Today's Content

- → Inversion Count (Hard)
- → Custom Comparator
  - (a) Sort factors
  - (b) Largest Number
  - (c) Sort Point Object

Q1 : Inversion Count [Google / Microsoft / DE Shaw]

Given A[N]. Find no. of pairs (i, j) st $i < j$ && $A_i > A_j$

Eg: ar[5] = {6, 2, 9, 3, 5}

	0	1	2	3	4
	6	2	9	3	5
	+3	+0	+2	+0	+0

(6,2) (6,3) (6,5) (9,3) (9,5) = 5 pairs

ar[10] = {10, 3, 8, 15, 6, 12, 2, 18, 7, 1}

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

= 26 pairs

BF idea: Check all pairs :-

```
int invCount (int a[]) {
    c = 0
    for( i=0; i<n; i++) {
        for(j = i+1; j<n; j++) {
            if ( a[i] > a[j] ) { c++ }
        }
    }
    return c
}
```

TC: $O(n^2)$
SC: $O(1)$

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

A i

0	1	2	3	4
10,	3,	8,	15,	6

Sort: 3 6 8 10 15

B j

5	6	7	8	9
12,	2,	18,	7,	1

Sort: 1 2 7 12 18

p1 p2

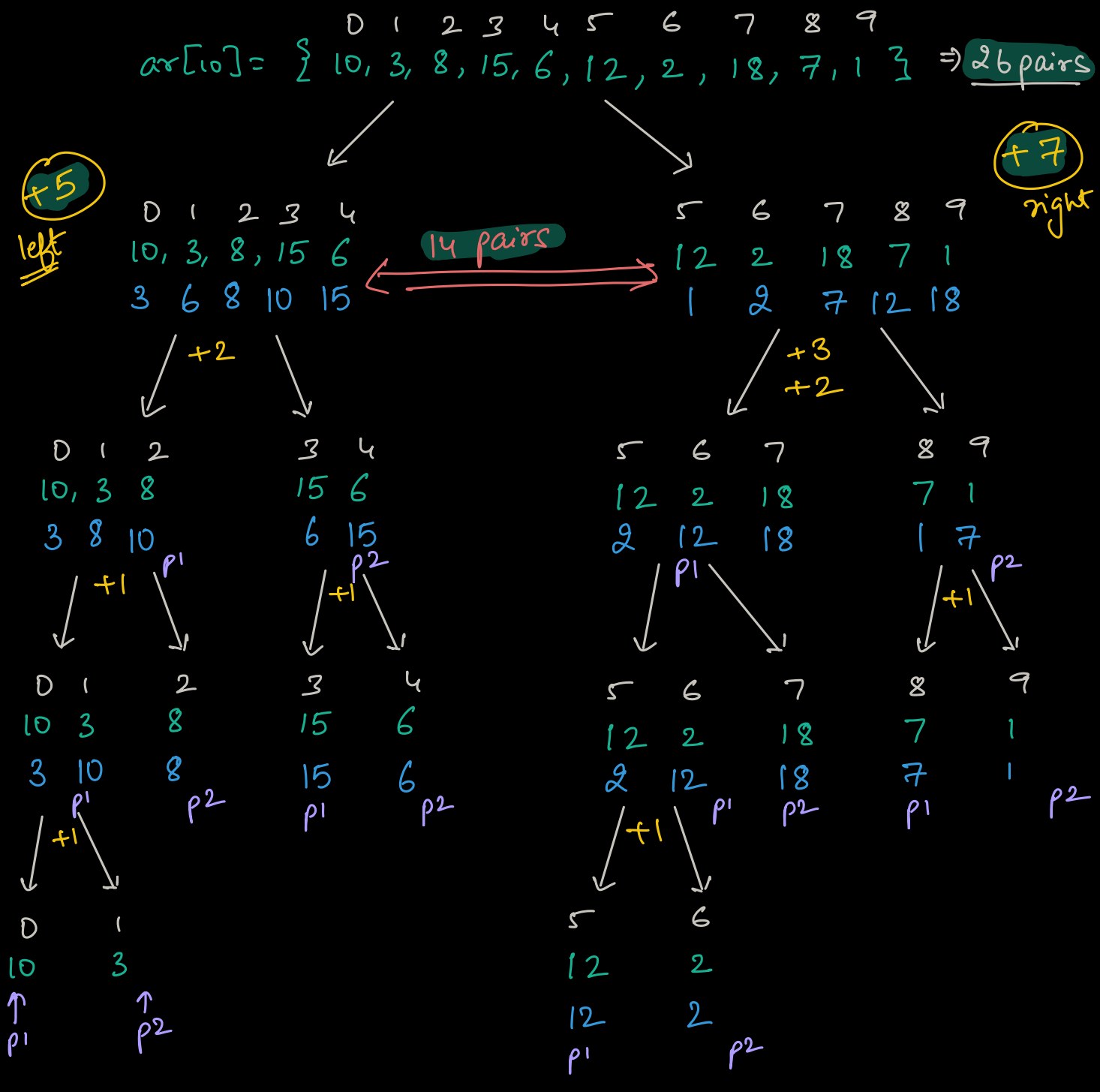
1	2	3	6	7	8	10	12	15	18
+5	+5			+3			+1		+0

⇒ 14 pairs

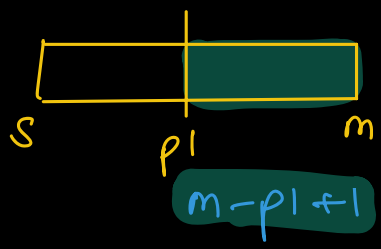
c = c + no. of elements remaining on left.

