Lecture :- Selection and Merge sort

Agenda

- Selection sort
- Merge 2 sorted arrays
- Merge sort
- Inversion count
- Stable sorting.

Selection sort [Inplace sort]

arr[] = { 2 8 4 -1 6 7 5 10 }
0 1 2 3 4 5 6 7
-1 2 8

Range

Min el [-1]	[0-7]	idx: 3	swap(arr, 0, 3);
Secmin [2]	[1,7]	idx: 3	swap(arr, 1, 3);
third min [4]	[2,7]	idx: 2	swap(arr, 2, 2);

Sorted array.

```
void selectionSort(int[] arr){
    for(i=0; i<arr.length; i++){
        int minIdx = i;
        int minValue = arr[i];
        for(j=i; j<arr.length; j++){
            if(arr[j] < minValue){
                minIdx = j;
                minValue = arr[j];
            }
        }
        swap(arr, i, minIdx);
    }
```

TC: $O(n^2)$

SC: $O(1)$

Inplace sorting — which doesn't require extra space

Qu: Given A[n] and B[m], merge these 2 sorted arrays.

Sorted arrays

A[3] = { -1 4 8 }

B[2] = { 2 9 }

finalarr[] = { -1 2 4 8 9 }

Approaches!:

1. A[n] B[m]

   res[] = [n+m]

2. copy A[n] into res[] — O(n)

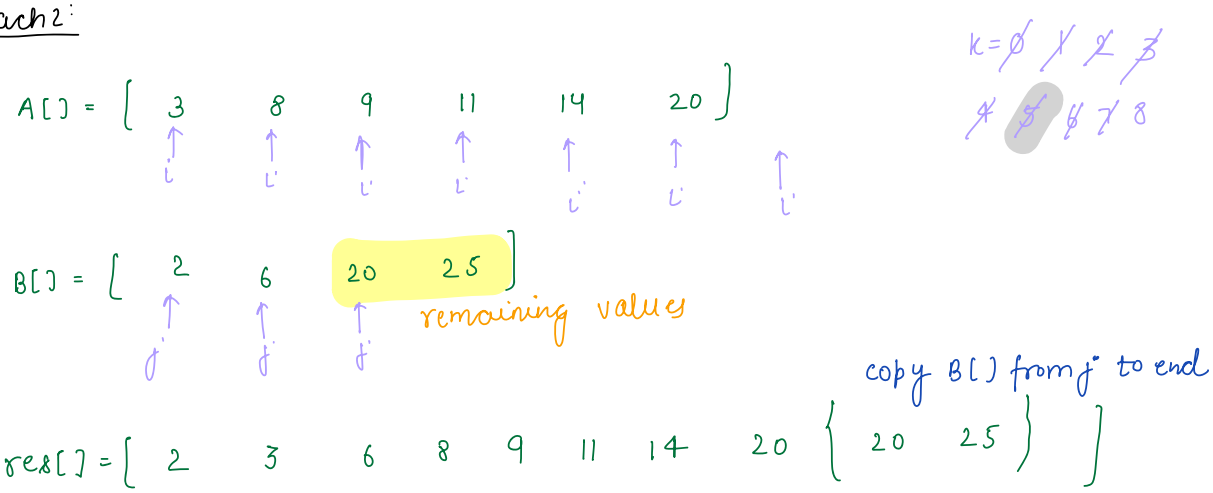
3. copy B[m] into res[] — O(m)

4. sort res[] — (n+m) log (n+m)

TC: O(n) + O(m) + O(n+m) * log(n+m)

This approach would have worked as well if A[n] & B[m] are unsorted.

