Divide and Conquer: Counting Inversions

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 8

Divide and Conquer

- Break up a problem into disjoint subproblems
- Combine these subproblem solutions efficiently

Divide and Conquer

- Break up a problem into disjoint subproblems
- Combine these subproblem solutions efficiently

Examples

Divide and Conquer

- Break up a problem into disjoint subproblems
- Combine these subproblem solutions efficiently

Examples

- Merge sort
  - Split into left and right half and sort each half separately
  - Merge the sorted halves

Divide and Conquer

- Break up a problem into disjoint subproblems
- Combine these subproblems solutions efficiently

Examples

- Merge sort
  - Split into left and right half and sort each half separately
  - Merge the sorted halves
- Quicksort
  - Rearrange into lower and upper partitions, sort each partition separately
  - Place pivot between sorted lower and upper partitions

Divide and Conquer

- Break up a problem into disjoint subproblems
- Combine these subproblem solutions efficiently

Examples

- Merge sort
  - Split into left and right half and sort each half separately
  - Merge the sorted halves
- Quicksort
  - Rearrange into lower and upper partitions, sort each partition separately
  - Place pivot between sorted lower and upper partitions
- Other examples?

Recommender systems

- Online services recommend items to you

Recommender systems

- Online services recommend items to you
- Compare your profile with other customers

Recommender systems

- Online services recommend items to you
- Compare your profile with other customers
- Identify people who share your likes and dislikes

Recommender 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

Recommender 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?

Recommender 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

Recommender 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

Recommender 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?

Recommender 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

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar
- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar
- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Permutations

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar
- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Permutations

- Fix the order of one ranking as a sorted sequence $1, 2, \ldots, n$

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar
- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Permutations

- Fix the order of one ranking as a sorted sequence $1, 2, \ldots, n$
- The other ranking is a permutation of $1, 2, \ldots, n$

Compare based on inversions

Inversions

- Pair of movies ranked in opposite order
  - You rank D above B, your friend ranks B above D
- No inversions — rankings identical
- Every pair inverted — maximally dissimilar
- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Permutations

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

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- Your ranking: D, B, C, A, E

  $\mathrm{D} = 1$, $\mathrm{B} = 2$, $\mathrm{C} = 3$, $\mathrm{A} = 4$, $\mathrm{E} = 5$

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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
  $= 2, 4, 3, 1, 5$

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$

Graphically

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1,2)$, $(1,3)$, $(1,4)$, $(3,4)$

Graphically

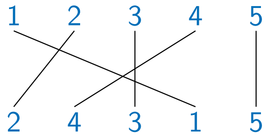
- Write the two permutations as two rows of nodes
- Connect every pair (j, j) between the two rows

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$

Graphically

- Write the two permutations as two rows of nodes
- Connect every pair (j, j) between the two rows



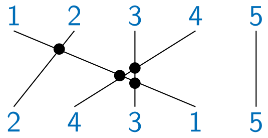
Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$

Graphically

- Write the two permutations as two rows of nodes
- Connect every pair (j, j) between the two rows



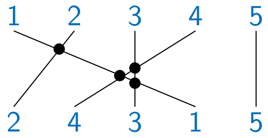
- Every crossing is an inversion

Counting inversions

- Number of inversions ranges from 0 to $n(n-1)/2$ — measure of dissimilarity
- 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 $= 2, 4, 3, 1, 5$
- Inversions in $2, 4, 3, 1, 5$?
- $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$

Graphically

- Write the two permutations as two rows of nodes
- Connect every pair (j, j) between the two rows



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

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- Divide into two lists
  - $L = [i_1, i_2, \ldots, i_{n/2}]$
  - $R = [i_{n/2+1}, i_{n/2+2}, \ldots, i_n]$

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- 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

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- 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 the boundary between L and R
  - $i \in L$, $j \in R$, $i > j$
  - How many elements in L are bigger than elements in R?

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- 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 the boundary between L and R
  - $i \in L$, $j \in R$, $i > j$
  - How many elements in L are bigger than elements in R?
- How to count inversions across the boundary?

