[l] ≤ arr[s]) {
            l++;
        } else if (arr[r] > arr[s]) {
            r--;
        } else {
            swap (arr[l], arr[r]);
            l++; , r--;
        }
    }

    swap (arr[s], arr[l-1]);

    return (l-1);
}
```

TC: O(n)
SC: O(1)




```
void QuickSort (int arr[], int s, int e) {

if (s ≥ e) ──→ s > e | no element
    return;    ──→ s == e | single element.

int index = rearrange (arr, s, e); // O(n)
QuickSort (arr, s, index-1);
QuickSort (arr, index+1, e);

}
```

TC: Best Case

$T(n) \Rightarrow n + T(n/2) + T(n/2)$

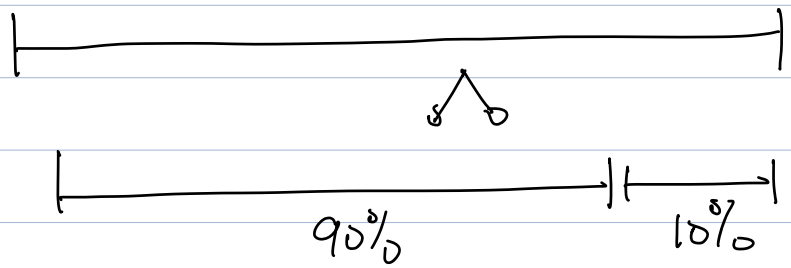
$T(n) \Rightarrow n \log n$

Worst Case

$T(n) \Rightarrow n + T(1) + T(n-1)$

$T(n) \Rightarrow n^2$ 1 2 3 4

Average Case ⇒

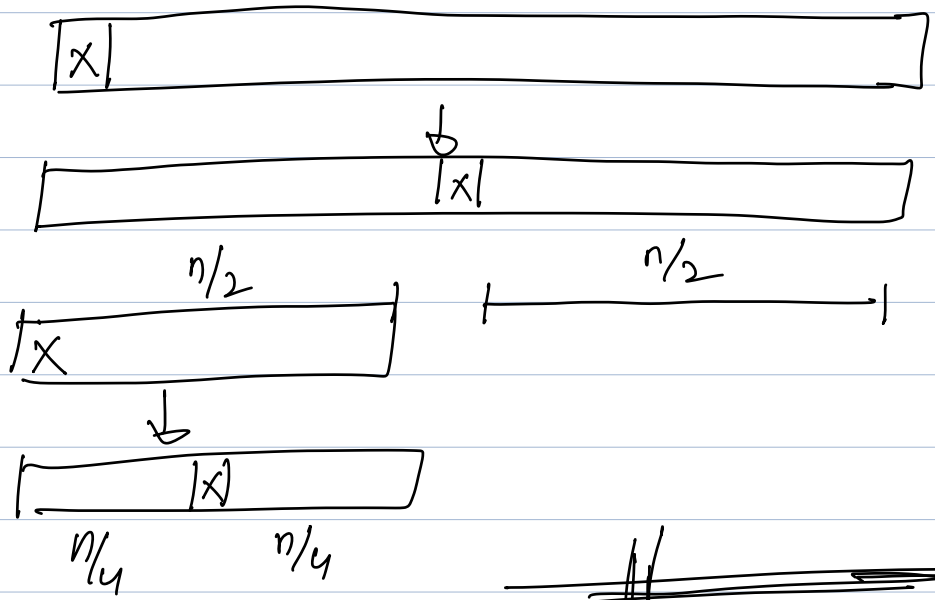


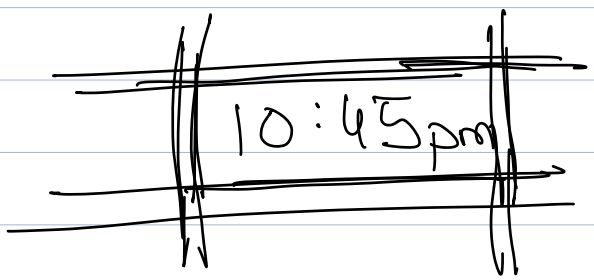
$T(n) \Rightarrow n + T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right)$

$T(n) \Rightarrow O(n \log n)$

SC: $O(H) \rightarrow O(\log n)$ ⇒ best, average

$O(n) \rightarrow$ worst case.


x
x
n/2
n/2
x
x
n/4
n/4


10:45pm

Card

⇒ 9, 8, 1, 4, 7, 3, 4, 2

9

8 9

1 8 9

1 4 8 9

1 4 7 8 9

1 3 4 7 8 9

1 3 4 4 7 8 9

1 2 3 4 4 7 8 9

Stable ✓

TC: $O(n^2)$

SC: $O(1)$

Inplace.

arr[] =

0	1	2	3	4	5	6
1	3	4	6	7	9	10

i

j (j & j+1)

While processing i^{th} element, we know that $[0, i-1]$ is already sorted

Pseudo Code

