Divide-and-Conquer

COMP215: Design & Analysis of Algorithms

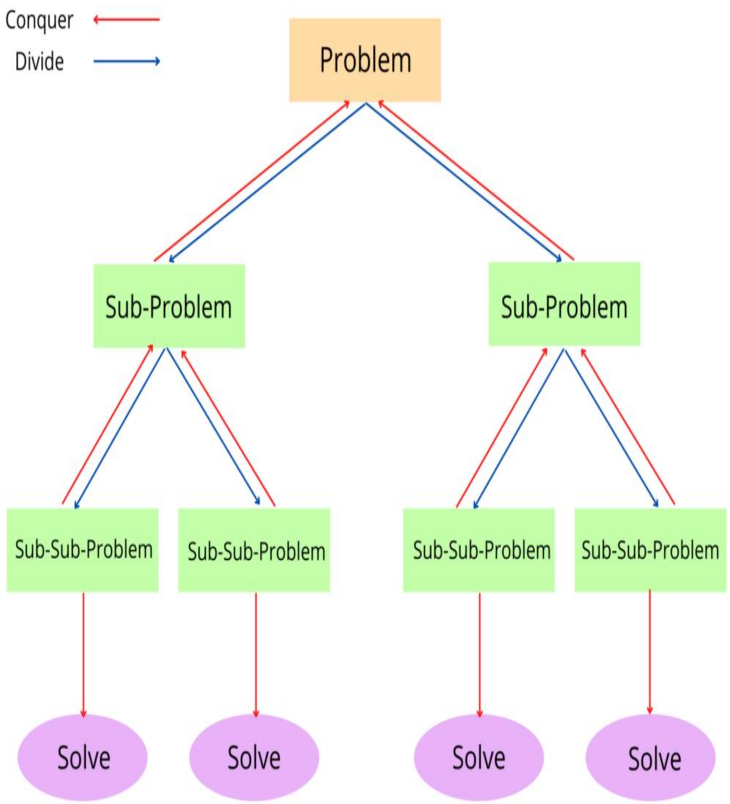
Today

- The Divide-and-Conquer Paradigm
- Counting Inversions

The Divide-and-Conquer Paradigm

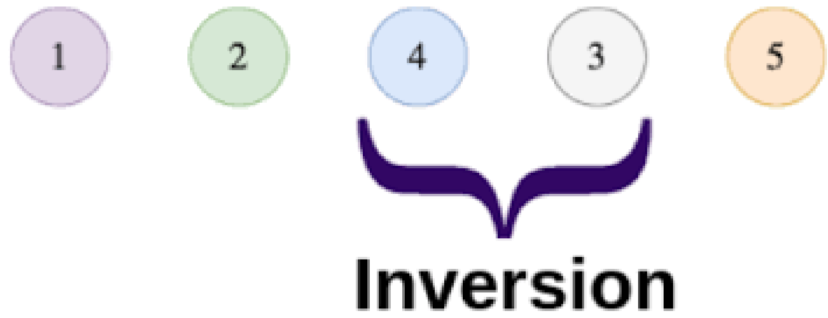
The Divide-and-Conquer Paradigm

1. *Divide* the input into smaller subproblems.
2. *Conquer* the subproblems recursively.
3. *Combine* the solutions for the subproblems into a solution for the original problem.



Counting Inversions

- An inversion of an array is a pair of elements that are "out of order," meaning that the element that occurs earlier in the array is bigger than the one that occurs later.



Problem: Counting Inversions

Input: An array A of distinct integers.

Output: The number of inversions of A—the number of pairs (i, j) of array indices with $i < j$ and $A[i] > A[j]$.

Counting Inversions (Example)

- How many inversions does this array have?

