Agenda

1) Quick sort
2) Complexity analysis of Quicksort and merge sort
3) 1 question on custom comparison

Q.1 **Partition an array**

Given an array, partition it based on last element (ele) such that **all elements < ele** are coming on left of it and **all elements >= ele** are coming on right of it.

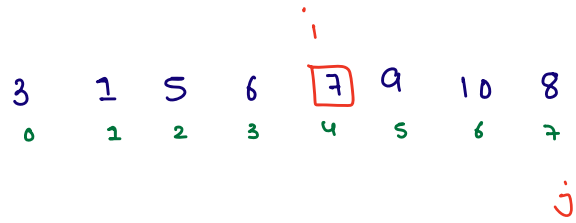
A = [8 9 3 1 5 6 10 **7**]
0 1 2 3 4 5 6 7

Expected TC : O(n)

(3 1 5 6) 7 (8 9 10)
↓ ↓
< 7 >= 7

A = [8 5 1 3 7 2 9 **6**]
0 1 2 3 4 5 6 7

(5 1 3 2) 6 (8 7 9)
↓ ↓
< 6 >= 6



i, j

i to j-1 ⇒ >= ele

ele = 7

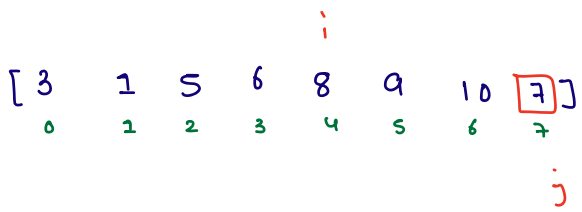
```
if (A[j] >= ele) {
    j++;
}
```

```
else {
    swap(A[i], A[j]);
    i++; j++;
}
```

```
swap(A[i], A[n-1]);
```

why?

=



i, j

i, j-1 ⇒ >= ele

↓

A[i] >= ele

```
if (A[j] >= ele) {
    j++;
}
```

```
else {
    swap(A[i], A[j]);
    i++; j++;
}
```

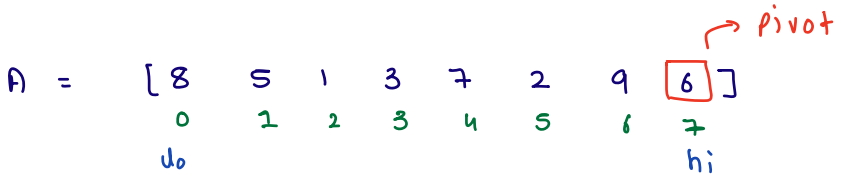
```
swap(A[i], A[n-1]);
```

⇒ finally your pivot (ele) is coming at i^{th} index.

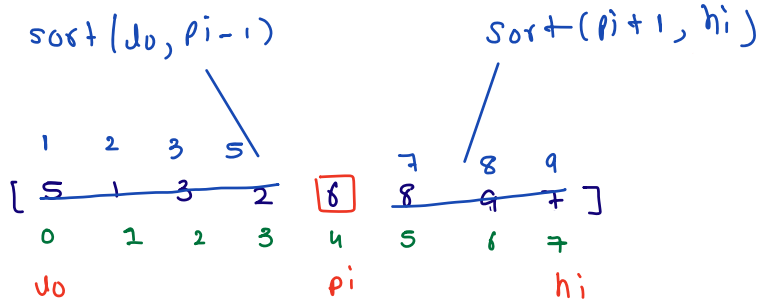
Quicksort

↳ divide and conquer

↳ recursion based



i) partition based on pivot and getting pivot index (pi)

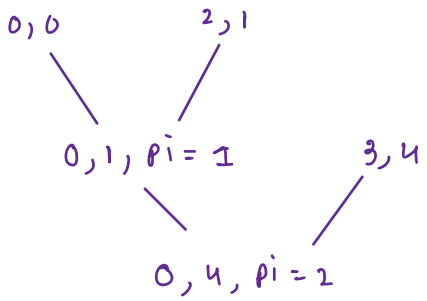


```
void quicksort (int [] A, int lo, int hi)
                                  if (lo >= hi) { return }
    int pivot = A[hi];
    int pi = partition(A, lo, hi, pivot);

    quicksort (A, lo, pi-1);
    quicksort (A, pi+1, hi);

}
```