Missing Part:

Total pairs = { Total Pairs in A } + { Total Pairs in B }
+ { Total Pairs Between A & B }



```
int c = 0
void merge(int a[], int s, int m, int e) {
    p1 = s, p2 = m+1, p3 = 0
    int temp[e-s+1] // No. of elements in [s e]
    while (p1<=m && p2<=e) {
        if (a[p1] <= a[p2]) {
            temp[p3] = a[p1], p3++, p1++
        }
        else {
            temp[p3] = a[p2], p2++, p3++
            c = c + m - p1 + 1
        }
    }
    // Copy remaining elements
    while(p1 <=m) {
        temp[p3] = a[p1], p3++, p1++
    }
    while(p2 <=e) {
        temp[p3] = a[p2], p3++, p2++
    }
    // temp[] is created. Copy back to a[]
    for(i = s ; i <= e ; i++) {
        a[i] = temp[i-s]
    }
}
```



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

```
void mergesort(int a[], int s, int e) {
    if (s==e) { return }
    mid = (s+e)/2
    mergesort(a, s, mid)
    mergesort(a, mid+1, e)
    merge(a, s, mid, e)
}
```

TC: O(nlogn)
SC: O(n)

Break: 8:22 am

```
return c    // from inv. count fn.
```

Comparator Overriding [Arrays.sort() or Collections.sort()]

By overriding compare(), you can alter the way in which objects are ordered. ⇒ CUSTOM SORTING.

→ Pass 2 arguments [Data that needs to be compared]

→ Based on rules, it should tell which data should come first.

Rules:-

```
public int compare(Object a, Object b){
    ① if a should be on left of b
          return negative number (-1)
    ② if a should be on right of b
          return positive number (+1)
    ③ if no change in order
          return 0
}
```

Syntax:

```
ArrayList<Object> A[N];
Collections.sort(A, new Comparator<Object>(){
        public int compare(Object a, Object b){
            ① if a should be on left of b
                  return negative number (-1)
            ② if a should be on right of b
                  return positive number (+1)
            ③ if no change in order
                  return 0
        }
});
```

Python:

```
def mycmp(a, b):
```

① if a should be on left of b
return negative number (-1)

② if a should be on right of b
return positive number (+1)

③ if no change in order
return 0

Syntax:

```
l = []  // l is a list containing Object/int/etc.
sorted(l, key = functools.cmp_to_key(mycmp))
```

Qn: Given A[N]. Sort the array in increasing order of no. of distinct factors. If 2 elements have same factors, then number with less value should come first.

Eg: a[] = {6, 8, 9} ⇒ {9 6 8}

6 → 4, 8 → 4, 9 → 3

a[] = {12 5 10 4 7 6 9 1 3}

12 → 6, 5 → 2, 10 → 4, 4 → 3, 7 → 2, 6 → 4, 9 → 3, 1 → 1, 3 → 2

⇒ {1 3 5 7 4 9 6 10 12}

Soln: Use custom comparator.

```
ArrayList<Integer> sortfactors(ArrayList<Integer> A){
    Collections.sort(A, new Comparator<Integer>(){

        public int compare(Integer a, Integer b){
            count1 = factors(a)
            count2 = factors(b)
            if(count1 == count2){
                if(a<b){return -1}
                else if(a>b){return +1}
                else{return 0}
            }
            else if(count1 < count2){
                return -1
            }
            else{
                return +1
            }
        }

    });
}
```

Qn: Given A[N] non-ve no. as integers. Concatenate the numbers such that you can form the largest number possible (as string).

Eg: a[] = {2, 3, 9, 0} ⇒ {9 3 2 0}

a[] = {3, 30, 34, 5, 9}
9 5 34 330

a[] = {1, 10, 2, 40, 41}
41 40 2 1 10

"30" v/s "3"

1	1
a	b

x		y
a+b	v/s	b+a
303		330

* We do "a+b" or "b+a" to make string length same.
* Compare whether a should be on the left or right.

```
String largestNum(ArrayList<Integer> A) {
    Collection.sort(A, new Comparator() {
        public int comparator(Integer a, Integer b) {
            String x = String.valueOf(a) + String.valueOf(b)
            String y = String.valueOf(b) + String.valueOf(a)
            if (x < y) {
                return +1
            }
            else if (x > y) { return -1 }
            else { return 0 }
        }
    });
    // Iterate over A & convert A[i] to string & return
}
```

Qn: Given N objects where each object contains

int x

int y

Sort the objects using only x in asc. order

```
class Point {
    int x
    int y
    Point(int a, int b) {
        x = a
        y = b
    }
}
```

Eg:- (10,1) (9,8) (2,4) (7,3) (-1,0) ⇒ (-1,0) (2,4) (7,3) (9,8) (10,1)

① Create an arraylist of Point object [N].

```
for(i=0; i<N; i++) {
    a.add(new Point(x[i], y[i]))
}
```

② Sort it using custom comparator.

```
Collections.sort(a, new Comparator<Point>() {
    public int compare(Point p1, Point p2) {
        if(p1.x < p2.x) {
            return -1
        }
        else if(p1.x > p2.x) {
            return +1
        }
        else { return 0 }
    }
});
```

③ Return arraylist

```
return a
```

Doubt:

4, 5, 3, 2, 7 X

min = 2

offset = 2.

0	1	2	3	4
4	5	3	2	7
3	3	4	5	
5				
2				

✓

0	1	2	3
100	102	101	99
102	100		102
99			

99	
99	

offset