Nov 7, 2014

Collaborative filtering: (Netflix)

for each user: "ranking" → users are "close"
if their rankings are "close"
(partial ranking in real life)

→ distance between rankings:
→ same ranking to have dist = 0
→ reverse rankings to have max dist.

inversion

Input: n items

Rankings : $a = a_1, \ldots, a_n$

$b = b_1, \ldots, b_n$

Output: # inversions between a & b

(i,j) is an inversion if $a_i > a_j$ but $b_i \leq b_j$

$i < j$

$i, j \in [n]$

Example n=

{3, 4}

3 4

4 3

inv = 1

2 1

1 2

Ex:

i j

1 2 3

3 2 1

$\#\text{inv} = \binom{3}{2} = 3$

↖ all pairs (i,j) $i<j$ are inversions $i, j \in [3]$

Simplification: $b = 1, 2, \ldots, n$ (WLOG assume Let of items $= [n]$)

Input: $a = a_1, \ldots, a_n$ $(a_i \in [n])$

Output: # inversions

A pair (i, j) is an inversion if $a_i > a_j$ $i < j$

(Recall: same defn as seen in proof of greedy algo to min max (alones).

Ex 1: $a = \begin{cases} 1, 2, 3, \ldots, n & \#\text{inv} = 0 \\ n, n-1, \ldots, 1 & \#\text{inv} = \binom{n}{2} \end{cases}$ ← max # inv.

Q: Naive algo to solve this problem?

A: Check all pairs $(i<j)$ → check if $a_i > a_j$

→ $O(n^2)$

→ lists all inversion ($\Omega(n^2)$ for any algo that explicitly lists all inversions)

Goal: $O(n \log n)$ algo to count the # inv

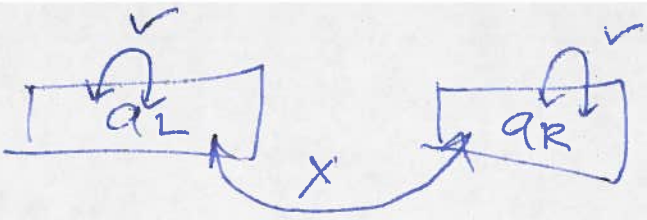
Count Inv (a, n)

If $n = 1$ return 0

If $n = 2$ If $a_1 > a_2$ return 1 else return 0

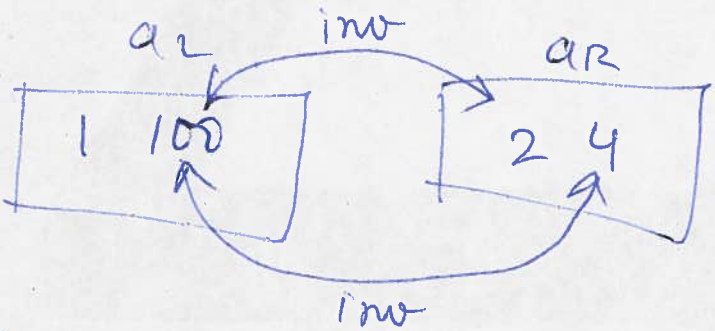
$a_L = a_1, \ldots, a_{\lceil \frac{n}{2} \rceil}$

$a_R = a_{\lceil \frac{n}{2} \rceil + 1}, \ldots, a_n$

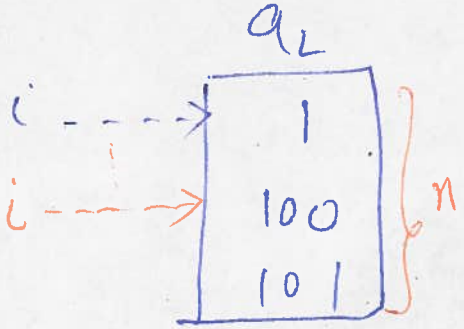


Return CountInv(a_L, $\lceil \frac{n}{2} \rceil$) + CountInv($a_R$, $n - \lceil \frac{n}{2} \rceil$)

$n = 4$



Crux: If a_L & a_R are sorted, count # crossing inversions.



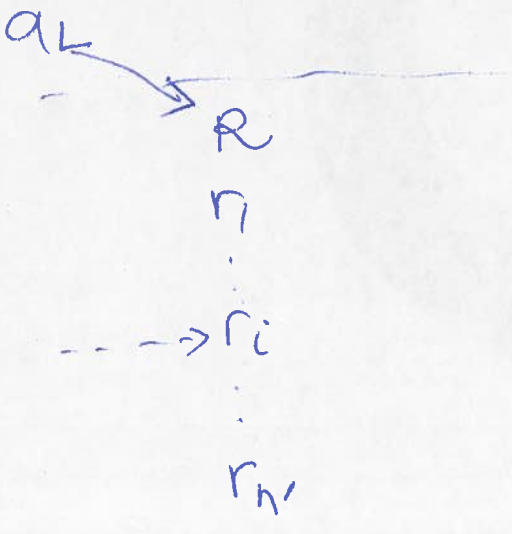
Step 2: $1 \leq 2$

$\Rightarrow$ 1 is not part of ANY inversion

Step 2: $100 \geq 2$

$\Rightarrow$ 2 participates in 2 inversions

$= n - i + 1$



Case 1: $r_i \leq s_j$

$\Rightarrow$ $(r_i, s_{j'})$ $\forall j' \geq j$ is NOT an inv

Case 2: $r_i > s_j$

$n' - i + 1 \leq$ # inv $\{ (r_{i'}, s_j) \; \forall i' \geq i$ IS an inv

Count # pairs (i,j) s.t $r_i > s_j$.

```
c ← 0
i, j ← 1
If r_i ≤ s_j
    i++
Else
    c += n'-i+1
    j++
```

Q: Why would a_L & a_R be sorted?

2nd main idea: Sort a_L & a_R in the recursive steps.

→ stronger problem: count # inv in a & also sort a

"Patch up" problem: Given sorted a_L & a_R count # crossing inv AND merge a_L & a_R into a sorted array

MERGE-COUNT (R, S)

```
i, j ← 1          O(n'+m) time
c ← 0
While i ≤ n' & j ≤ m
    If r_i ≤ s_j
        Output r_i : i++
    Else
        c += n'-i+1
        Output s_j
        j++
```

Ex: Correctness

If one list is non-empty output that list.