Approach2:




```
int[] merge( int[] A, int[] B ) {
    int n = A.length;
    int m = B.length;
    int[] res = new int[n+m];
    int i=0, j=0, k=0;
    while( i<n && j<m ) {          // —— O(max(n,m))
        if( A[i] <= B[j] ) {
            res[k] = A[i];
            i++;
            k++;
        } else {
            res[k] = B[j];
            j++;
            k++;
        }
    }
```

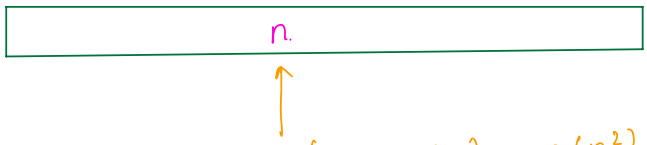
```
// Remaining el.
while( i < n) {                    — O(n)
    res[k] = A[i];
    i++;
    k++;
}

while( j < m) {                    — O(m)
    res[k] = B[j];
    j++;
    k++;
}

return result;
}
```

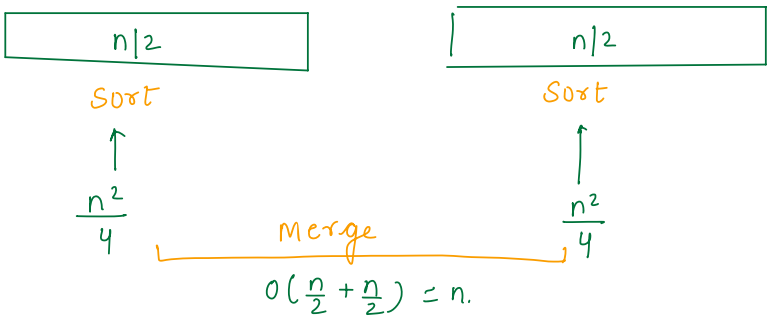
TC: $O(n+m)$

SC: $O(1)$ (or) $O(n+m)$

Merge sort

Given arr[n], sort it. Lets say $n = 100$



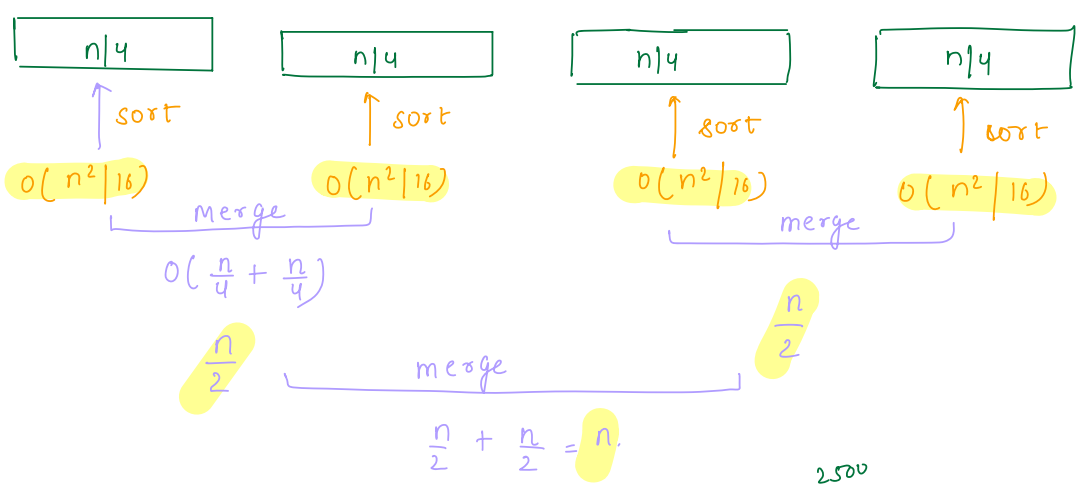


$$\frac{n^2}{4} + \frac{n^2}{4} + n = \frac{10000}{4} + \frac{10000}{4} + 100$$

(sort left, sort right, merging)

