Counting Inversions

David Fernández-Baca

Computer Science 311
Iowa State University

February 8, 2022

Inversions

Definition

Let A be an array consisting of n numbers. An inversion of A is a pair (i, j) where $0 \leq i < j \leq n-1$ such that $A[i] > A[j]$

Example

The array

$$A = \langle 2, 3, 1, 5, 4 \rangle$$

has 3 inversions,

$$(0, 2), (1, 2), (3, 4),$$

corresponding to

$$2 \leftrightarrow 1, 3 \leftrightarrow 1, 5 \leftrightarrow 4.$$

The Inversion Counting Problem

Given: An array A consisting of n numbers.

Return: The number of inversions of A.

Collaborative Filtering

- Streaming service tries to match Alice's movie preferences with others'.
- Alice ranks n movies.
- Streaming service consults database to find people with similar tastes.
- To measure similarity between Alice's ranking and Bob's ranking, take Alice's ranking as a reference and count number of inversions in Bob's ranking.
- For example, if

Alice's ranking: $\langle 1, 2, 3, 4, 5 \rangle$ Bob's ranking: $\langle 2, 3, 1, 5, 4 \rangle$,

then there are 3 inversions, $(0, 2), (1, 2), (3, 4)$, but if

Alice's ranking: $\langle 1, 2, 3, 4, 5 \rangle$ Carol's ranking: $\langle 1, 3, 2, 5, 4 \rangle$,

then there are 2 inversions, $(1, 2), (3, 4)$.

```
NaïveInversionCount(A)
    n = A.length
    count = 0
    for i = 0 to n − 1 do
        for j = i + 1 to n − 1 do
            if A[i] > A[j] then
                count++
    return count
```

Fact

`NaïveInversionCount(`A`)` *computes the number of inversions in* A *in* $\Theta(n^2)$ *time.*

Next

A $O(n \log n)$ divide-and-conquer algorithm based on `MergeSort`.

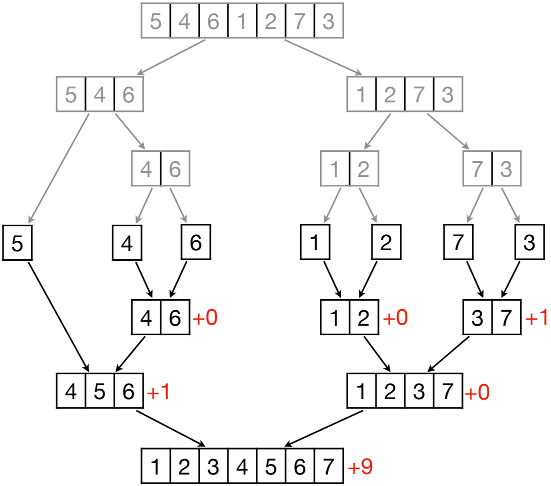
Using Divide-and-Conquer

- Split A into two arrays B and C with $\approx n/2$ elements each.
- Recursively sort B and C and, in doing so, also count the numbers r_B and r_C of inversions in B and C.
- Merge B and C to obtain a sorted version of A and, while merging, count the number r_{BC} of inversions between B and C.
- Then,

$$\#(\text{Inversions in } A) = r_B + r_C + r_{BC}.$$

- Return $r_B + r_C + r_{BC}$ and A.

Example



11 inversions

```
SortAndCount(A)
    n = A.length
    if n == 1 then
        return ⟨0, A⟩
    B = ⟨A[0], ..., A[⌊n/2⌋ − 1]⟩
    C = ⟨A[⌊n/2⌋], ..., A[n − 1]⟩
    ⟨r_B, B⟩ = SortAndCount(B)
    ⟨r_C, C⟩ = SortAndCount(C)
    ⟨r_BC, A⟩ = MergeAndCount(B, C)
    return ⟨r_B + r_C + r_BC, A⟩
```

Counting Inversions While Merging

Focus on `Merge`'s **while** loop.

```
Merge(B, C)
    p = B.length, q = C.length
    create an empty array D of length p + q
    i = 0, j = 0
    while i < p and j < q do
        if B[i] ≤ C[j] then
            append B[i] to D
            i++
        else
            append C[j] to D
            j++
    if i ≥ p then
        for k = j to q − 1 do  append C[k] to D
    else
        for k = i to p − 1 do  append B[k] to D
    return D
```

Counting Inversions While Merging

Within **while** loop, maintain an inversion counter (initialized to 0).

