COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Course Overview

- Basic graph algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flows
- Computational Intractability

Divide and Conquer

- You have already seen multiple examples of Divide and Conquer algorithms:
  - Binary Search
  - Merge Sort
  - Quick Sort
  - Multiplying two n-bit numbers in $O\left(n^{\log_2 3}\right)$ time.

- Main Idea: *Divide the input into smaller parts. Solve the smaller parts and combine their solution.*

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Example: Consider the integers sequence $A = [7, 2, 8, 3, 4, 1, 9, 10]$
- What is the total number of inversions?

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Example: Consider the integers sequence $A = [7, 2, 8, 3, 4, 1, 9, 10]$
- What is the total number of inversions? 10

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Naïve algorithm: Check $A[i], A[j]$ for all pairs $i < j$.
- Running time of the naïve algorithm?

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Naïve algorithm: Check $A[i], A[j]$ for all pairs $i < j$.
- Running time of the naïve algorithm? $O(n^2)$

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Divide and conquer strategy:
  - Divide the array into two parts A_L and A_R
  - Count the number of inversions c_L in A_L
  - Count the number of inversions c_R in A_R
  - Count the number of inversions c_{LR} across A_L and A_R
  - Output $c_L + c_R + c_{LR}$

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Divide and conquer strategy:
  - Divide the array into two parts A_L and A_R
  - Count the number of inversions c_L in A_L
  - Count the number of inversions c_R in A_R
  - Count the number of inversions c_{LR} across A_L and A_R
  - Output $c_L + c_R + c_{LR}$
- How much time does it take to find the number of inversions across A_L and A_R?
  - If we can do this in $O(n)$ time, then the recurrence relation for the running time will be $T(n) \leq 2 \cdot T(n/2) + cn$.
  - The solution for the above is $T(n) = O(n \log n)$

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Divide and conquer strategy:
  - Divide the array into two parts A_L and A_R
  - Count the number of inversions c_L in A_L
  - Count the number of inversions c_R in A_R
  - Count the number of inversions c_{LR} across A_L and A_R
  - Output $c_L + c_R + c_{LR}$
- How much time does it take to find the number of inversions across A_L and A_R?
  - Suppose we have sorted A_L and A_R, how much time does it take to count the inversions across A_L and A_R?

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

- Divide and conquer strategy:
  - Divide the array into two parts A_L and A_R
  - Count the number of inversions c_L in A_L
  - Count the number of inversions c_R in A_R
  - Count the number of inversions c_{LR} across A_L and A_R
  - Output $c_L + c_R + c_{LR}$
- How much time does it take to find the number of inversions across A_L and A_R?
  - Suppose we have sorted A_L and A_R, how much time does it take to count the inversions across A_L and A_R? $O(n)$

Problem

Counting inversions: Given a sequence of distinct integers, $A[1], A[2], ..., A[n]$ output the number of pairs (i, j) such that $i < j$ and $A[i] > A[j]$. Such pairs are called *inversions*.

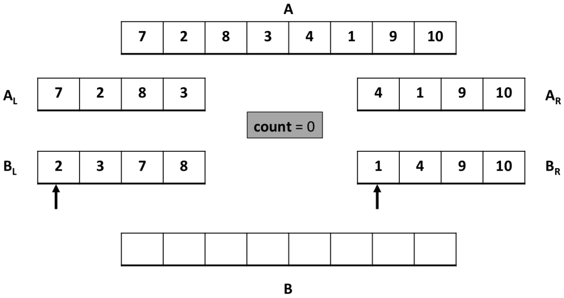
Algorithm

`SortCountInversions(`A`)`

- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2 + 1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions(`A_L`)`
- $(c_R, B_R) \leftarrow$ `SortCountInversions(`A_R`)`
- $(c_{LR}, B) \leftarrow$ `MergeCount(`B_L, B_R`)`
- return$((c_L + c_R + c_{LR}), B)$