A = [1 2 3 4 5]
    0 1 2 3 4

base case
=

lo hi



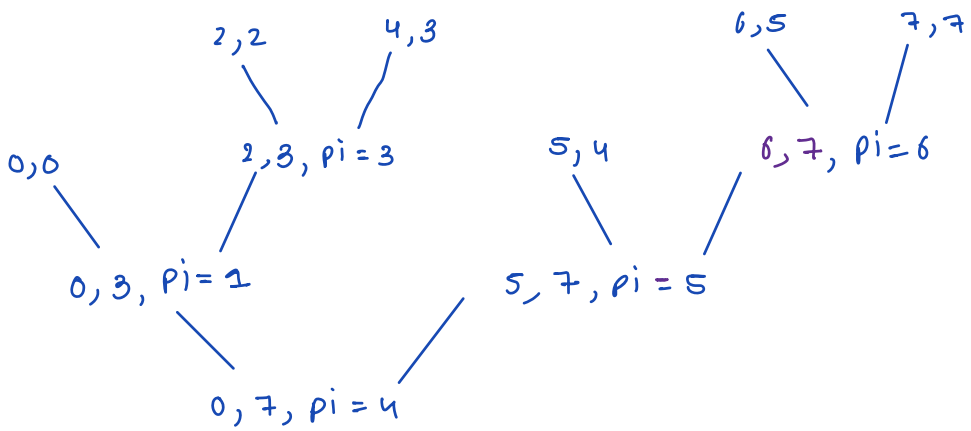
```java
static int partition(int[]A,int lo,int hi,int pivot) {
    int i=lo,j=lo;

    while(j < hi) {
        if(A[j] >= pivot) {
            j++;
        }
        else {
            //swap A[i],A[j]
            int temp = A[i];
            A[i] = A[j];
            A[j] = temp;

            i++;
            j++;
        }
    }

    //swap A[i],A[hi]
    int temp = A[i];
    A[i] = A[hi];
    A[hi] = temp;

    return i;
}
```

```java
static void quickSort(int[]A,int lo,int hi) {
    if(lo >= hi) {
        return;
    }

    int pivot = A[hi];

    int pi = partition(A,lo,hi,pivot);

    quickSort(A,lo,pi-1);
    quickSort(A,pi+1,hi);
}
```

if (A[j] >= pivot) {
    j++;
}
else {
    swap A[i], A[j]
    i++; j++;
}

A = [1 2 3 5 6 7 8 9]
    0 1 2 3 4 5 6 7

A = [8 5 1 3 7 2 9 6]
    0 1 2 3 4 5 6 7



(lo, hi, pi)

Practice contest

→ window 8am to 8pm

→ 2 different ques { 1 on hashing, 1 on recursion }

→ 90 min

→ Discussion => 10pm

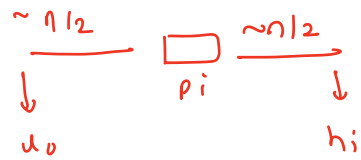
TC and SC complexity analysis:

