16 February 2024. Divide & Conquer

Plan

* Review of Recurrence Relations
  ↳ Merge Sort

* Announcements

* Counting Inversions

Divide & Conquer

* <u>Divide</u>. Split the problem instance into smaller sub-instances

* <u>Recurse</u>. Solve each sub-instance recursively

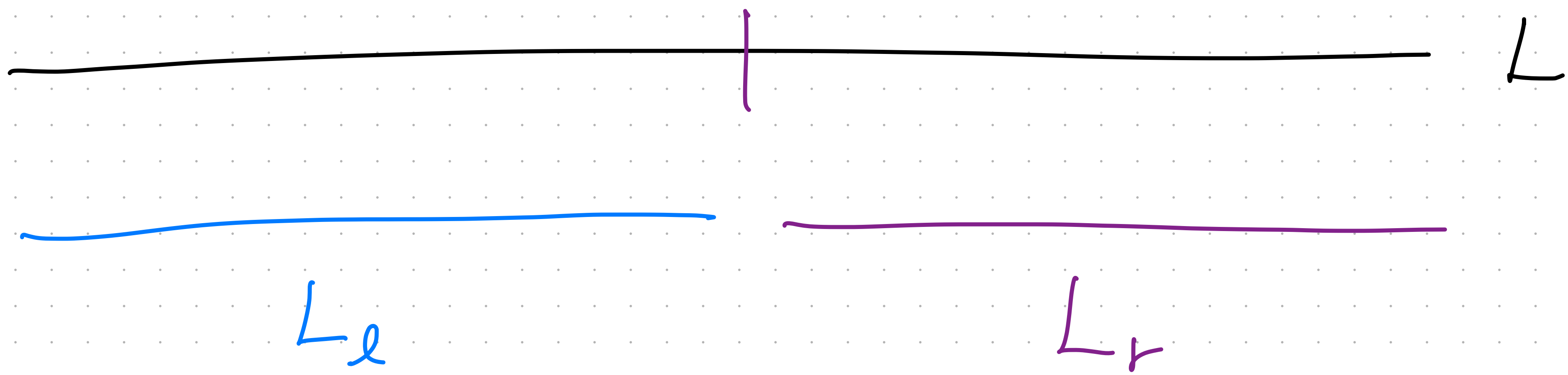
* <u>Combine</u>. Given sub-solutions, combine into global solution for original instance.

Canonical Example: Merge Sort.

Given: List of unsorted integers

Return: List in sorted order.

$l_1 \le l_2 \le \dots \le l_n$



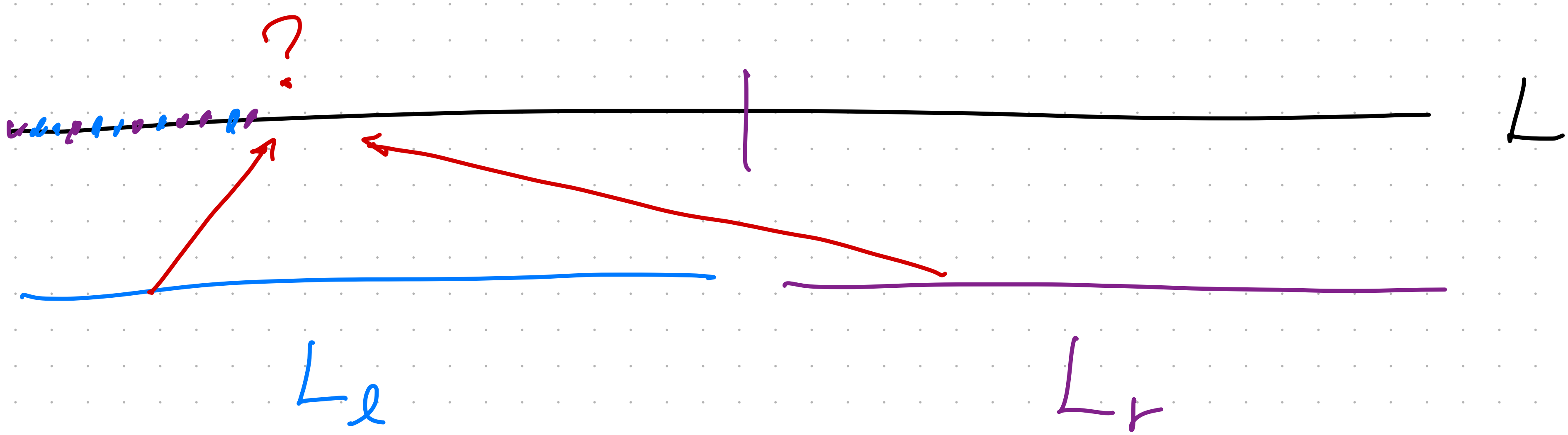
↳ Recurse on left half of list

↳ Recurse on right half of list

Canonical Example. Merge Sort.

