Algorithms: COMP3121/3821/9101/9801

Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales

MORE DIVIDE AND CONQUER ALGORITHMS

Counting the number of inversions

- Assume that you have m users ranking the same set of n movies. You want to determine for any two users A and B how similar their tastes are (for example, in order to make a recommender system).

Counting the number of inversions

- Assume that you have m users ranking the same set of n movies. You want to determine for any two users A and B how similar their tastes are (for example, in order to make a recommender system).

- How should we measure the degree of similarity of two users A and B?

Counting the number of inversions

- Assume that you have m users ranking the same set of n movies. You want to determine for any two users A and B how similar their tastes are (for example, in order to make a recommender system).

- How should we measure the degree of similarity of two users A and B?

- Lets enumerate the movies on the ranking list of user B assigning to the top choice of user B index 1, assign to his second choice index 2 and so on.

Counting the number of inversions

- Assume that you have m users ranking the same set of n movies. You want to determine for any two users A and B how similar their tastes are (for example, in order to make a recommender system).

- How should we measure the degree of similarity of two users A and B?

- Lets enumerate the movies on the ranking list of user B assigning to the top choice of user B index 1, assign to his second choice index 2 and so on.

- For the i^{th} movie on $B's$ list we can now look at the position (i.e., index) of that movie on $A's$ list, denoted by a_i.

Counting the number of inversions

- Assume that you have m users ranking the same set of n movies. You want to determine for any two users A and B how similar their tastes are (for example, in order to make a recommender system).
- How should we measure the degree of similarity of two users A and B?
- Lets enumerate the movies on the ranking list of user B assigning to the top choice of user B index 1, assign to his second choice index 2 and so on.
- For the i^{th} movie on $B's$ list we can now look at the position (i.e., index) of that movie on $A's$ list, denoted by a_i.

Counting the number of inversions

- Thus, if the first listed movie on $B's$ list is the second movie on $A's$ list, then $a_1 = 2$; if the second choice of user B is the fifth choice of user A, then $a_2 = 5$, and so on.

Counting the number of inversions

- Thus, if the first listed movie on $B's$ list is the second movie on $A's$ list, then $a_1 = 2$; if the second choice of user B is the fifth choice of user A, then $a_2 = 5$, and so on.

- A good measure of similarity of these two users is the total number of *inversions*, i.e., total number of pairs of movies i, j such that movie i precedes movie j on $B's$ list but movie j is higher up on $A's$ list than the movie i.

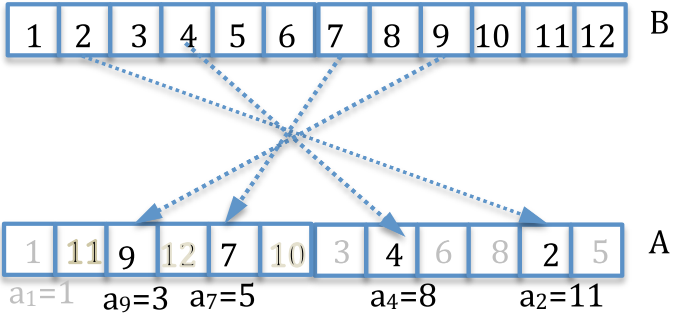
Counting the number of inversions

- Thus, if the first listed movie on $B's$ list is the second movie on $A's$ list, then $a_1 = 2$; if the second choice of user B is the fifth choice of user A, then $a_2 = 5$, and so on.

- A good measure of similarity of these two users is the total number of *inversions*, i.e., total number of pairs of movies i, j such that movie i precedes movie j on $B's$ list but movie j is higher up on $A's$ list than the movie i.

- In other words, we count the number of pairs of movies i, j such that $i < j$ (movie i precedes movie j on $B's$ list) but $a_i > a_j$ (movie i is in the position a_i on $A's$ list which is after the position a_j of movie j on $A's$ list.

Counting the number of inversions



For example 1 and 2 do not form an inversion because $a_1 < a_2$ ($a_1 = 1$ and $a_2 = 11$ because a_1 is on the first and a_2 is on the eleventh place in A); However, for example 4 and 7 do form an inversion because $a_7 < a_4$ ($a_7 = 5$ because a_7 is on the fifth place in A and $a_4 = 8$)

Counting the number of inversions

- An easy way to count the total number of inversions between two lists is by looking at all pairs $i < j$ of movies on one list and determining if they are inverted in the second list, but this would produce a quadratic time algorithm, $T(n) = \Theta(n^2)$.

Counting the number of inversions

- An easy way to count the total number of inversions between two lists is by looking at all pairs $i < j$ of movies on one list and determining if they are inverted in the second list, but this would produce a quadratic time algorithm, $T(n) = \Theta(n^2)$.

- We now show that this can be done in a much more efficient way, in time $O(n \log n)$, by applying a Divide-And-Conquer strategy.

Counting the number of inversions

- An easy way to count the total number of inversions between two lists is by looking at all pairs $i < j$ of movies on one list and determining if they are inverted in the second list, but this would produce a quadratic time algorithm, $T(n) = \Theta(n^2)$.

