Design and Analysis of Algorithms I

Divide and Conquer

Counting Inversions I

The Problem

<u>Input</u>: array A containing the numbers 1, 2, 3, ... in in some arbitrary order.

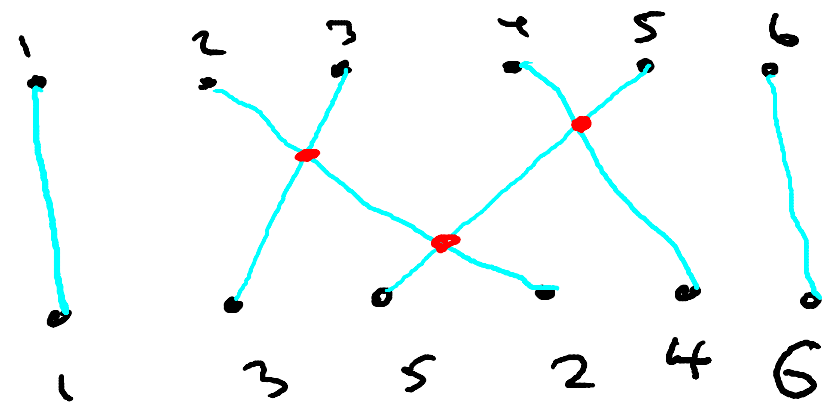
<u>Output</u>: number of <u>inversions</u> = number of pairs (i, j) of array indices with $i < j$ and $A[i] > A[j]$.

Examples and Motivation

Example: (1, 3, 5, 2, 4, 6)

Inversions:

(3, 2), (5, 2), (5, 4)



Motivation: Numerical similarity measure between two ranked lists. (e.g., for "collaborative filtering")

What is the largest-possible number of inversions that a 6-element array can have?

- 15 (in general, $\binom{n}{2} = \frac{n(n-1)}{2}$)
- 21
- 36
- 64

High-Level Approach

Brute-force: $O(n^2)$ time.

Can we do better? YES!

KEY IDEA #1: Divide + Conquer.

Call an inversion (i,j) [with $i<j$]:

left if $i,j \leq n/2$

right if $i,j > n/2$

split if $i \leq \frac{n}{2} < j$

Note: can compute these recursively

need separate subroutine for these

High-Level Algorithm

```
Count(array A, length n)
  if n=1 return 0
  else
     x = Count(1st half of A, n/2)
     y = Count(2nd half of A, n/2)
     z = CountSplitInv(A, n)
  return x+y+z
```

currently unimplemented

<u>Goal</u>: implement CountSplitInv in linear ($O(n)$) time $\Rightarrow$ then Count will run in $O(n \log n)$ time [just like MergeSort].