Given. List of unsorted integers

Return. List in sorted order.



↳ Recurse on left half of list

↳ Recurse on right half of list

MergeSort (L)

if $|L| = 1$. Return L.

$L_\ell \leftarrow$ left half of L

$L_r \leftarrow$ right half of L

$sL_\ell \leftarrow$ MergeSort (L_ℓ)

$sL_r \leftarrow$ MergeSort (L_r)

Return Combine (sL_ℓ, sL_r)

Recurrence Relation for RT Analysis.

Define $T(n)$ ← Running time on instances of size n.

Goal. Express $T(n)$ recursively

in terms of $T(k)$ for $k < n$.

Merge Sort (L) $\quad T(n)$

if $|L| = 1$. Return L. $\quad O(1)$

$L_l \leftarrow$ left half of L

$L_r \leftarrow$ right half of L

$sL_l \leftarrow$ MergeSort(L_l) $\quad T(n/2)$

$sL_r \leftarrow$ MergeSort(L_r) $\quad T(n/2)$

Return Combine(sL_l, sL_r) $\quad O(n)$

$l \overset{?}{\leq} r$

Merge Sort Recurrence.

$$T(n) \leq 2 \cdot T(n/2) + c \cdot n$$

$$T(1) = \Theta(1).$$

Merge Sort Recurrence.

$$2\cdot\left(2\cdot T(n/4) + c\cdot\left(\frac{n}{2}\right)\right) + cn$$

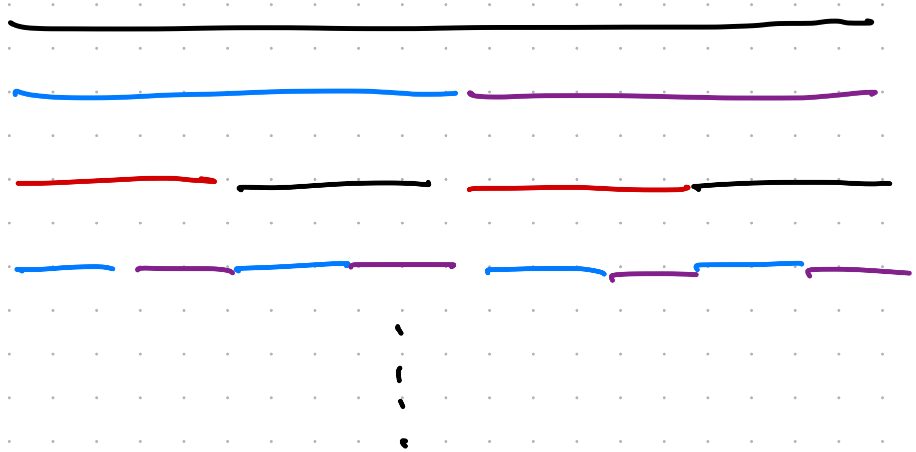
$$T(n) \le \overbrace{2\cdot T(n/2)} + c\cdot n$$

$$T(1) = \Theta(1).$$

RT

$O(n\log n)$

Analyzing the Recurrence.



Total work per level

$c\cdot n$

# of levels?

$\log n$

$\log_2 n = O(\log_2 n)$ $\quad \log_2 n$

Another Recurrence

$$T(n) \le 2 \cdot T\left(\underbrace{\frac{n}{2}}\right) + n^2$$

$$T(n) \le n^2 \log n$$

$$T(n) \le O(n^2)$$

$$T(n) \le 2 \cdot T\left(\frac{n}{2}\right) + n^2$$

$$\le 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$\underbrace{2 \cdot \left(\frac{n}{2}\right)^2}_{\frac{n^2}{2}} + n^2$$

$$n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \ldots \le 2n^2$$

Slow Recurrences.

$$T(n) \geq 2 \cdot T(n-1)$$

$$\geq 2 \cdot (2 \cdot T(n-2))$$

$$\geq 2 \, 2 \cdots \quad T(1)$$

$$= 2^n$$

$$T(n) \geq \underbrace{T(n-1) + T(n-2)}_{\text{Fibonacci}}$$

$$\geq 2 \cdot T(n-2)$$

$$\geq 2^{n/2}$$

Announcements

* Prelim Tues 2/20, 7:30pm. — 9:00 pm
  - ↳ See Ed for room assignments.

