Divide and Conquer: Counting Inversions
Rank Analysis

- **Collaborative filtering**
  - matches your preference (books, music, movies, restaurants) with that of others
  - finds people with similar tastes
  - recommends new things to you based on purchases of these people

- **Meta-search tools**
  - same query to many search engines
  - synthesize result by looking for similarities of resulting rankings

- The basis: compare the similarity of two rankings
Given numbers 1 to n (the things) rank these according to your preference
  ■ You get some permutation of 1..n
  ■ Compare to someone else's permutation

**Extreme similarity**
  ■ somebody else's ranking is exactly the same

**Extreme dissimilarity**
  ■ somebody else's ranking is exactly the opposite

**In the middle:**
  ■ count the number of out of place rankings
Count the number of **inversions** of a ranking

- \( r_1, r_2, \ldots, r_n \)
- count the number of out of order pairs
  - \( i < j \quad r_i > r_j \)

- eg: 2 1 4 3 5
- 2 inversions: (2,1) (4,3)

Why is this synonymous with comparing two different rankings?

Because we can re-number, such that one of the rankings becomes 1,2,...,n
Visualizing inversions

zero inversions

1  2  3  4  5

1  2  3  4  5

one inversion

2  1  3  4  5

1  2  3  4  5
Visualizing inversions

how many? 3 2 1 4 5

enumerate them

1 2 3 4 5

how many? 5 2 3 4 1

1 2 3 4 5
Does Bubble sort count inversions?
Selection sort?
Insertion sort?
These are $O(n^2)$

Do these sorts on:
and see what happens
Do bubble sort, show each swap, count inversions

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5
Can we do better?

Notice: there are potentially \( n \times (n-1)/2 \) inversions. **WHY?**

Bubble and insertion sort count each individual inversion.

To do better we must not count each individual inversion.

Think of merge sort:

- in merge sort we do not swap all elements that are out of order with each other, we make larger distance "swaps"
- if we can merge sort and keep track of the number of inversions we may get an \( O(n \log n) \) algorithm
sort [4 2 3 5 1]

- sort LEFT: [4 2 3]
  - sort left: [4 2] → [2 4]: 1 inversion
  - sort right: [3]
  - merge(left, right) → [2 3 4] 1 inversion (3 jumps over 4)

- sort RIGHT: [5 1] → [1 5] 1 inversion

- merge(LEFT, RIGHT) → [1 2 3 4 5]
  3 inversions (1 jumps over 2, 3 & 4)

Total inversions: 1 + 1 + 1 + 3 = 6 (go check the visualization)
The algorithm

While merging in merge sort keep track of the number of inversions.
When merging an element from left: no inversions added
When merging an element from right: how many inversions added?

As many elements as are remaining in left, because the element from the right jumps over them
Counting Inversions: Algorithm

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, R) ← Merge-and-Count(A, B)
   return r = r_A + r_B + r and the sorted list R

Merge-and-Count(L,R)
   count = 0
   while L and R not empty:
       append smallest of Li and Rj to result
       if Rj smallest
           add number of elements remaining in L to count
           if one list empty
               append the other one to result
   return count, result
Running time

Just like merge sort, the sort and count algorithm running time satisfies:

\[ T(n) = 2 \ T(n / 2) + cn \]

Running time is therefore \( O(n \log n) \)
Repeated substitution

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = cn \log_2 n$.

\[
T(n) = \begin{cases} 
    c & \text{if } n = 1 \\
    2T(n/2) + cn & \text{otherwise}
\end{cases}
\quad \text{sorting both halves}
\quad \text{merging}
\]

For $n > 1$:

\[
T(n) = 2T(n/2) + cn \\
= 4T(n/4) + cn + 2n/2 \\
= 8T(n/8) + cn + cn + 4cn/4 \\
\vdots \\
= 2^\log_2 n T(1) + cn + \cdots + cn \\
= O(n \log_2 n)
\]
mergesort: Recurrence Analysis

\[ f(n) = a \cdot f\left(\frac{n}{b}\right) + cn^d \]

\[ a = \]
\[ b = \]
\[ d = \]
\[ O(\?) \]

\[ f(n) = \begin{cases} 
O(n^d) & \text{if } a < b^d \\
O(n^d \log n) & \text{if } a = b^d \\
O(n^{\log_b a}) & \text{if } a > b^d 
\end{cases} \]
Merge and Count

Merge and count step.

- Combine two sorted halves into sorted whole.

Two sorted halves:

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>11</th>
<th>16</th>
<th>17</th>
<th>23</th>
<th>25</th>
</tr>
</thead>
</table>

Auxiliary array:

<table>
<thead>
<tr>
<th>2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Total: 6
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

Total: 6

two sorted halves

auxiliary array
Merge and Count

Merge and count step.
  ■ Combine two sorted halves into sorted whole.

Total: 6
Merge and count step.

- Combine two sorted halves into sorted whole.
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

```
3  7  10  14  18  19
```

```
2  11  16  17  23  25
```

Total: 6 + 3
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

![Diagram: Merge and Count]

- Two sorted halves: 3, 7, 10, 14, 18, 19 and 2, 11, 16, 17, 23, 25
- Auxiliary array: 2, 3, 7, 10, 11, 14
- Total: 6 + 3
Merge and Count

Merge and count step.
  - Combine two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\quad \downarrow \quad
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & \end{array}
\quad \text{two sorted halves}
\]

\[
\begin{array}{cccccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & \quad \text{auxiliary array}
\end{array}
\]

Total: \(6 + 3 + 2\)
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

![Diagram of two sorted halves merging into an auxiliary array.](image)

Total: $6 + 3 + 2 + 2$
Merge and count step.

- Given two sorted halves, count number of inversions where $a_i$ and $a_j$

Combine two sorted halves into sorted whole.

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

Total: $6 + 3 + 2 + 2$
Merge and Count

Merge and count step.
- Given two sorted halves, count number of inversions where $a_i$ and $a_j$ are in different halves.
- Combine two sorted halves into sorted whole.

```
  3  7 10 14 18 19
  2 11 16 17 23 25
  6  3  2  2
```

two sorted halves

```
  2  3  7 10 11 14 16 17 18 19
```

auxiliary array

Total: $6 + 3 + 2 + 2$
Merge and count step.
- Given two sorted halves, count number of inversions where \(a_i\) and \(a_j\) are in different halves.
- Combine two sorted halves into sorted whole.

```
first half exhausted

3  7  10  14  18  19

2  11  16  17  23  25

2  3  7  10  11  14  16  17  18  19

Total: 6 + 3 + 2 + 2
```
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

![Diagram](https://via.placeholder.com/150)

```
3 7 10 14 18 19
2 11 16 17 23 25
6 3 2 2 0
```

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

Total: \(6 + 3 + 2 + 2 + 0\)
Merge and Count

Merge and count step.
- Combine two sorted halves into sorted whole.

Total: $6 + 3 + 2 + 2 + 0 + 0$