Design and Analysis of Algorithms I

Divide and Conquer

Counting Inversions II

Piggybacking on Merge Sort

KEY IDEA # 2 : have recursive calls both count inversions and sort.
[i.e. , piggy back on Merge Sort]

Motivation : Merge subroutine naturally uncovers split inversions [as we'll see]

High-Level Algorithm (revised)

Sort-and-Count (array A, length n)

if n=1, return 0

else

Sorted version of 1^{st} half

(B,X) = Sort-and-Count(1^{st} half of A, n/2)

Sorted version of 2^{nd} half

(C,Y) = Sort-and-Count(2^{nd} half of A, n/2)

Sorted version of A

(D,Z) = CountSplitInv(A,n)

CURRENTLY UNIMPLEMENTED

return X+Y+Z

<u>Goal</u> : implement CountSplitInv in linear (O(n)) time

=> then Count will run in O(nlog(n)) time [just like Merge Sort]

Pseudocode for Merge:

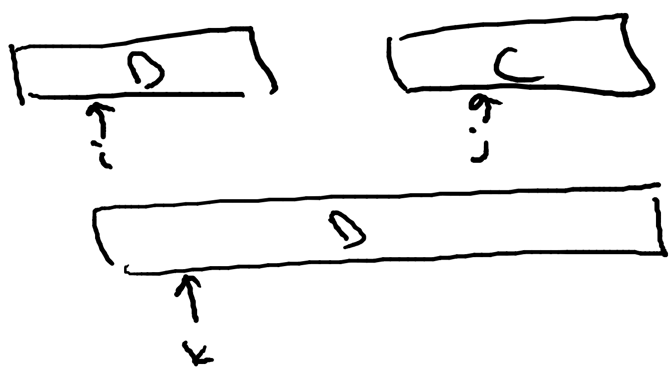
D = output [length = n]

B = 1^{st} sorted array [n/2]

C = 2^{nd} sorted array [n/2]

i = 1

j = 1




```
for k = 1 to n
        if B(i) < C(j)
                D(k) = B(i)
                i++
        else [C(j) < B(i)]
                D(k) = C(j)
                j++
end
```

(ignores end cases)

Tim Roughgarden

Ter
ver

Suppose the input array A has no split inversions. What is the relationship between the sorted subarrays B and C?

- B has the smallest element of A, C the second-smallest, B, the third-smallest, and so on.
- All elements of B are less than all elements of C.
- All elements of B are greater than all elements of C.
- There is not enough information to answer this question.

Example

Consider merging B 1 3 5 and 2 4 6 . C

Output : 1 2 3 4 5 6 D

⟹When 2 copied to output, discover the split inversions (3,2) and (5,2)

⟹ when 4 copied to output, discover (5,4)

General Claim

<u>Claim</u> the split inversions involving an element y of the 2nnd array C are precisely the numbers left in the 1st array B when y is copied to the output D.

<u>Proof</u> : Let x be an element of the 1st array B.

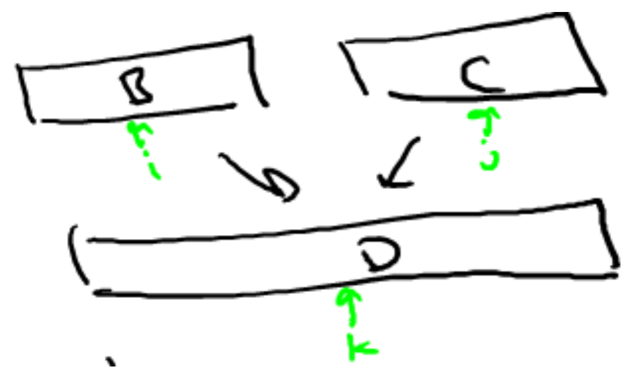
1. if x copied to output D before y, then $x < y$

$\Rightarrow$ no inversions involving x and y

2. If y copied to output D before x, then $y < x$

=> X and y are a (split) inversion. Q.E.D

Merge_and_CountSplitInv



-- while merging the two sorted subarrays, keep running total of number of split inversions

-- when element of 2^{nd} array C gets copied to output D, increment total by number of elements remaining in 1^{st} array B

<u>Run time of subroutine</u> : $\overbrace{O(n)}^{\text{merge}} + \overbrace{O(n)}^{\text{running total}} = O(n)$

=> Sort_and_Count runs in O(nlog(n)) time [just like Merge Sort]