Divide and conquer

- Friend's permutation is $i_1, i_2, \dots, i_n$
- Divide into two lists
  - $L = [i_1, i_2, \dots, i_{n/2}]$
  - $R = [i_{n/2+1}, i_{n/2+2}, \dots, i_n]$
- Recursively count inversions in L and R
- Add inversions across the boundary between L and R
  - $i \in L$, $j \in R$, $i > j$
  - How many elements in L are bigger than elements in R?
- How to count inversions across the boundary?
- Adapt merge sort

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- 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 the boundary between L and R
  - $i \in L$, $j \in R$, $i > j$
  - How many elements in L are bigger than elements in R?
- How to count inversions across the boundary?
- Adapt merge sort
- Recursively sort and count inversions in L and R

Divide and conquer

- Friend's permutation is $i_1, i_2, \ldots, i_n$
- 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 the boundary between L and R
  - $i \in L$, $j \in R$, $i > j$
  - How many elements in L are bigger than elements in R?
- How to count inversions across the boundary?
- Adapt merge sort
- Recursively sort and count inversions in L and R
- Count inversions while merging — merge and count

Divide and conquer

Merge and count

- Merge $L = [i_1, i_2, \ldots, i_{n/2}]$ and $R = [i_{n/2+1}, i_{n/2+2}, \ldots, i_n]$, sorted

Divide and conquer

Merge and count

- Merge $L = [i_1, i_2, \ldots, i_{n/2}]$ and $R = [i_{n/2+1}, i_{n/2+2}, \ldots, i_n]$, sorted
- Count inversions while merging
  - If we add i_m from R to the output, i_m is smaller than elements currently in L
  - i_m is hence inverted with respect to elements currently in L
  - Add current size of L to inversion count

Divide and conquer

Merge and count

- Merge $L = [i_1, i_2, \ldots, i_{n/2}]$ and $R = [i_{n/2+1}, i_{n/2+2}, \ldots, i_n]$, sorted
- Count inversions while merging
  - If we add i_m from R to the output, i_m is smaller than elements currently in L
  - i_m is hence inverted with respect to elements currently in L
  - Add current size of L to inversion count

```
def mergeAndCount(A,B):
  (m,n) = (len(A),len(B))
  (C,i,j,k,count) = ([],0,0,0,0)
  while k < m+n:
    if i == m:
      C.append(B[j])
      (j,k) = (j+1,k+1)
    elif j == n:
      C.append(A[i])
      (i,k) = (i+1,k+1)
    elif A[i] < B[j]:
      C.append(A[i])
      (i,k) = (i+1,k+1)
    else:
      C.append(B[j])
      (j,k,count) = (j+1,k+1,count+(m-i))
  return(C,count)
```

Divide and conquer

Merge and count

- Merge $L = [i_1, i_2, \ldots, i_{n/2}]$ and $R = [i_{n/2+1}, i_{n/2+2}, \ldots, i_n]$, sorted
- Count inversions while merging
  - If we add i_m from R to the output, i_m is smaller than elements currently in L
  - i_m is hence inverted with respect to elements currently in L
  - Add current size of L to inversion count
- `sortAndCount` is merge sort with `mergeAndCount`

```
def sortAndCount(A):
  n = len(A)

  if n <= 1:
     return(A,0)

  (L,countL) = sortAndCount(A[:n//2])
  (R,countR) = sortAndCount(A[n//2:])

  (B,countB) = mergeAndCount(L,R)

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

Analysis

- Recurrence is similar to merge sort
  - $T(0) = T(1) = 1$
  - $T(n) = 2T(n/2) + n$

Analysis

- Recurrence is similar to merge sort
  - $T(0) = T(1) = 1$
  - $T(n) = 2T(n/2) + n$
- Solve to get $T(n) = O(n \log n)$

Analysis

- Recurrence is similar to merge sort
  - $T(0) = T(1) = 1$
  - $T(n) = 2T(n/2) + n$
- Solve to get $T(n) = O(n \log n)$
- Note that the number of inversions can still be $O(n^2)$
  - Number ranges from 0 to $n(n-1)/2$

Analysis

- Recurrence is similar to merge sort
  - $T(0) = T(1) = 1$
  - $T(n) = 2T(n/2) + n$
- Solve to get $T(n) = O(n \log n)$
- Note that the number of inversions can still be $O(n^2)$
  - Number ranges from 0 to $n(n-1)/2$
- We are counting them efficiently without enumerating each one