$$= 2500 + 2500 + 100$$

$$= 5100 \text{ itr}$$

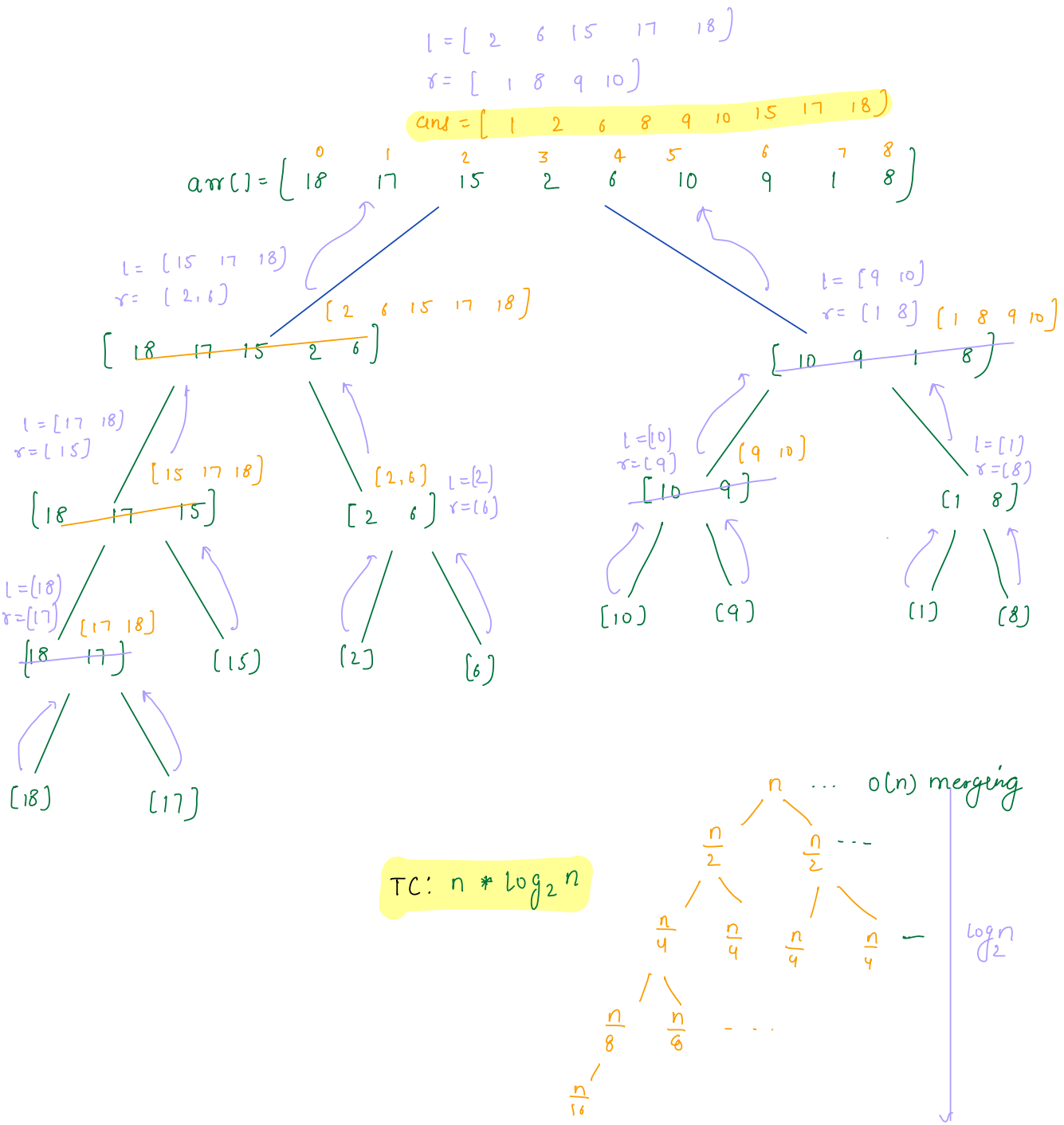


$$\frac{n^2}{16} * 4 + \underbrace{\frac{n}{2} + \frac{n}{2} + n}_{2n.} \Rightarrow \frac{n^2}{4} + 2n = \frac{\overset{2500}{\cancel{10000}}}{\cancel{4}} + 2 * 100 = 2700 \text{ itr}$$

Observation More divide the array, lesser the no. of iterations

Idea of merge sort.

Max division: until I get an array of size 1.



