Agenda

- ✓ ◦ → Problems
- ✓ ◦ → Comparators (Java)
- ⇒ ◦ → Radix Sort (good to know)
- → Heap Sort ☆ [Later after Heaps]

Inversion Count

$$arr = \begin{array}{|ccccc|} 0 & 1 & 2 & 3 & 4 \\ \hline 3 & 1 & 5 & 6 & 2 \\ \hline \end{array}$$

Count Inversions

if Pair (i,j) forms inv

→ $i < j$

$a[i] > a[j]$

3, 1

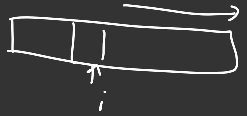
~~3, 5~~

3, 2

5, 2

} 4 inversion ← output

6', 2 ∫



Brute Force

$O(N^2)$ time

$O(1)$ space.

```
cnt = 0
for (i=0; i< n-1, i++) {
    for(j = i+1, j<n-1; j++) {

        if(a[i] > a[j]){
            cnt++;
    }       }
}
print(cnt).          // 4
```

0	1	2	3	4
3	1	5	6	2

$N = 10^6$

N^2 ? ×

$N\sqrt{N} = 10^6 . 10^3 = 10^9 > 10^8$

$N \log N = 10^6 . \log 10^6$

$= 2 \times 10^7 < 10^8$ ✓

Sorting → $N \log N$

or lesser

Divide & Conquer

CI(arr, s, e)

3	1	5	6	2

L = CI(arr, s, mid)

⇒ 3 1 5

3, 1

R = CI(arr, mid+1, e)

6 2

6, 2

i → j

Cross inversion

3 , 2

5 , 2

$ans = L + R +$ cross Inv

$= 1 + 1 + 2$

$= \boxed{4}$

merge Sort

```
int CountInvs ( arr, s, e) {

        // Base Case
        if (s >= e)
              return 0

        // Rec Case      mid = (s+e)/2
        Left  =  CountInvs(arr, s, mid);
        Right =  CountInvs(arr, mid+1, e)
        total =  Left + Right + merge(arr, s, mid, e);
        return total;
}
```

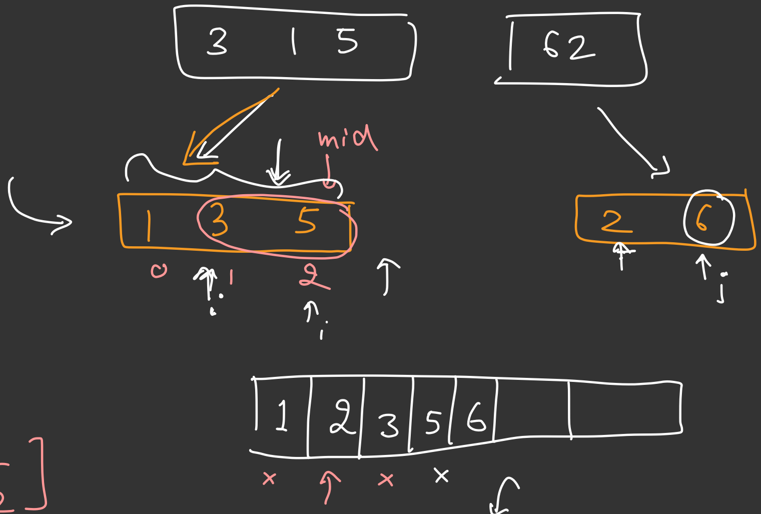
$O(N\log N)$

return cross Invs for given array.

Sorted

Merge

[3,2
5,2]



⇒ 2 cross Inversions

Cross inversions

$a(i) > a(j)$

~~cnt++~~ ?