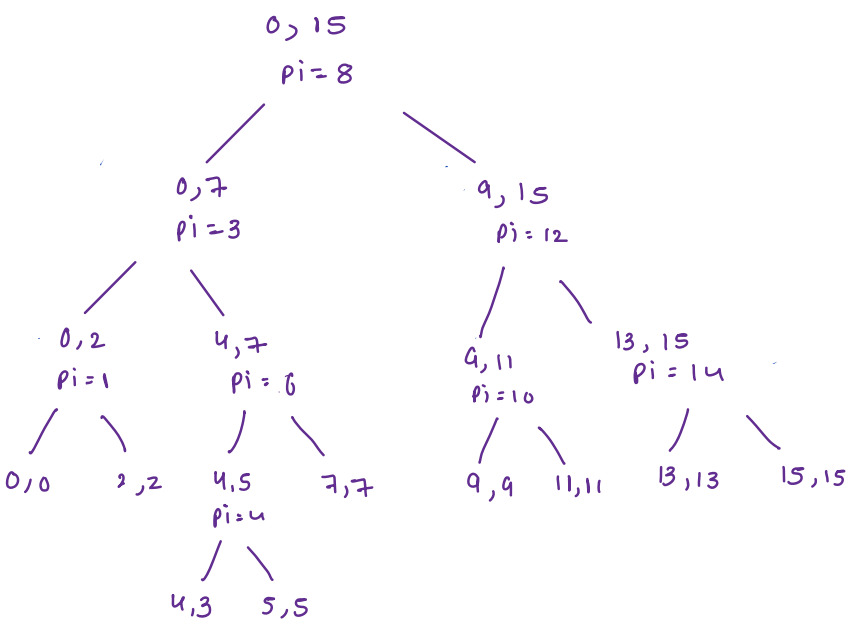
i) quicksort

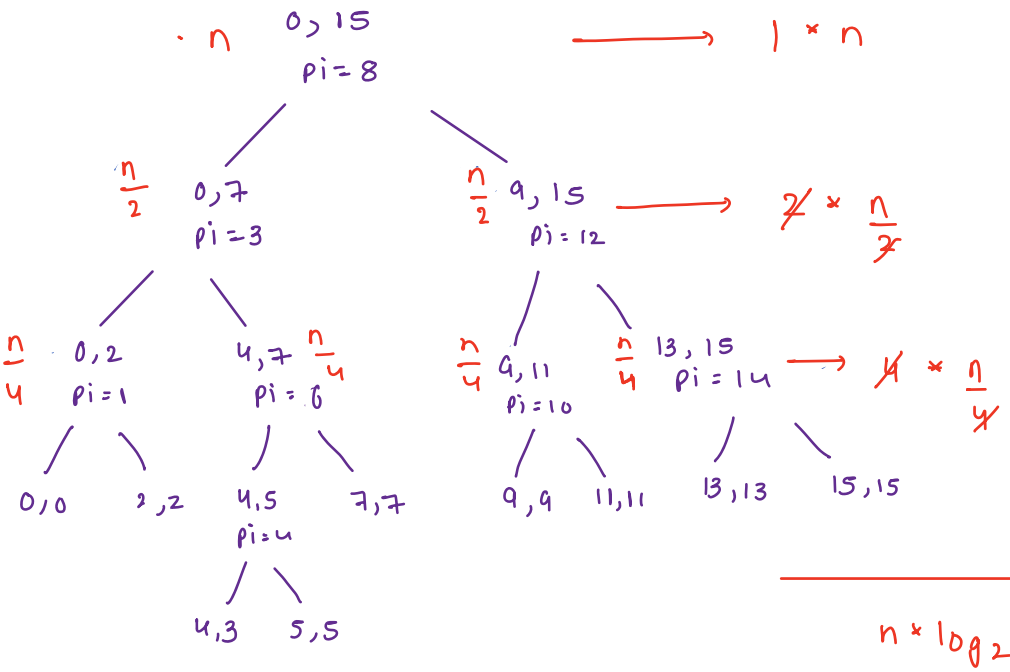
n: original array length

best thing to happen:

pivot is coming in mid (lo to hi)







$n * \log_2 n$

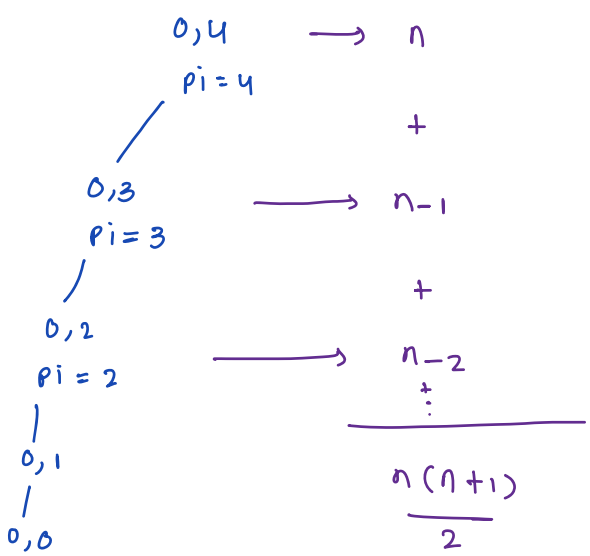
TC: $O(n \log_2 n)$

(Best)

worst scenario

↳ pi is coming not in mid but on one of the corner

TC: $O(n^2)$



$\frac{n(n+1)}{2}$

TC of quicksort :