1	3	5	2	4	6

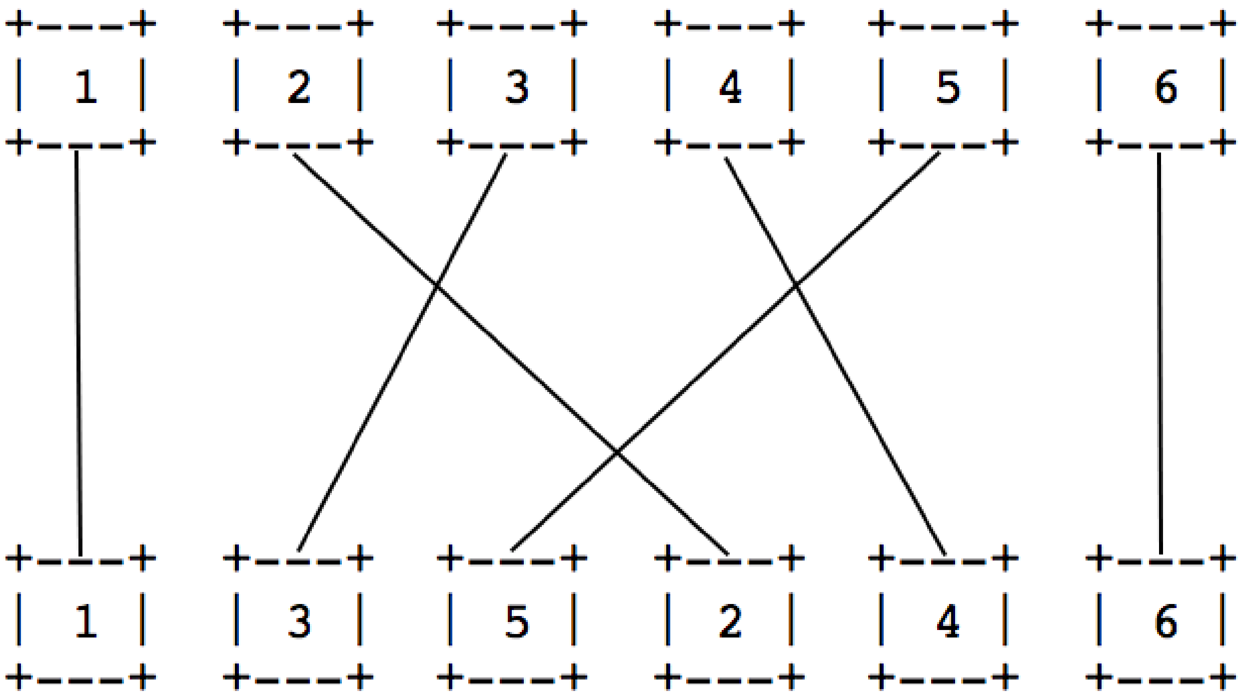
- (3,2), (5,2),(5,4) ➔ 3

- How many inversions does this array have?

5	4	2	1	3	7	6

- (5,4), (5,2), (5,1),(5,3), (4,2),(4,1),(4,3), (2,1),(7,6) ➔ 9

Counting Inversions (Example)



Counting Inversions

Quiz 3.1

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

a) 15

b) 21

c) 36

d) 64

Counting Inversions (Collaborative Filtering)

- One reason to count Inversions is to compute a numerical similarity measure that quantifies how close two ranked lists are to each other

- Example:
- suppose I ask you and a friend to rank, from favorite to least favorite, ten movies that you have both seen. **Are your tastes "similar" or "different?"**

Counting Inversions (Collaborative Filtering)

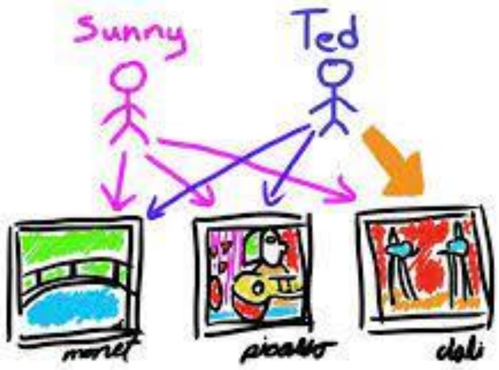
- How can we measure that?

Your favorite movie	1	2	3	4	5	6	7	8	9	10
Your Friend Ranking	5	2	6	1	3	4	9	7	8	10

- Notes:
  - If your rankings are identical:
    - This array will be sorted and have no inversions.
  - The more inversions the array has:
    - The more pairs of movies on which you disagree about their relative merits, and the more different your preferences.

Counting Inversions

- Why do I need the similarity measure between rankings ? **To do collaborative filtering**
- Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. Which can be used then to generate recommendations



Counting Inversions (Algorithm)

- Suggestion for the ?
  - Exhaustive Search:

Input: array A of n distinct integers.
Output: the number of inversions of A.

$numInv := 0$
for $i := 1$ to $n - 1$ **do**
    for $j := i + 1$ to n **do**
        if $A[i] > A[j]$ **then**
            $numInv := numInv + 1$
return $numInv$

$O(n^2)$

Can we do better?

Counting Inversions (Divide-and-conguer)

- The "divide" step will be exactly as in the **MergeSort** algorithm,
  - with one recursive call for the left half of the array
  - one for the right half.
- To understand more, let's classify the inversions (i, j) of an array A of length n into one of three types:
  - **left inversion**: an inversion with i, j both in the first half of the array (i.e., $i, j \leq n/2$);
  - **right inversion**: an inversion with i, j both in the second half of the array (i.e., $i, j > n/2$);
  - **split inversion**: an inversion with i in the left half and j in the right half (i.e., $i \leq n/2 < j$).

Counting Inversions- High-Level Algorithm

```
CountInv
```

Input: array A of n distinct integers.
Output: the number of inversions of A.

---

if $n = 0$ or $n = 1$ **then** `// base cases`
    return 0
else
    $leftInv :=$ `CountInv`(first half of A)
    $rightInv :=$ `CountInv`(second half of A)
    $splitInv :=$ `CountSplitInv`(A)
    return $leftInv + rightInv + splitInv$

Counting Inversions- Using MergeSort

`Sort-and-CountInv`

Input: array A of n distinct integers.
Output: sorted array B with the same integers, and the number of inversions of A.

---

if $n = 0$ or $n = 1$ **then** `// base cases`
    return $(A, 0)$
else
    $(C, leftInv)$:= `Sort-and-CountInv`(first half of A)
    $(D, rightInv)$:=
     `Sort-and-CountInv`(second half of A)
    $(B, splitInv)$:= `Merge-and-CountSplitInv`(C, D)
    return $(B, leftInv + rightInv + splitInv)$

Counting Inversions- Using MergeSort

-

Merge

Input: sorted arrays C and D (length $n/2$ each).
Output: sorted array B (length n).
Simplifying assumption: n is even.

$i := 1$, $j := 1$
for $k := 1$ to n **do**
  if $C[i] < D[j]$ **then**
    $B[k] := C[i]$, $i := i + 1$
  else // $D[j] < C[i]$
    $B[k] := D[j]$, $j := j + 1$

→

Merge-and-CountSplitInv

Input: sorted arrays C and D (length $n/2$ each).
Output: sorted array B (length n) and the number of split inversions.
Simplifying assumption: n is even.

$i := 1$, $j := 1$, $splitInv := 0$
for $k := 1$ to n **do**
  if $C[i] < D[j]$ **then**
    $B[k] := C[i]$, $i := i + 1$
  else // $D[j] < C[i]$
    $B[k] := D[j]$, $j := j + 1$
    $splitInv := splitInv + \underbrace{(\frac{n}{2} - i + 1)}_{\#\text{ left in } C}$
return $(B, splitInv)$