TC: $n * \log_2 n$

```
void mergeSort(int[] arr, int s, int e) {
    // Base case: size of arr is 1.
    if (s==e) {        // [ 8 ]   s=0, e=0
        return;
    }
    int mid = (s+e)/2;
    mergeSort(arr, s, mid);
    mergeSort(arr, mid+1, e);
    merge(arr, s, mid, e); // Implement
}

void merge(int[] arr, int s, int mid, int e) {
    left = arr[s — mid]
    right = arr[mid+1 — end]
    int res = new int[e-s+1];
    int i = s;
    int j = mid+1;
    int k = 0;
```

left = arr[s — mid]

right = arr[mid+1 — end]

```
while( i<=mid && j<=e ) {
    if( A[i] <= B[j] ) {
        res[k] = A[i];
        i++;
        k++;
    } else {
        res[k] = B[j];
        j++;
        k++;
    }
}

// Remaining el.
while( i<=mid )          — O(n)
    res[k] = A[i];
    i++;
    k++;
}
while( j<=e )            — O(m)
    res[k] = B[j];
    j++;
    k++;
}
// copy sorted res[] { s to e } back to original array
    k=0;
for( int idx = s; idx<=e; idx++ ) {
    arr[idx] = res[k];
    k++;
```

Break: 8:48 AM

3

Qu Given A[n] and B[m], count no of pairs (i,j) such that A[i] > B[j]

A[] = [7 3 5]

B[] = [2 0 6]

⟨7 2⟩ ⟨3 2⟩ ⟨5 2⟩

⟨7 0⟩ ⟨3 0⟩ ⟨5 0⟩

⟨7 6⟩

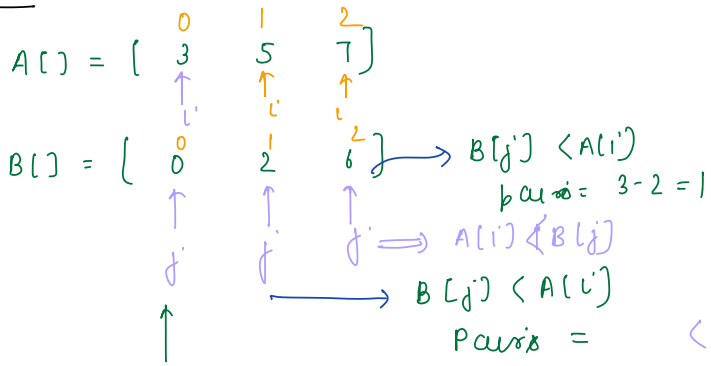
= 7 pairs

Brute force: Go all the pairs

TC: $O(n^2) \simeq O(n*m)$

SC: O(1)

Approach2:



cnt = arr.length [n] − i

⟨3,0⟩ ⟨5,0⟩ ⟨7,0⟩

pairs = n − i

3 − 0 = 3.

```
int countPairs [int[] A, int[] B) {
    int i = 0, j = 0;    pair = 0
    Arrays.sort(A);  ⟶ o(n log n)
    Arrays.sort(B);  ⟶ o(m log m)
    while(i < A.length && j < B.length){
        if( A[i] <= B[j]) {
            i++;
        } else {
            pair += A.length - i;
            j++;
        }
    }
    return pair;
}
```

A[i] > B[j]