$cnt += \underline{\underline{(mid - i + 1)}}$

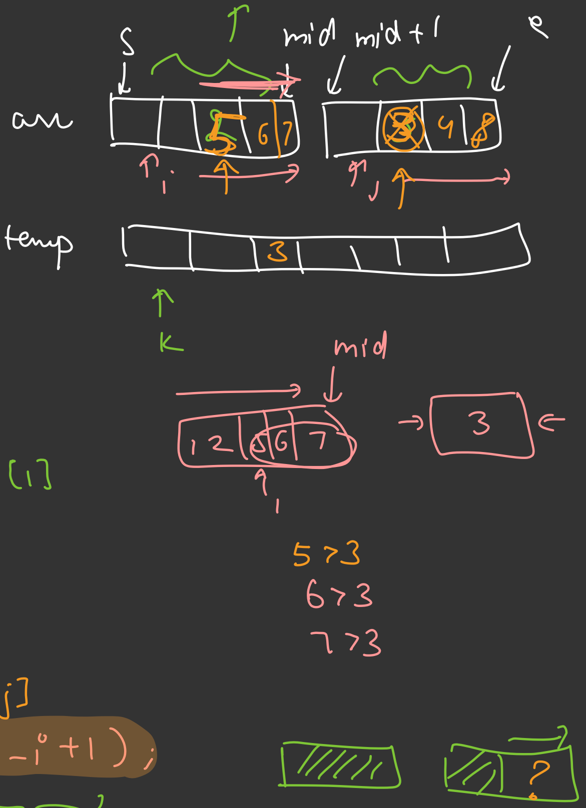
$= 2 - 1 + 1$

$= \boxed{2}$

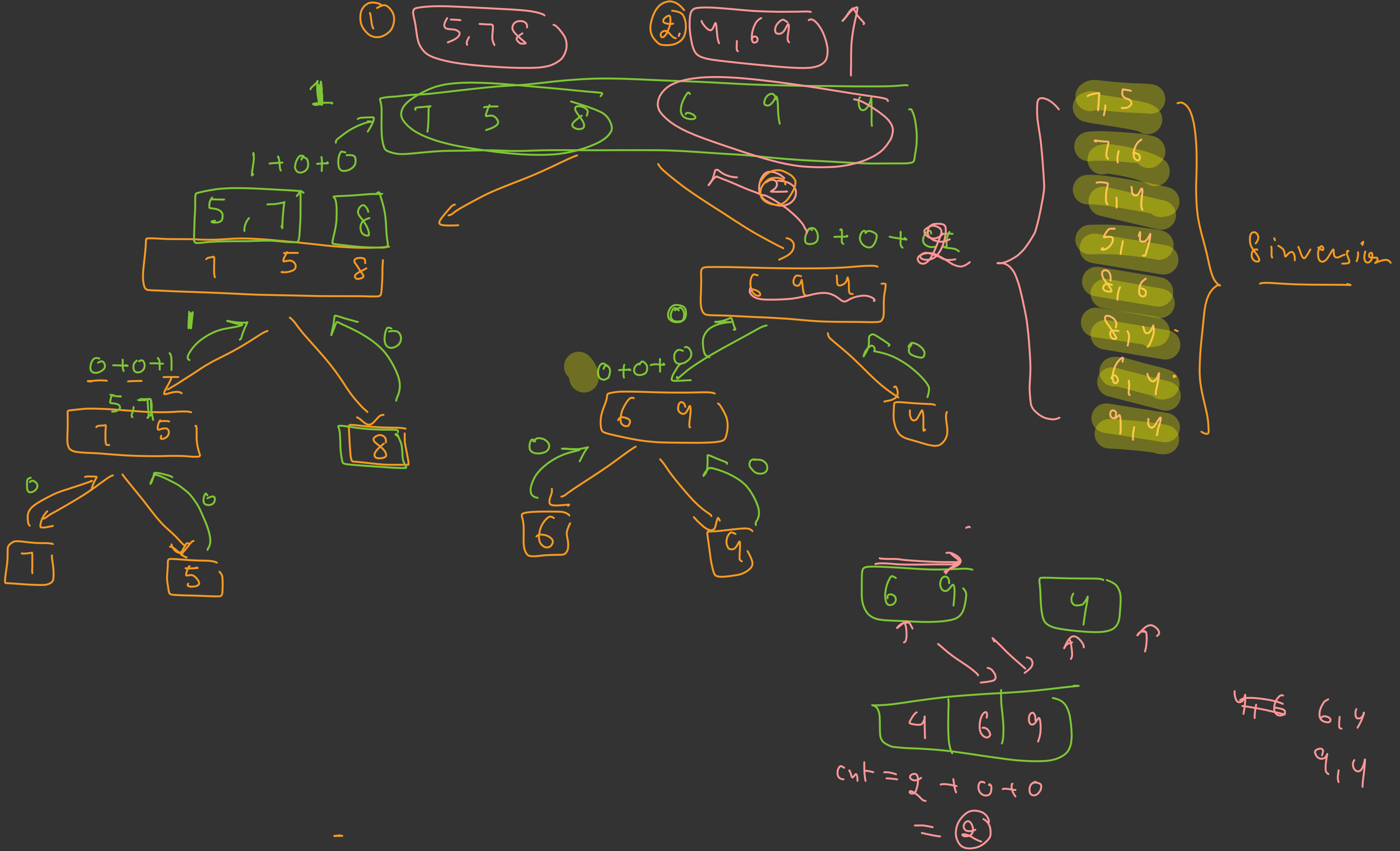
```
if (a[i] > a[j]) {

}
```

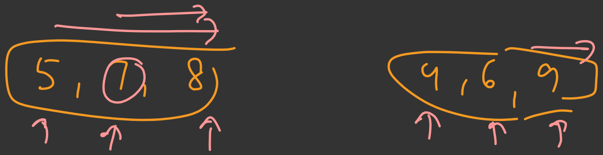
```
int   merge ( int  arr[], s, mid, e) {

      temp [ ];
        i=s  ,  K=s
        j=mid+1
       cnt=0
      while(i<=mid && j<=e){

            if (a[i] < a[j]) {
                  temp[k] = arr[i]
                  i++, K++
            }
            else{
         →        temp[k] = arr[j]
                  cnt += (mid-i+1);
                  j++, K++
            }
}
```

↑ multiple pairs in just one step



```
while(j<=e){
        temp(k) = arr(j)
        j++, k++
}

while(i<=mid){
        temp(k) = arr(i)
        i++, k++
}

Copy temp → arr

    for(i=s; i<=e; i++)
    {   arr(i) = temp(i)   }

return cnt.
    =         CI = 5      =  8
}
```


① 5,7 8
② 4,6 9
1
7 5 8 6 9 4
1+0+0
5,7 8
7 5 8
1
0+0+1
5,7
7 5
0
0
7
5
0
8
2
0+0+2