	TC	SC (only due to call stack space)
Best	$O(n \log n)$	$O(\log_2 n)$
worst	$O(n^2)$	$O(n)$

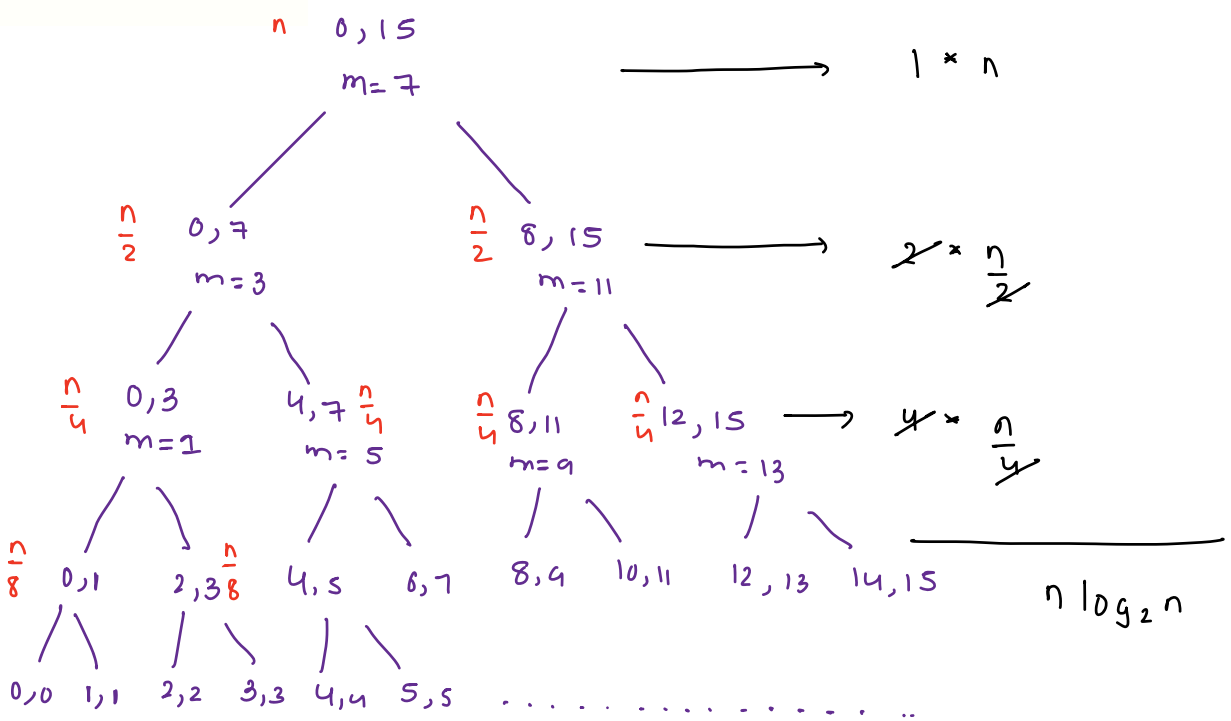
TC of Quicksort : $O(n \log n)$ to $O(n^2)$

SC of Quicksort : $O(\log_2 n)$ to $O(n)$

Merge Sort

```java
static int[] merge(int[]A,int[]B) {
    int n = A.length;
    int m = B.length;
    int[]ans = new int[n+m];

    int i=0,j=0,k=0;

    while(i < n && j < m) {
        if(A[i] < B[j]) {
            //use A[i]
            ans[k] = A[i];
            i++;
            k++;
        }
        else {
            //use B[j]
            ans[k] = B[j];
            j++;
            k++;
        }
    }

    //if values are pending in A[]
    while(i < n) {
        ans[k] = A[i];
        i++;
        k++;
    }

    //if values are pending in B[]
    while(j < m) {
        ans[k] = B[j];
        j++;
        k++;
    }

    return ans;
}
```

```java
static int[] mergeSort(int[]arr,int lo,int hi) {
    if(lo == hi) {
        int[]sa = new int[1];
        sa[0] = arr[lo]; //or arr[hi]
        return sa;
    }

    int mid = (lo + hi)/2;

    //sort the array from lo to mid
    int[]A = mergeSort(arr,lo,mid);

    //sort the array from mid+1 to hi
    int[]B = mergeSort(arr,mid+1,hi);

    int[]ans = merge(A,B);
    return ans;
}
```

