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: 1, 2, ..., n.
- Your rank: $a_1, a_2, ..., a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

### Songs

<table>
<thead>
<tr>
<th>Me</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Inversions

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j.
Applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

\[
\begin{array}{cccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7
\end{array}
\]

Divide: \(O(1)\).

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7
\end{array}
\]
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: \( 2T(n / 2) \)

- 5 blue-blue inversions
- 8 green-green inversions

| 1 | 5 | 4 | 8 | 10 | 2 | 6 | 9 | 12 | 11 | 3 | 7 |

5-4, 5-2, 4-2, 8-2, 10-2   6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7
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.

\[
\begin{array}{cccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7
\end{array}
\]

Divide: \( O(1) \).

\[
\begin{array}{cccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7
\end{array}
\]

5 blue-blue inversions
8 green-green inversions

Conquer: \( 2T(n / 2) \)

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.
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.

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

Count: $O(n)$

Merge: $O(n)$

$T(n) \leq T\left(\left\lfloor n/2 \right\rfloor \right) + T\left(\left\lceil n/2 \right\rceil \right) + O(n) \Rightarrow T(n) = O(n \log n)$
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
}
Merge-and-Count Algorithm

Merge-and-Count(\(A,B\))

Maintain a \textit{Current} pointer into each list, initialized to point to the front elements

Maintain a variable \textit{Count} for the number of inversions, initialized to 0

While both lists are nonempty:

\begin{itemize}
  \item Let \(a_i\) and \(b_j\) be the elements pointed to by the \textit{Current} pointer
  \item Append the smaller of these two to the output list
  \item If \(b_j\) is the smaller element then
    \begin{itemize}
      \item Increment \textit{Count} by the number of elements remaining in \(A\)
    \end{itemize}
  \item EndIf
\end{itemize}

Advance the \textit{Current} pointer in the list from which the smaller element was selected.

EndWhile

Once one list is empty, append the remainder of the other list to the output

Return \textit{Count} and the merged list