* Prelim Review Sessions
  - ↳ Saturday 1–3:30pm
  - ↳ Sunday 2–4:30 pm

  Gates G01

  Written materials & example videos will be posted on Canvas.

* No homework this week :)

Counting Inversions.

Rankings are important in CS / social applications

How "similar" are two rankings?

Counting Inversions.

Rankings are important in CS / social applications

How "similar" are two rankings?

Example. Measuring trends in content creator popularity.

Given a list of February's top creators, how similar is it to January's top creators.

Given. A permutation S of $\{1, \dots, n\}$

Find. The number of "inversions" in S

Given. A permutation S of $\{1, \dots, n\}$

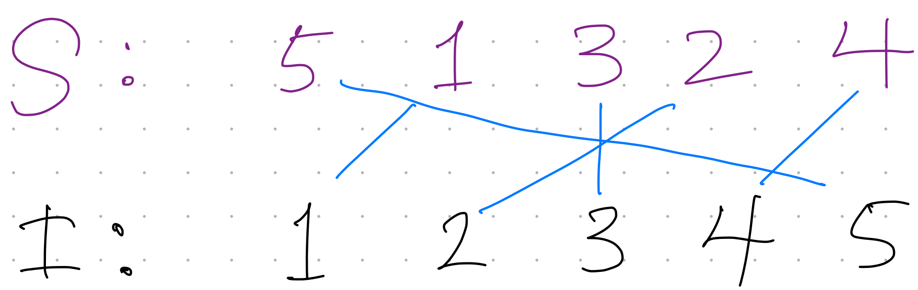
Find. The number of "inversions" in S

$$\underbrace{1\ \ 2\ \ 3}_{0 \text{ inversions}} \qquad \underbrace{1\ 3\ 2}_{1} \qquad \underbrace{2\ 1\ 3}_{1} \qquad \underbrace{2\ 3\ 1}_{2} \qquad \underbrace{3\ 1\ 2}_{2} \qquad \underbrace{3\ 2\ 1}_{3}$$

Given. A permutation S of $\{1, \dots, n\}$

Find. The number of "inversions" in S

$$\underbrace{1\ \ 2\ \ 3}_{0 \text{ inversions}} \qquad \underbrace{1\ 3\ 2}_{1} \qquad \underbrace{2\ 1\ 3}_{1} \qquad \underbrace{2\ 3\ 1}_{2} \qquad \underbrace{3\ 1\ 2}_{2} \qquad \underbrace{3\ 2\ 1}_{3}$$

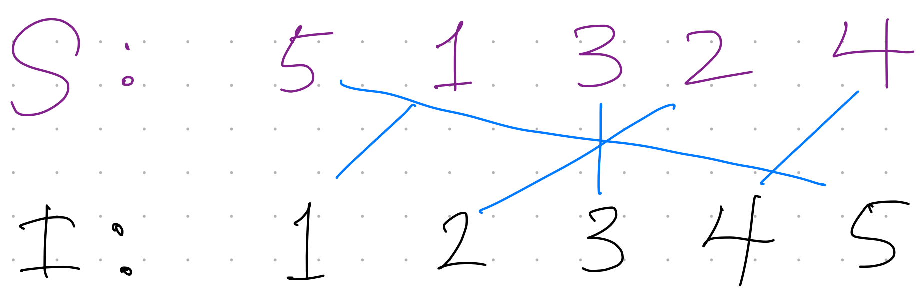


How many crossings?

Given. A permutation S of $\{1, \dots, n\}$

Find. The number of "inversions" in S

1 2 3	1 3 2	2 1 3	2 3 1	3 1 2	3 2 1
0 inversions	1	1	2	2	3



How many crossings?

$$\#\text{inversions}(S) = \left| \left\{ \begin{matrix} i \in \{1 \dots n\} \\ j \in \{1 \dots n\} \end{matrix} \text{ s.t. } \begin{matrix} i < j \\ S_i > S_j \end{matrix} \right\} \right|$$

Naive Algorithm?