Algorithm

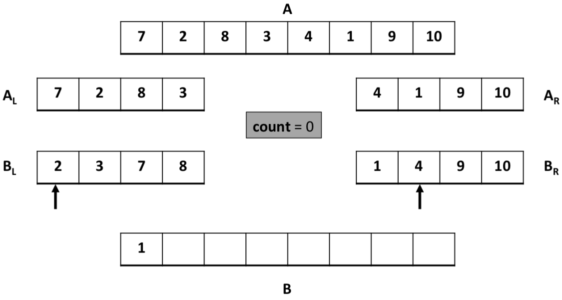
`SortCountInversions`(A)
- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions`(A_L)
- $(c_R, B_R) \leftarrow$ `SortCountInversions`(A_R)
- $(c_{LR}, B) \leftarrow$ `MergeCount`(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Algorithm

`SortCountInversions`(A)

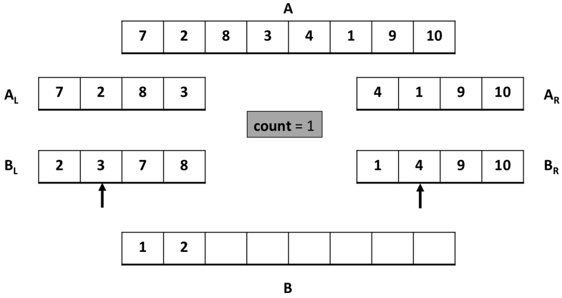
- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions`(A_L)
- $(c_R, B_R) \leftarrow$ `SortCountInversions`(A_R)
- $(c_{LR}, B) \leftarrow$ `MergeCount`(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Algorithm

`SortCountInversions`(A)

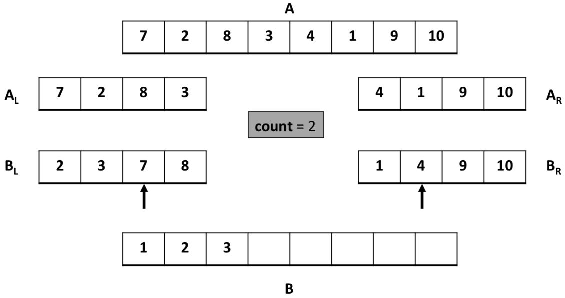
- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions`(A_L)
- $(c_R, B_R) \leftarrow$ `SortCountInversions`(A_R)
- $(c_{LR}, B) \leftarrow$ `MergeCount`(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Algorithm

`SortCountInversions(`A`)`

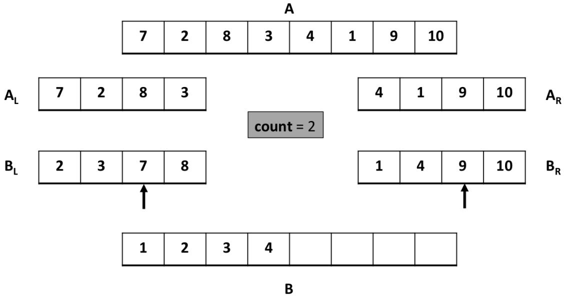
- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions(`A_L`)`
- $(c_R, B_R) \leftarrow$ `SortCountInversions(`A_R`)`
- $(c_{LR}, B) \leftarrow$ `MergeCount(`B_L, B_R`)`
- return$((c_L + c_R + c_{LR}), B)$



Algorithm

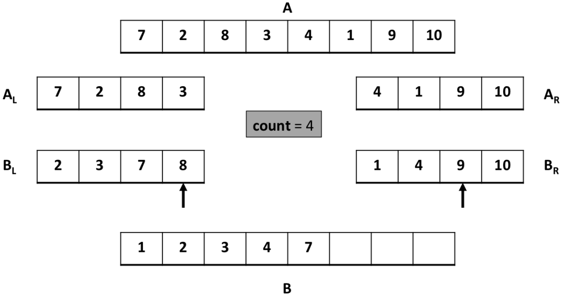
`SortCountInversions(`A`)`

- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions(`A_L`)`
- $(c_R, B_R) \leftarrow$ `SortCountInversions(`A_R`)`
- $(c_{LR}, B) \leftarrow$ `MergeCount(`B_L, B_R`)`
- return$((c_L + c_R + c_{LR}), B)$



