5.3 Counting Inversions

Counting Inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Q. How can we measure the distance between two rankings?

- My rank: $b_1, b_2, \ldots, b_n$ with $b_1 < b_2 < \ldots < b_n$
- Your rank: $a_1, a_2, \ldots, a_n$.

Songs

	A	B	C	D	E
You	1	3	4	2	5
Me	1	2	3	4	5

TU Delft

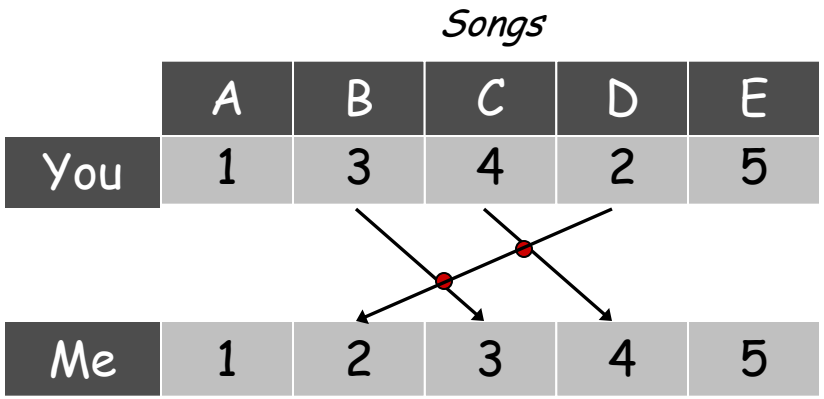
Counting Inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $b_1, b_2, \ldots, b_n$ with $b_1 < b_2 < \ldots < b_n$
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.



Inversions

3-2, 4-2

Q. Give a brute-force algorithm to calculate the number of inversions.

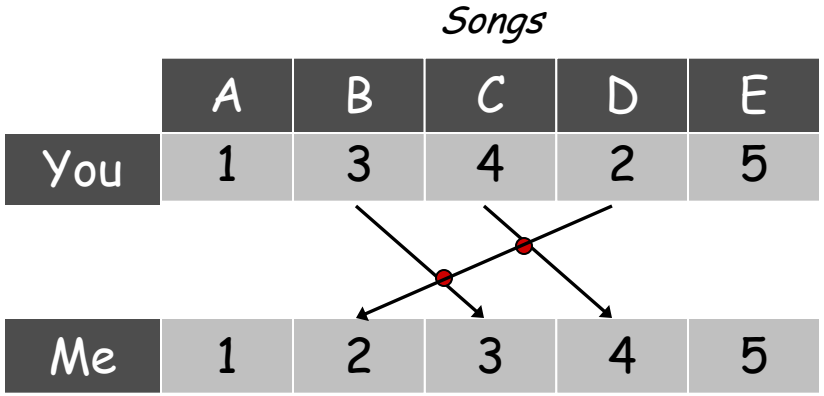
Counting Inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $b_1, b_2, \ldots, b_n$ with $b_1 < b_2 < \ldots < b_n$
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if i < j, but $a_i > a_j$.



Inversions

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j (similar to *bubble sort*)

Applications

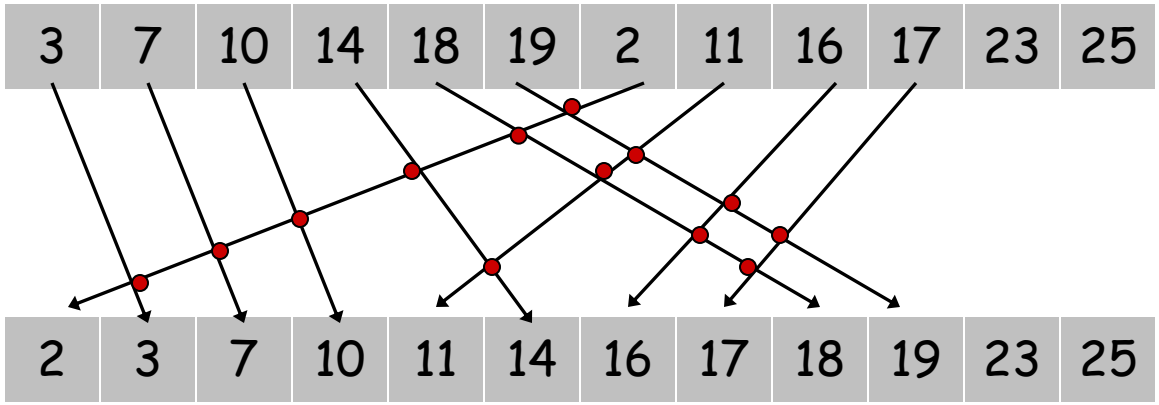
Applications.

- Voting theory.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Collaborative filtering (amazon.com, restaurants, movies)

TU Delft

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.



Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

Divide: $O(1)$.

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

Divide: $O(1)$.

Conquer: $2T(n / 2)$

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

3	7	10	14	18	19

2	11	16	17	23	25

Divide: $O(1)$.

Conquer: $2T(n / 2)$

2	3	7	10	11	14	16	17	18	19	23	25

Combine: ???

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16	17	18	19	23	25

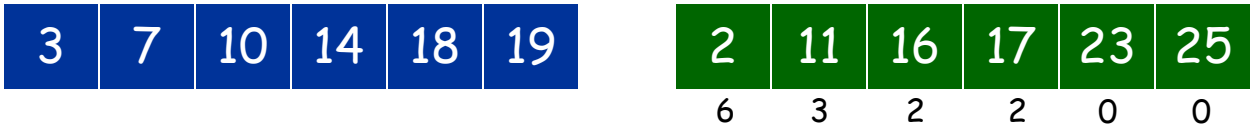
TU Delft

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant



13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

Merge: O(n)

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (r_A, A) ← Sort-and-Count(A)
   (r_B, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = r_A + r_B + r and the sorted list
L
}
```