Check All Pairs (S)

count = 0

For $i = 1 \cdots n-1$

  For $j = i+1 \cdots n$

    if $S_i > S_j$.

      count $\leftarrow$ count + 1.

Return count.

Check All Pairs (S)

count = 0

For $i = 1 \cdots n-1$

  For $j = i+1 \cdots n$

    if $S_i > S_j$.

      count $\leftarrow$ count + 1.

Return count.

Correctness?

Checks every pair ✓

Time Complexity?

$\Theta(n^2)$

Check All Pairs (S)

```
count = 0
For i = 1 ... n-1
    For j = i+1 ... n
        if S_i > S_j .
            count ← count + 1 .
Return count.
```

Correctness ?

Checks every pair ✓

Time Complexity ?

$\Theta(n^2)$

Can we do better?

Observation. Sorting "inverts inversions"

For every permutation S

$$\mathrm{Sort}(S) \longrightarrow I = \{1, 2, \ldots, n\}$$

Observation. Sorting "inverts inversions"

For every permutation S

$$\text{Sort}(S) \longrightarrow I = \{1, 2, \ldots, n\}$$

Idea.

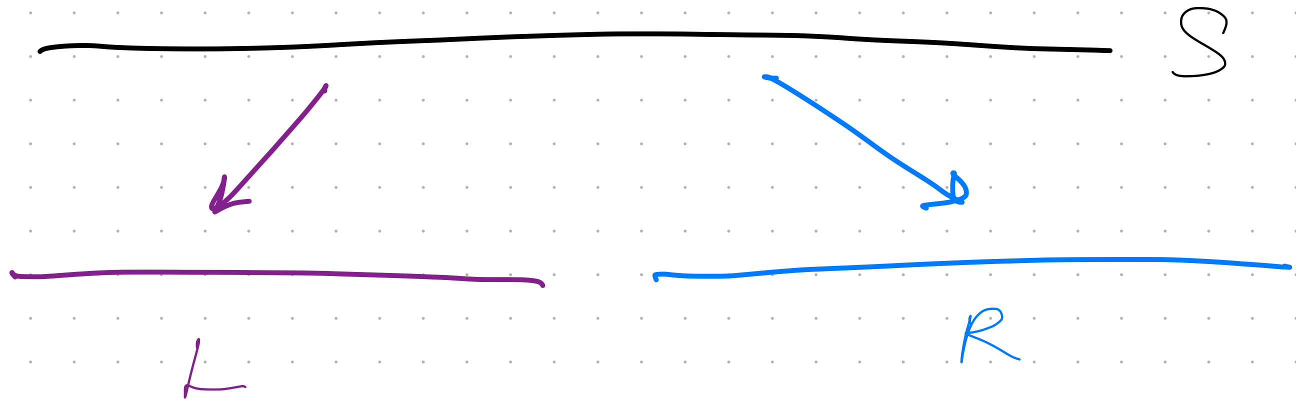
- Run sorting algorithm
- Keep track of inversions along the way.