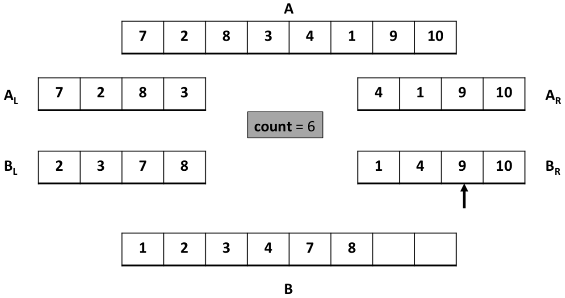
Algorithm

`SortCountInversions`(A)
- if $(|A| = 1)$return$(0, A)$
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions`(A_L)
- $(c_R, B_R) \leftarrow$ `SortCountInversions`(A_R)
- $(c_{LR}, B) \leftarrow$ `MergeCount`(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Algorithm

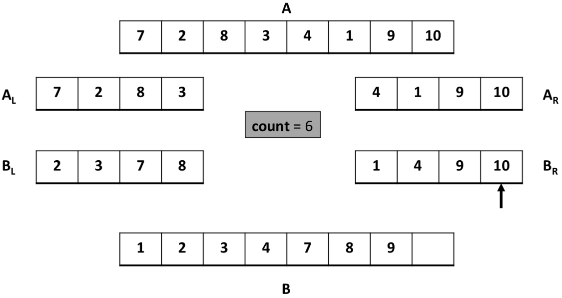
SortCountInversions(A)
- if $(|A| = 1)$return(0)
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ SortCountInversions(A_L)
- $(c_R, B_R) \leftarrow$ SortCountInversions(A_R)
- $(c_{LR}, B) \leftarrow$ MergeCount(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Divide and Conquer
Counting Inversions

Algorithm

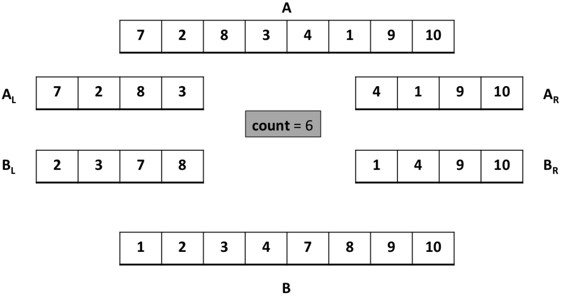
SortCountInversions(A)
- if ($|A| = 1$)return(0)
- Let $A_L \leftarrow A[1]...A[n/2]$
- Let $A_R \leftarrow A[n/2+1]...A[n]$
- $(c_L, B_L) \leftarrow$ SortCountInversions(A_L)
- $(c_R, B_R) \leftarrow$ SortCountInversions(A_R)
- $(c_{LR}, B) \leftarrow$ MergeCount(B_L, B_R)
- return($(c_L + c_R + c_{LR}), B$)



Divide and Conquer
Counting Inversions

Algorithm

`SortCountInversions`(A)
- if $(|A| = 1)$return(0)
- Let $A_L \leftarrow A[1] \ldots A[n/2]$
- Let $A_R \leftarrow A[n/2+1] \ldots A[n]$
- $(c_L, B_L) \leftarrow$ `SortCountInversions`(A_L)
- $(c_R, B_R) \leftarrow$ `SortCountInversions`(A_R)
- $(c_{LR}, B) \leftarrow$ `MergeCount`(B_L, B_R)
- return$((c_L + c_R + c_{LR}), B)$



Course Overview

- Graph Algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flows
- Computational Intractability

Dynamic Programming

- Main idea: *Break the given problem in to a few sub-problems and combine the solutions of the smaller sub-problems to get solutions to larger ones.*
- How is it different than Divide and Conquer?
  - Here you are allowed overlapping sub-problems.

- Main idea: *Break the given problem in to a few sub-problems and combine the solutions of the smaller sub-problems to get solutions to larger ones.*
- How is it different than Divide and Conquer?
  - Here you are allowed overlapping sub-problems.
- Suppose your recursive algorithm gives a recursion tree that has many common sub-problems (e.g., recursion for computing Fibonacci numbers), then it helps to save the solution of sub-problems and use this solution whenever the same sub-problem is called.
- Dynamic programming algorithms are also called *table-filling* algorithms

Problem

Longest increasing subsequence: You are given a sequence of integers $A[1], A[2], ..., A[n]$ and you are asked to find a longest increasing subsequence of integers.

- Example: The longest increasing subsequence of the sequence $(7, 2, 8, 6, 3, 6, 9, 7)$ is ?

End