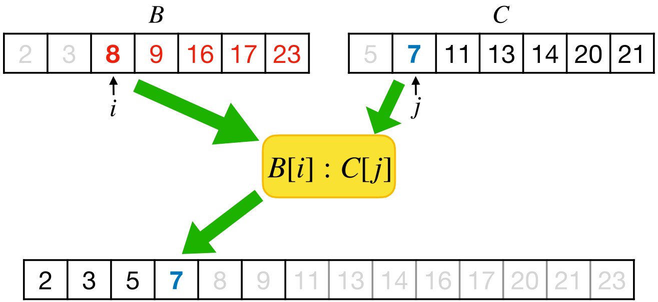
- If $B[i] \leq C[j]$, then $B[i]$ is not inverted with any of $C[j], \ldots, C[q]$.

  $\Rightarrow$ no need to increment inversion count.

- If $B[i] > C[j]$, then $C[j]$ is inverted with $B[i], \ldots, B[p]$.

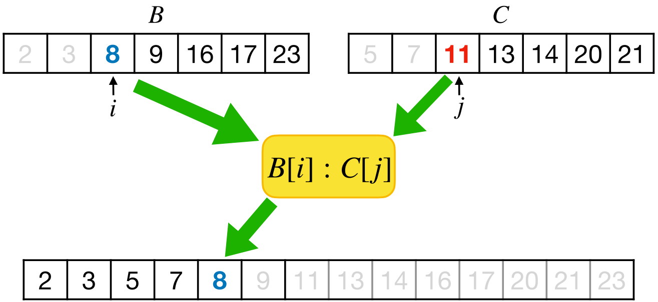
  $\Rightarrow$ increment inversion count by $p - i + 1$.

$B[i] > C[j] \implies C[j]$ is inverted with $B[i], \ldots, B[p-1]$



$C[1]$ is inverted with $B[2], B[3], B[4], B[5], B[6]$

$B[i] \leq C[j] \Longrightarrow B[j]$ not inverted with $C[j], \ldots, C[q-1]$



$B[2]$ is not inverted with $C[2], C[3], C[4], C[5], C[6]$

```
MergeAndCount(B, C)
    p = B.length, q = C.length
    create an empty array D of length p + q
    i = 0, j = 0, count = 0
    while i < p and j < q do
        if B[i] ≤ C[j] then
            /* B[i] is not inverted with C[j],...,C[q-1].   */
            append B[i] to D
            i++
        else
            /* C[j] is inverted with B[i],...,B[p-1].   */
            count = count + (p - i)
            append C[j] to D
            j++
    if i ≥ p then
        for k = j to q - 1 do  append C[k] to D
    else
        for k = i to p - 1 do  append B[k] to D
    return ⟨count, D⟩
```

Theorem

`SortAndCount` *computes the number of inversions in an array of length* n *in* $\Theta(n \log n)$ *time.*

Proof.

- Key Observation: `MergeAndCount` *runs in* $\Theta(n)$ *time.*
- Thus, `SortAndCount` satisfies the same recurrence as `MergeSort`:

$$T(n) = \begin{cases} c & \text{if } n \le 1, \\ 2T(n/2) + cn & \text{otherwise.} \end{cases}$$

- Hence, $T(n) = \Theta(n \log n)$.

□

Bibliography

References

[CLRS] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms* (3rd edition), MIT Press, 2009.

[KT] Jon Kleinberg and Éva Tardos, *Algorithm Design*, Addison-Wesley, 2006.

[R] Tim Roughgarden. *Algorithms Illuminated Part 1: The Basics*, 2017.