6 9 4
0
0+0+0
6 9
0
0
6
9
0
4
7,5
7,6
7,4
5,4
8,6
8,4
6,4
9,4
8 inversion
6 9
4
4 6 9
cnt = 2 + 0 + 0
= ②
~~4+6~~ 6,4
9,4



7, 6

8, 6

4	5	6	7	8	9

$cnt = 0 + 3 + 0 + 2 + 0 \quad 0 \quad 0$

$= \boxed{5}$

We are given a list of activities, each activity has a start and end time. We want perform max activities assuming max 1 activity in given time period.

Q Activity Count

many over lapping activity ⇒ Maximum activites that I can perform. assumig I can do max 1 activity at given time

→ 2 — 8 PM office

→ 1 - 2 lunch

→ 5 — 8 Movie

→ 3 — 4 Sleep

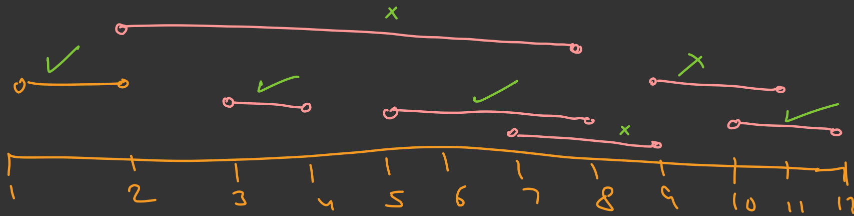
→ 7 — 9 friends

→ 10 — 12 dinner

→ 9 — 11 Scaler

Break + Thinking Time [15 mins]

10.20



Max that I can do is → $\boxed{\text{4 activites}}$

s1 — e1

s2 — e2

⋮ ⋮

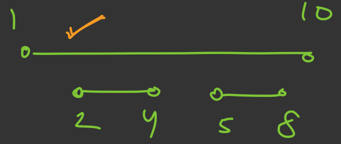
Choices

X Start time | end time | duration (e − s) X

(1) —— 10

(2)—4

(5)— 8



Picking an activity that starts early doesn't ensure the same activity ends early.