```
for (int i = 1 ; i < n ; i++) {
    int j = i - 1;

    while ( j ≥ 0 && arr[j] > arr[j+1]) {
        swap (arr[j], arr[j+1]);
        j--;
    }

}
```

x_1 x_2 x_3 x_4 x_5 x_6

	TC	Inplace	Stable	Swaps
Bubble	$O(n^2)$ $O(n)$ = best case	Yes	Yes	n^2
Selection	$O(n^2)$	Yes	No	n
Insertion	$O(n^2)$ $O(n)$ = best case.	Yes	Yes	n^2
QuickSort	$O(n^2)$ = worst $O(n \log n)$ best & average	No SC: $O(H$ SC: $O(\log n)$	No	$\geq n \simeq n \log n$
MergeSort	$O(n \log n)$	No SC: $O(n)$	Yes	no swaps

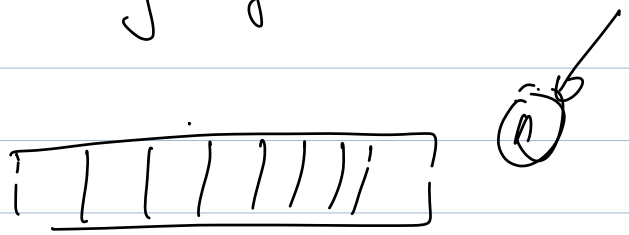
↳ $n \log n$ times we are comparing and writing.

SC: $O(1) \longrightarrow O(\log n)$ Space

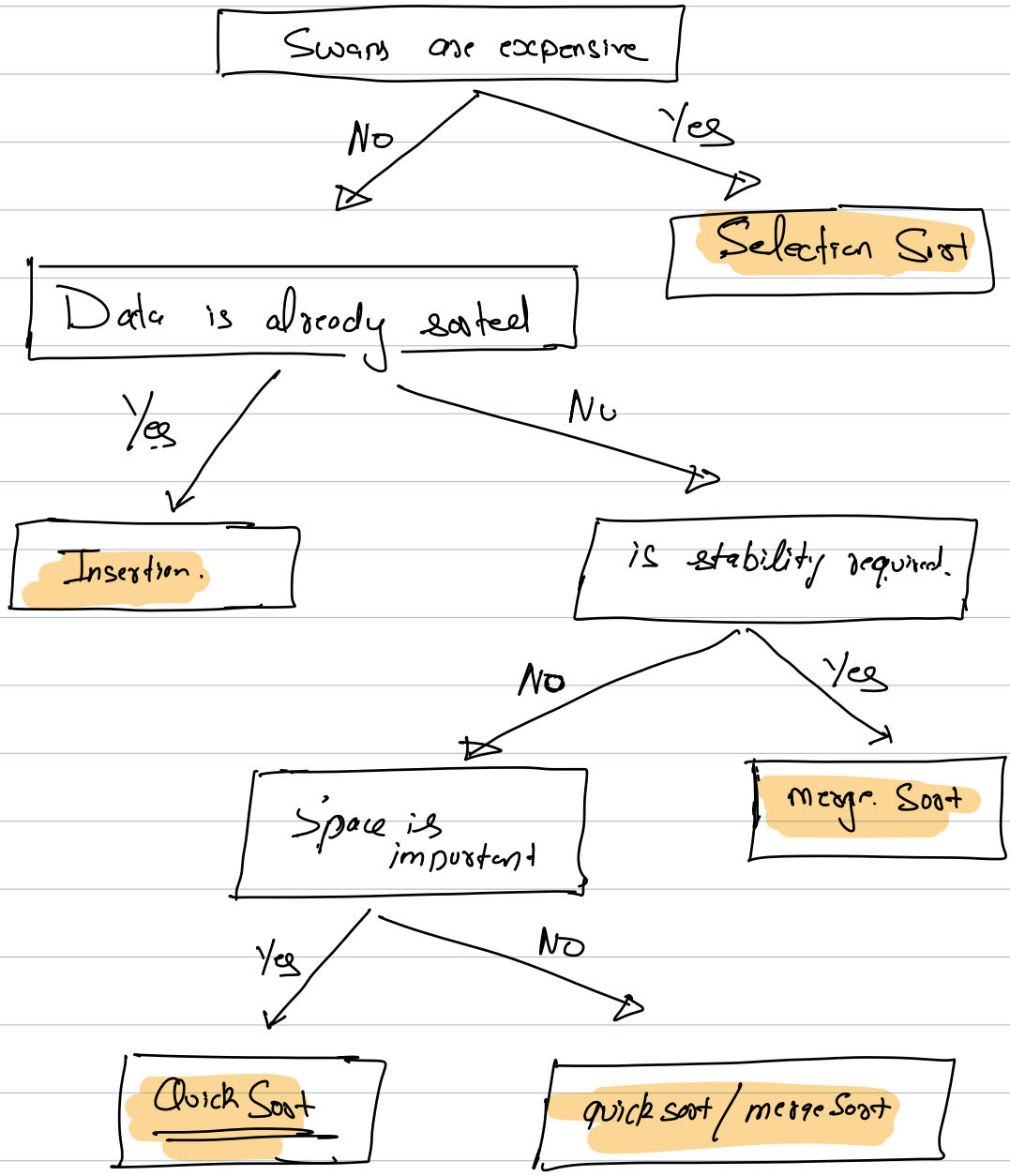
$n = 2^{32}$ $\log n \Rightarrow 32$

Metrics to judge a sorting algorithm.

1) Time
2) Space.
3) Stability.
4) Swaps.
5) Already have a sorted array.
6) How big is your data



↳ Can your data be stored on RAM full.

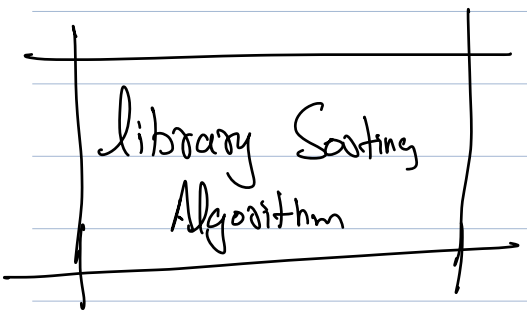

Swaps are expensive
No
Yes