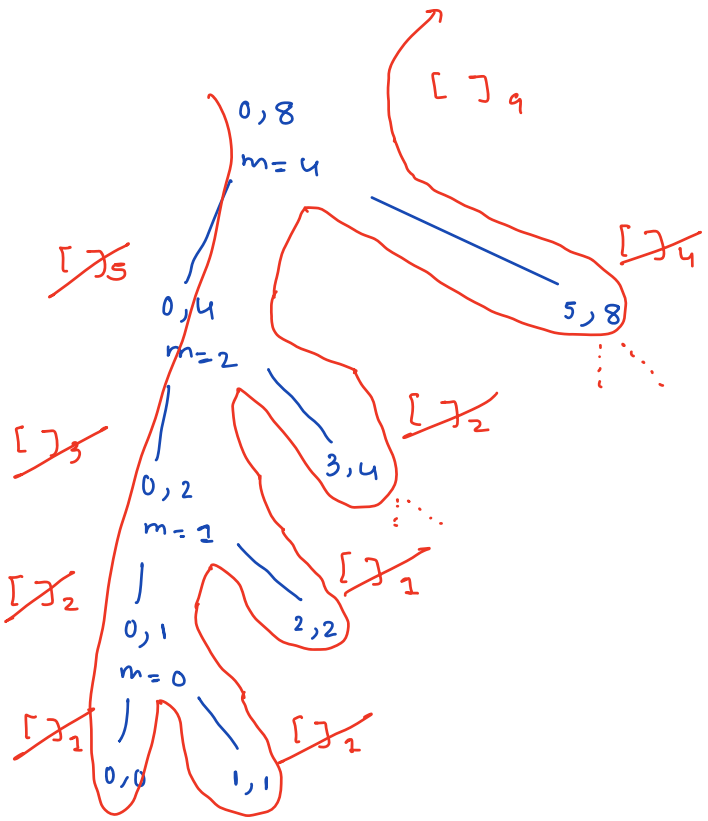


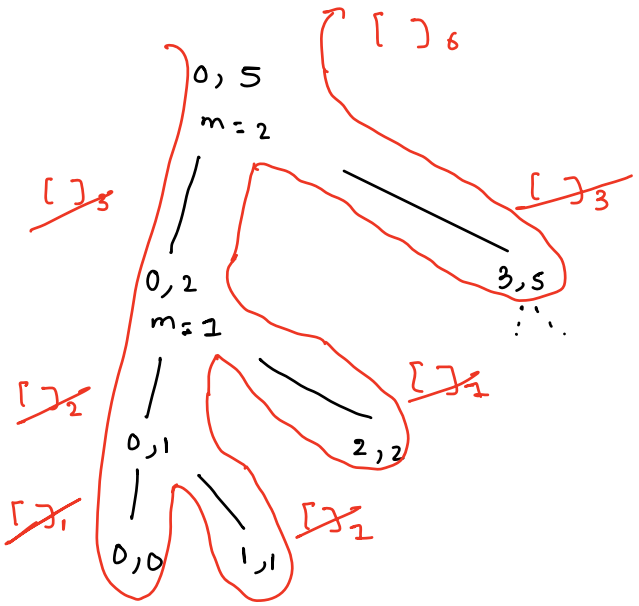
TC: $O(n \log_2 n)$

SC: storage created by you — call stack

$\not{n}$ — $\log_2 n$

SC: $O(n)$




[ ]6
0, 5
m = 2
[ ]5
[ ]3
0, 2
3, 5
m = 1
[ ]2
[ ]4
0, 1
2, 2
[ ]1
[ ]2
0, 0
1, 1

Doubts
=

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

O(nlogn)
↓
Sorting

1) freq map creation

fa

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

↳ fa[5] → freq of 5

ps

1	2	3	5	5	7	7	7	8

fa[i] => how many times "i" value is coming in A[].

max = 8

size = max + 1

```
for (int i=0; i<A.length; i++) {
    int ele = A[i];
    fa[ele] += 1;
}
```