- We now show that this can be done in a much more efficient way, in time $O(n \log n)$, by applying a DIVIDE-AND-CONQUER strategy.

- Clearly, since the total number of pairs is quadratic in n, we cannot afford to inspect all possible pairs.

Counting the number of inversions

- An easy way to count the total number of inversions between two lists is by looking at all pairs $i < j$ of movies on one list and determining if they are inverted in the second list, but this would produce a quadratic time algorithm, $T(n) = \Theta(n^2)$.

- We now show that this can be done in a much more efficient way, in time $O(n \log n)$, by applying a DIVIDE-AND-CONQUER strategy.

- Clearly, since the total number of pairs is quadratic in n, we cannot afford to inspect all possible pairs.

- The main idea is to tweak the MERGE-SORT algorithm.

Counting the number of inversions

- An easy way to count the total number of inversions between two lists is by looking at all pairs $i < j$ of movies on one list and determining if they are inverted in the second list, but this would produce a quadratic time algorithm, $T(n) = \Theta(n^2)$.

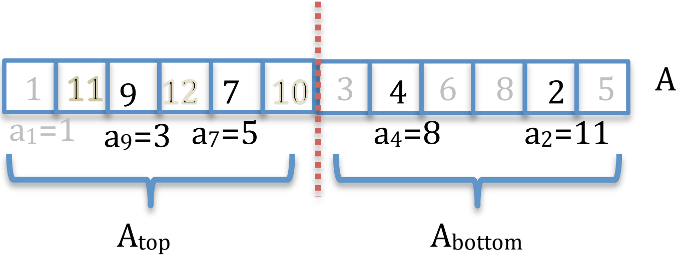
- We now show that this can be done in a much more efficient way, in time $O(n \log n)$, by applying a DIVIDE-AND-CONQUER strategy.

- Clearly, since the total number of pairs is quadratic in n, we cannot afford to inspect all possible pairs.

- The main idea is to tweak the MERGE-SORT algorithm.

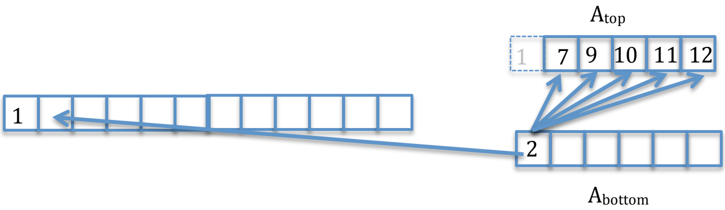
Counting the number of inversions

- We split the list $\langle a_1, a_2, \ldots, a_n \rangle$ into two (approximately) equal parts $A_{top} = \langle a_1, \ldots, a_{\lfloor 2/n \rfloor} \rangle$ and $A_{bottom} = \langle a_{\lfloor 2/n \rfloor + 1}, \ldots, a_n \rangle$.
- Note that the total number of inversions in array A is equal to the sum of the number of inversions $I(A_{top})$ in A_{top} (such as 9 and 7) plus the number of inversions $I(A_{bottom})$ in A_{bottom} (such as 8 and 11) plus the number of inversions $I(A_{top}, A_{bottom})$ across the two halves (such as 7 and 4).

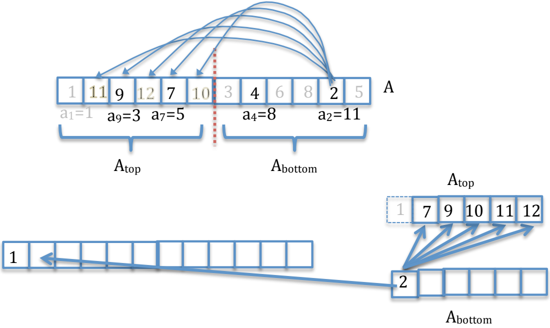


Counting the number of inversions

- We now recursively sort arrays A_{top} and A_{bottom} and obtain the number of inversions $I(A_{top})$ in the sub-array A_{top} and the number of inversions $I(A_{bottom})$ in the sub-array A_{bottom};
- We now merge the two sorted arrays A_{top} and A_{bottom} while counting the number of inversions which are across the two sub-arrays, i.e., such that $i < j$ but such that $a_i \in A_{bottom}$ and $a_j \in A_{top}$:



Counting the number of inversions



- We now merge the two sorted arrays A_{top} and A_{bottom} while counting the number of inversions $I(A_{top}, A_{bottom})$ which are across the two sub-arrays.
- When the next smallest element among all elements in both arrays is an element in A_{bottom}, such an element clearly is in an inversion with all the remaining elements in A_{top} and we add the total number of elements remaining in A_{top} to the current value of the number of inversions across A_{top} and A_{bottom}.

Counting the number of inversions

- Whenever the next smallest element among all elements in both arrays is an element in A_{top}, such an element clearly is not involved in any inversions across the two arrays (such as 1, for example).
- After the merging operation is completed, we obtain the total number of inversions $I(A_{top}, A_{bottom})$ across A_{top} and A_{bottom}.
- The total number of inversions $I(A)$ in array A is finally obtained as:

$$I(A) = I(A_{top}) + I(A_{bottom}) + I(A_{top}, A_{bottom})$$