Selection Sort
Data is already sorted
Yes
No
Insertion.
is stability required.
No
Yes
Space is important
merge Sort
Yes
No
Quick Sort
quick sort / merge Sort

Interview

1) Merge Sort / Quick Sort

2) General logic on important Sorting Algorithm.

3) Given situations, which sorting algorithm.



⇒ Hybrid of various Algorithm.

Radix Sort!

3 2 1 0

2 4 5 6

arr → | 22 | 59 | 100 | 6 | 999 | 200 | 33 |

1st Step: Sort according to the 0th digit.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

100 22 33 6 59

200 999

arr → | 100 | 200 | 22 | 33 | 6 | 59 | 999 |

2nd Step: Sort according to the 1st digit.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

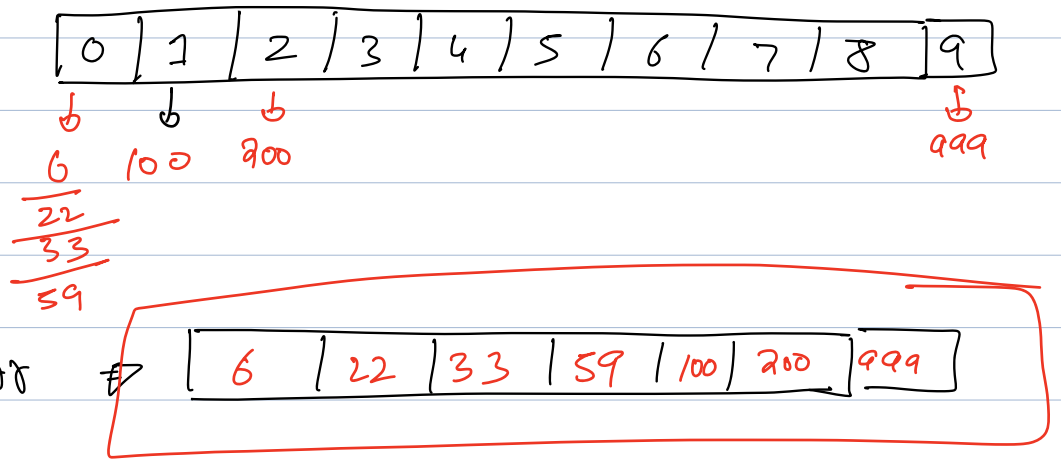
100 22 33 59 999

200

6

arr → | 100 | 200 | 6 | 22 | 33 | 59 | 999 |

3rd Step: Sort according to the 2nd digit



568 079

TC: $N \times K$

$K \rightarrow \log_{10}(\text{max\_number})$

↳ length of the maximum element

SC: $O(N)$

K = 100 n = 10