Given S

Divide

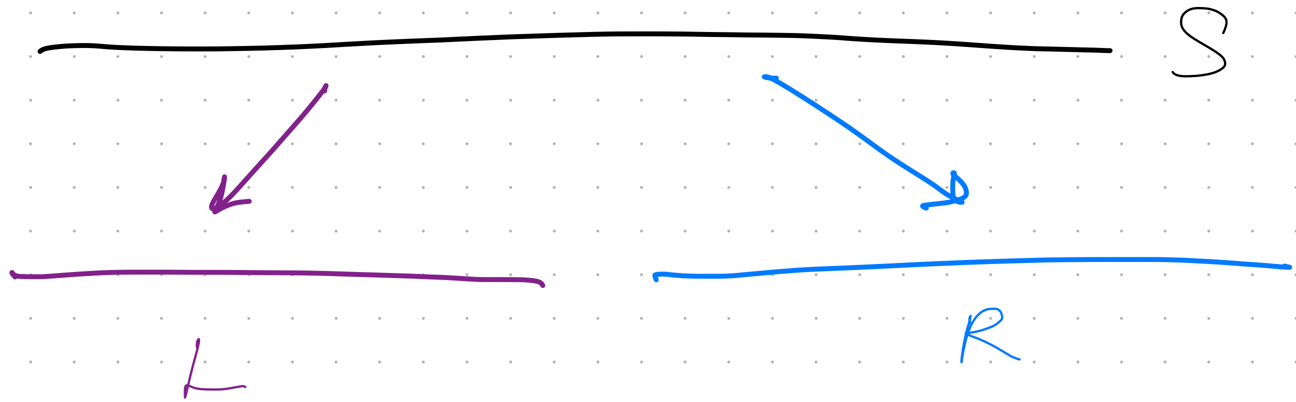
Recurse

Combine



$n_L = \#\text{inversions}(L)$ $\quad$ $n_R = \#\text{inversions}(R)$

Given S



Divide

Recurse

$n_L = \#\text{inversions}(L)$ $n_R = \#\text{inversions}(R)$

Combine

$$n_S = n_L + n_R + \boxed{???}$$

Given S — S

Divide — L, R

Recurse — $n_L = \#\text{inversions}(L)$, $n_R = \#\text{inversions}(R)$

Combine — $n_S = n_L + n_R + \boxed{???}$

$\#$ inversions across $L \leftrightarrow R$

$$\left|\left\{ \begin{matrix} i \in |L| \\ j \in |R| \end{matrix} \text{ s.t. } R_j < L_i \right\}\right|$$

Thought Experiment

What if L and R were sorted?

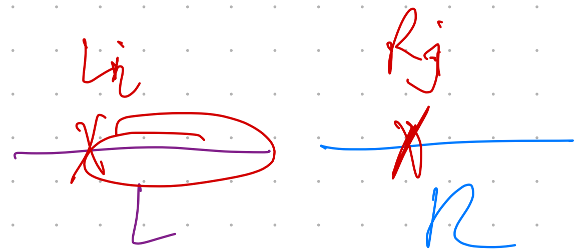
Suppose $R_j < L_i$.

What can we conclude?

Thought Experiment

What if L and R were sorted?

Suppose $R_j < L_i$.



What can we conclude? $R_j < L_i$

Claim. If $R_j < L_i$, then

$$R_j < L_k \quad \forall k \geq i$$

[Pf. By sorted order, $L_i < L_k$ for all $k > i$.]

Count Across Sorted Lists (L, R)

Count = 0.

$i = 1$, $j = 1$.

For $k = 1 \longrightarrow |L| + |R|$.

- if $R_j < L_i$:
  - count $\longleftarrow$ count $+ 1 + \underbrace{(|L| - i)}_{= \#\text{ elems in } L \geq L_i}$
  - $j{+}{+}$
- else :
  - $i{+}{+}$

Return count

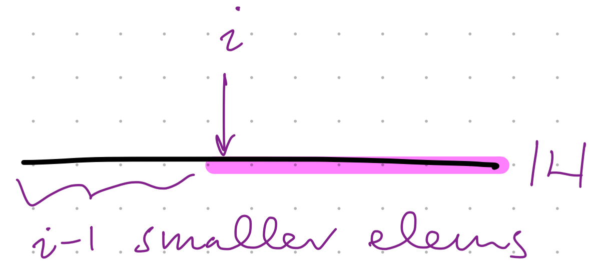
// Note: need to handle case where i/j go off end of L/R

Count Across Sorted Lists (L, R)

$\text{Count} = 0.$

$i = 1, \quad j = 1.$

For $k = 1 \longrightarrow |L| + |R|$.



if $R_j < L_i$:

$$\text{count} \leftarrow \text{count} + 1 + \underbrace{(|L| - i)}_{\substack{= \#\text{ elems in } L \\ \geq L_i}}$$

$j{+}{+}$

else :

$i{+}{+}$

Return count

// Note: need to handle case where i/j go off end of L/R

Count Across Sorted Lists (L, R)

Running Time?

Count = 0.

$i = 1$, $j = 1$.

