NPTEL MOOC,JAN-FEB 2015
Week 5, Module 6

DESIGN AND ANALYSIS OF ALGORITHMS

Divide and conquer: Counting inversions

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Divide and conquer

- Break up a problem into disjoint subproblems
- Combine these subproblem solutions efficiently
- Examples
  - Merge sort
    - Sort left and right half, then merge
  - Quicksort
    - Rearrange into lower and upper partitions, then sort each partition separately

Recommendation systems

- Online services recommend items to you
- Compare your profile with other customers
  - Identify people who share your likes and dislikes
  - Recommend items that they like
- Comparing profiles: how similar are your rankings to those of others?

Comparing rankings

- You and your friend rank 5 movies, A, B, C, D, E
  - Your ranking: D, B, C, A, E
  - Your friend's ranking: B, A, C, D, E
- How to measure how similar these rankings are?
  - For each pair of movies, compare preferences
    - You rank B above C, so does your friend
    - You rank D above B, your friend ranks B above D

Counting inversions

- Inversion: pair of movies ranked in opposite ordern
  - No inversions: rankings are identical
  - n(n-1)/2 inversions: every pair is inverted
    - maximum dissimilarity of rankings

Counting inversions …

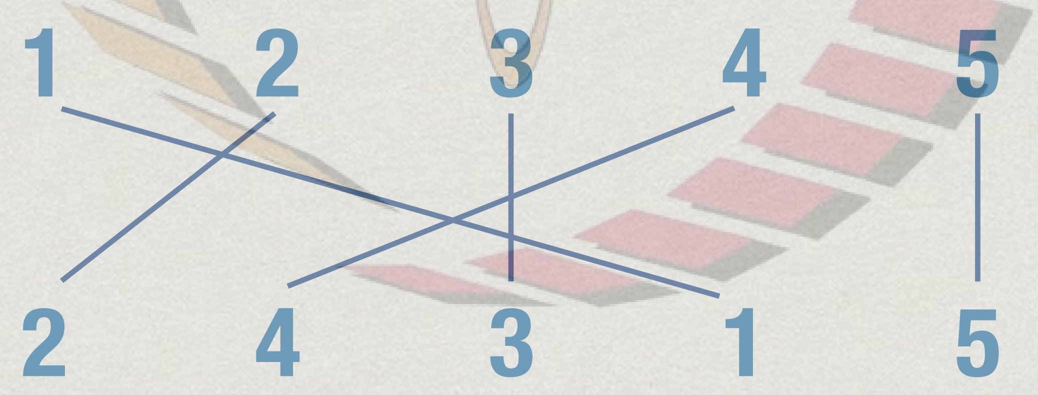
- Equivalent formulation
  - Fix the order of one ranking as a sorted sequence 1, 2, …, n
  - The other ranking is a permutation of 1, 2, …, n
  - An inversion is a pair (i,j), i < j where j appears before i in the permutation

Counting inversions …

- Your ranking: D, B, C, A, E
  - D = 1, B = 2, C = 3, A = 4, E = 5
- Your friend's ranking: B, A, C, D, E
  - Corresponding permutation — 2, 4, 3, 1, 5
- Inversions are (1,2), (1,3), (1,4), (3,4)

Graphically …

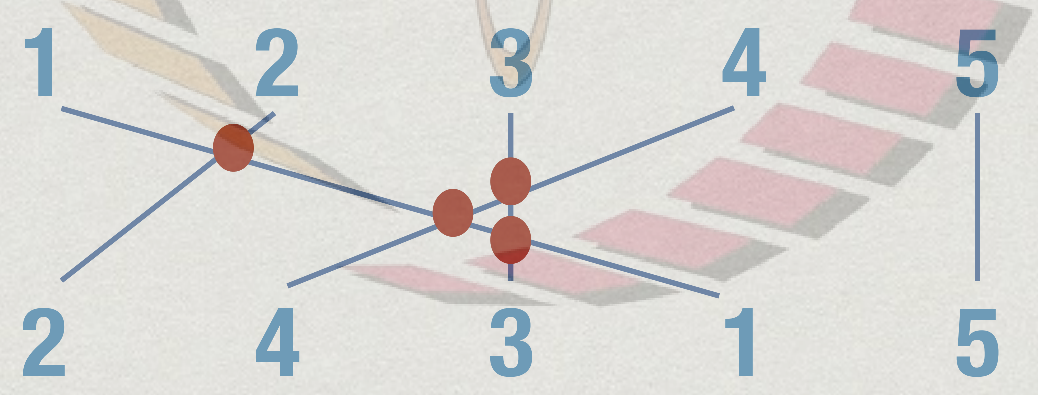
- Your ranking: 1, 2, 3, 4, 5
- Your friend's ranking: 2, 4, 3, 1, 5



- Every crossing is an inversion
- Brute force: check every (i,j) — $O(n^2)$

Graphically …

- Your ranking: 1, 2, 3, 4, 5
- Your friend's ranking: 2, 4, 3, 1, 5



- Every crossing is an inversion
- Brute force: check every (i,j) — $O(n^2)$

Divide and conquer

- Consider your friend's permutation $[i_1, i_2, \ldots, i_N]$
- Divide into two lists
  - $L = [i_1, i_2, \ldots, i_{N/2}]$, $R = [i_{N/2+1}, i_{N/2+2}, \ldots, i_N]$
- Recursively count inversions in L and R
- Add inversions across L and R
  - How many elements in R are bigger than elements in L?

Adapt merge sort

- Divide $[i_1, i_2, \ldots, i_N]$ into two lists
  - $L = [i_1, i_2, \ldots, i_{N/2}]$, $R = [i_{N/2+1}, i_{N/2+2}, \ldots, i_N]$
- Recursively **sort and count** inversions in L and R
- Count inversions across L and R while merging
  - **merge and count**

Merge and count

- $L = [i_1, i_2, \ldots, i_{N/2}]$, $R = [i_{N/2+1}, i_{N/2+2}, \ldots, i_N]$, sorted
- Count inversions across L and R while merging
  - Any element from R added to output is inverted with respect to all elements currently in L
  - Add current size of L to number of inversions

Merge and count

```
function MergeCount(A,m,B,n)
  // Merge A[0..m-1], B[0..n-1] into C[0..m+n-1]

  i = 0; j = 0; k = 0; count = 0;
  // Current positions in A,B,C and inversion count

  while (k < m+n)
    // Case 1: Move head of A into C, no inversions
    if (j==n or A[i] <= B[j])
      C[k] = A[i]; i++; k++;
    // Case 2: Move head of B into C, update count
    if (i==m or A[i] > B[j])
      C[k] = B[j]; j++; k++;
      count = count + (m-i)

  return(count,C)
```

Sort and count

```
function MergeSortCount(A,left,right)
  // Sort the segment A[left..right-1] into B

  if (right - left == 1) // Base case, no inversions
    B[0] = A[left]; count = 0
    return(0,B)

  if (right - left > 1)  // Recursive call

    mid = (left+right)/2

    (countL,L) = MergeSortCount(A,left,mid)
    (countR,R) = MergeSortCount(A,mid,right)

    (countM,B) = MergeCount(L,mid-left,R,right-mid)

    return(countL+countR+countM,B)
```

Analysis

- Similar to Merge Sort
  - $T(1) = 1$
  - $T(n) = 2T(n/2) + n$
- Solve to get $T(n) = O(n \log n)$
- Total number of inversions can be $n(n-1)/2 = O(n^2)$
- We are counting them efficiently without enumerating each one!