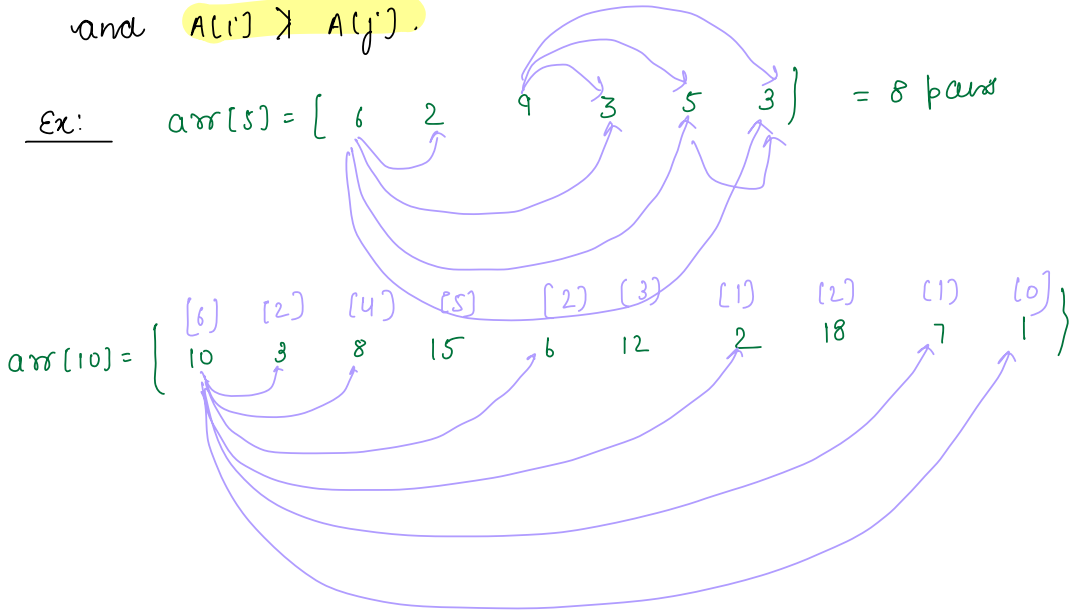
TC: $O(n \log n) + O(m \log m)$

SC: $O(1)$

Qu Inversion count

Given arr[n], find count of pairs such that **i<j** and **A[i] > A[j]**.

Ex: arr[5] = [6 2 9 3 5 3] = 8 pairs



Brute force: Go to all pairs

TC: $O(n^2)$

SC: $O(1)$

Approach2

arr[10] = [10 3 8 15 6 12 2 18 7 1]

A[] = [10 3 8 15 6]

sort

A[] = [3 6 8 10 15]

<12 2> → missed

B = [12 2 18 7 1]

sort

B = [1 2 7 12 18]

[5-0] [5-0] [5-2] [5-4]

<8 7>
<10 7>
<15 7>

<15 12>

<3 1> <3 2>
<6 1> <6 2>
<8 1> <8 2>
<10 1> <10 2>
<15 1> <15 2>

[12 2 18 7 1]

left = [12 2 18]

sort

left[] = [2 12 18]

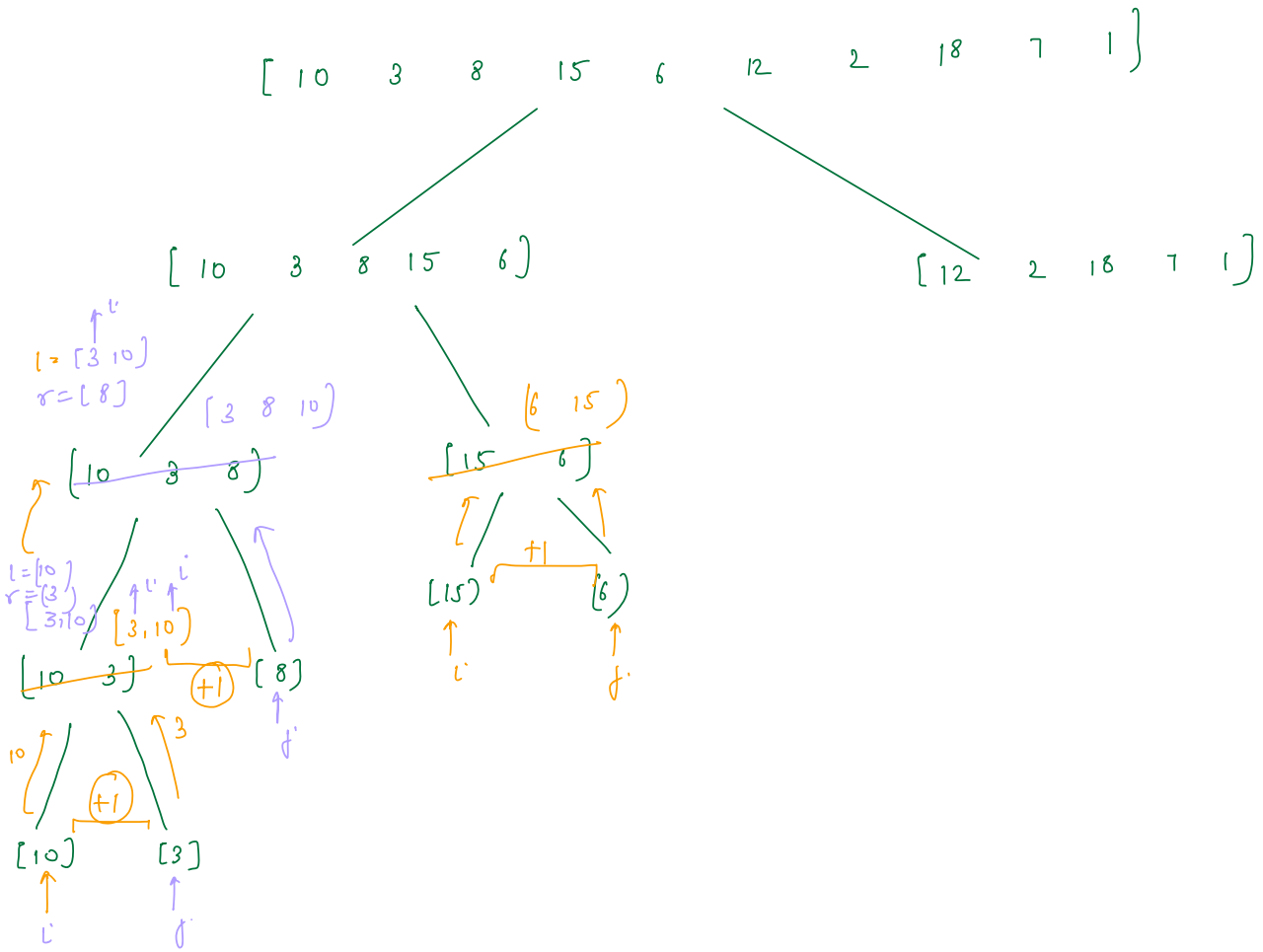
right[] = [7 1]