For $k = 1 \longrightarrow |L| + |R|$.

if $R_j < L_i$:

count $\leftarrow$ count $+ 1 + \underbrace{(|L| - i)}_{\substack{= \#\text{ elems in } L \\ \geq L_i}}$

j++

else :

i++

Return count

// Note: need to handle case where i/j go off end of L/R

Claim. If L and R are sorted, then CountAcrossSortedLists returns count of # inversions across L and R

Claim. If L and R are sorted, then CountAcrossSortedLists returns count of # inversions across L and R

Pf. Every inversion involves some

$$R_j < L_i .$$

Claim. If L and R are sorted, then CountAcrossSortedLists returns count of # inversions across L and R

Pf. Every inversion involves some

$$R_j < L_i .$$

Note, if alg finds such a pair, processes R_j and advances j++. So we only process R_j once.

<u>Claim</u>. If L and R are sorted, then CountAcrossSortedLists returns count of # inversions across L and R

<u>Pf</u>. Every inversion involves some

$$R_j < L_i .$$

Note, if alg finds such a pair, processes R_j and advances j++. So we only process R_j once.

By processing order,

$\implies$ i is the least index s.t. $R_j < L_i$.

<u>Claim</u>. If L and R are sorted, then CountAcrossSortedLists returns count of # inversions across L and R

<u>Pf</u>. Every inversion involves some

$$R_j < L_i .$$

Note, if alg finds such a pair, processes R_j and advances j++. So we only process R_j once.

By processing order,

$\Longrightarrow$ i is the least index s.t. $R_j < L_i$.

So we consider every such R_j once, and add all of its inversions (by prev. slide) to count. ∎

CountInversionsMerge (L, R)

$S = [\ ]$ // array of $|L| + |R|$

Count = 0.

$i = 1$, $j = 1$.

For $k = 1 \longrightarrow |L| + |R|$.

if $R_j < L_i$:

count $\longleftarrow$ count $+ 1 + \underbrace{(|L| - i)}_{= \#\text{ elems in } L \geq L_i}$

$S_k \longleftarrow R_j$

$j{+}{+}$

else :

$S_k \longleftarrow L_i$

$i{+}{+}$

Return (S, count)

// Note: need to handle case where i/j go off end of L/R

Claim If L and R are sorted, then CountInversionsMerge returns S in sorted order.

Pf. By induction. Follows by correctness of MergeSort.

CountInversionsSort(S).

if $|S| \leq 1$, return $\emptyset$.

$(L_{sorted}, n_L) \leftarrow \text{CountInversionsSort}(L)$

$(R_{sorted}, n_R) \leftarrow \text{CountInversionsSort}(R)$

$(S_{sorted}, n_X) \leftarrow \text{CountInversions}\underline{\text{Merge}}(L_{sorted}, R_{sorted})$

Return $(S_{sorted}, n_L + n_R + n_X)$

Correctness. By induction.

Claim CountInversionsSort(S) returns

S in sorted order, and

count = # inversions in S

Correctness. By induction.

Claim CountInversionsSort(S) returns

S in sorted order, and

count = # inversions in S

Base Case. $|S| \leq 1$, Already sorted.

0 inversions ✓

Correctness. By induction.

Claim CountInversionsSort (S) returns

S in sorted order, and

count = # inversions in S

Base Case. $|S| \leq 1$, Already sorted.

0 inversions ✓

Induction.

Follows by correctness of CountInversionsMerge

Correctness. By induction.

Claim CountInversionsSort(S) returns S in sorted order, and count = # inversions in S

Base Case. $|S| \le 1$, Already sorted. 0 inversions ✓

Induction.

Follows by correctness of CountInversionsMerge

$$\text{Total \# inversions} = \underbrace{\text{\# on left} + \text{\# on right}}_{\text{By IH}} + \underbrace{\text{\# across L \& R}}_{\text{By correctness of CountInversionsMerge.}}$$

Running Time Analysis.

CountInversionsSort(S). $\qquad T(n)$

if $|S| \leq 1$, return $\emptyset$.

$(L_{sorted}, n_L) \leftarrow \text{CountInversionsSort}(L)$ $\qquad T(n/2)$

$(R_{sorted}, n_R) \leftarrow \text{CountInversionsSort}(R)$ $\qquad T(n/2)$

$(S_{sorted}, n_X) \leftarrow \text{CountInversions}\underline{\text{Merge}}(L_{sorted}, R_{sorted})$?

Return $(S_{sorted}, n_L + n_R + n_X)$

Claim. Count Inversions Sort runs in $O(n \log n)$ time.

As n grows, much faster than $\Theta(n^2)$.