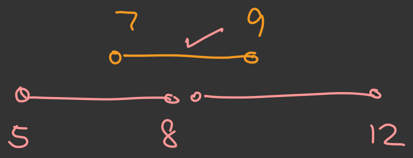
Pick an activity that ends early

More time for remaining activities

(2) 5 — 8 (3 hrs)

(3) 8 — 12 (4 hrs)

(1) 7 − 9 (2 hrs)

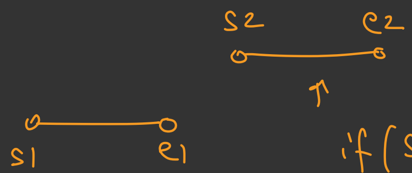


End Time

2	8	PM office	IV
1	2	lunch	I
5	8	Movie	III
3	4	Sleep	II
7	9	friends	V
10	12	dinner	VII
9	11	Scaler	VI

✓ 1 — 2	lunch	
✓ 3 — 4	$3 \geq 2$	
✓ 5 — 8	$5 \geq 4$	
2 — 8	✗	
7 — 9	✗	
9 — 11	✓	$9 \geq 11$
10 — 12	✗	

4 activities



if$(s2 \geq e1)$

① Sort the "activites" acc to end time → finishes early

"Key"

2

```
cnt = 1
endTime = list[0].end;
for( i=1 ; i<=n-1 ; i++ ) {
    if( list[i].start ≥ endTime){
            cnt ++,
            endTime = list[i].end;
    }
}
print(cnt)
```

```
class Activity{
    // string name;
    int startTime;
    int endTime;

    Activity(int start,int end){
        startTime = start;
        endTime = end;
    }
    //put a method that returns the 'key' for sorting
    int getEndTime(){
        return endTime;
    }
    void print(){
        System.out.println("start " + startTime + "endTime " + endTime);
    }
}
```

```java
public class ActivitySelection {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // create an array of objects
        Activity list[] = new Activity[n];

        //read the objects(each activity)
        for(int i=0; i<n;i++){
            int s,e;
            s = sc.nextInt();
            e = sc.nextInt();
            list[i] = new Activity(s,e);
        }

        // Algorithm (Java 8 and above)) [KeyExtractor]
        Arrays.sort(list,
Comparator.comparing(Activity::getEndTime));
```

⇒ Comparator

Interface

extract the Key

```java
        // Count
        int cnt = 1;
        int endTime = list[0].endTime;

        for(int i=1; i<=n-1; i++){
            if(list[i].startTime >= endTime){
                list[i].print();
                cnt++;
                endTime = list[i].endTime;

            }
        }
        System.out.println("count " + cnt);

    }
}
```

Radix Sort

3 Phases

Max d digit

36, 51, 64, 118, 5, 70

Couting sort

70, 51, 64, 05, 36, 118

Countis sort

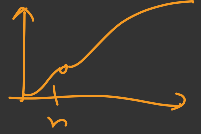
005, 118, 036, 051, 064, 070,

005, 036, 051, 064, 070, 118

Sorted Array

Tim Sort →

[Insertion Sort
→ merge sort



Base-10

1 1 8

Range → 10

Counting Sort

$O(d(n + b))$

$\log_b N0$ Base 10

https://www.geeksforgeeks.org/radix-sort/

1000 $\log_{10} 1000 = \boxed{3}$

10000 $\log_{10} 10000 = \boxed{4}$

Code

https://github.com/prateek27/java-mar-22