sort

right[] = [1 7]

+3 +2

<2 1> <12 7>
<12 1> <18 7>
<18 1>


[10 3 8 15 6 12 2 18 7 1]
[10 3 8 15 6]
[12 2 18 7 1]
[3 8 10]
[6 15]
+1
+1
+1

```
int countPairs(int[] arr, int s, int e) {
    if (s == e) {
        return 0;
    }
    mid = (s + e)/2;
    int l = countPairs(arr, s, mid);
    int r = countPairs(arr, mid+1, e);
    int mergePairs = merge(arr, s, mid, e);
    return l + r + mergePairs;
}
```

```
int merge(int[] arr, int s, int mid, int e) {
    left = arr[s — mid]
    right = arr[mid+1 — end]
    int res = new int[ e - s + 1 ];
    int i = s;
    int j = mid+1;
    int k = 0;
    int pairs = 0;
```

```
while( i<=mid && j<=e ) {
    if( A[i] <= B[j] ) {
        res[k] = A[i];
        i++;
        k++;
    } else {
        pairs += n - i;
        res[k] = B[j];
        j++;
        k++;
    }
}

// Remaining el.
while( i<=mid )          — O(n)
    res[k] = A[i];
    i++;
    k++;
}

while( j<=e )            — O(m)
    res[k] = B[j];
    j++;
    k++;
}

// copy sorted res[] { s to e } back to original array
    k=0;
for( int idx = s; idx<=e; idx++ ) {
    arr[idx] = res[k];
    k++;
}

return pairs;
}
```

Thankyou ☺

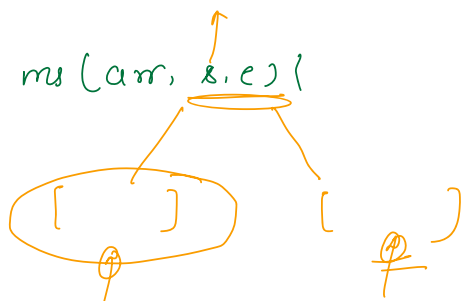
Doubts:

ms(arr, s, e)

index

s.... mid mid+1 .... e

```
void ms(arr) {
    ms(arr, 0, arr.length-1);
}

void ms(arr, s, e) {
}
```



DO this

left[] = new int[n/2];

copy n/2 in left

right[] =