16 February 2024. Divide & Conquer

Plan

* Review of Recurrence Relations
  - Merge Sort
* Announcements
* Counting Inversions
Divide & Conquer

* **Divide.** Split the problem instance into smaller sub-instances

* **Recurse.** Solve each sub-instance recursively

* **Combine.** Given sub-solutions, combine into global solution for original instance.
Canonical Example: Merge Sort.

Given: List of unsorted integers

Return: List in sorted order.

\[ l_1 \leq l_2 \leq \cdots \leq l_n \]

\[ L \]

\[ \text{L} \rightarrow \text{Recursively sort left half of list} \]

\[ \text{L} \rightarrow \text{Recursively sort right half of list} \]
Canonical Example: Merge Sort.

Given: List of unsorted integers

Return: List in sorted order

\[ \text{Le} \rightarrow \text{Lr} \text{ L} \]

\[ \rightarrow \text{ Recurse on left half of list } \]

\[ \rightarrow \text{ Recurse on right half of list } \]
**Merge Sort (L).**

if \(|L| = 1\). Return \(L\).

\(L_e \leftarrow \text{left half of } L\)

\(L_r \leftarrow \text{right half of } L\)

\(sL_e \leftarrow \text{Merge Sort } (L_e)\)

\(sL_r \leftarrow \text{Merge Sort } (L_r)\)

Return Combine \((sL_e, sL_r)\)
Recurrence Relation for RT Analysis

Define \( T(n) \) \( \rightarrow \) Running time on instances of size \( n \). 

Goal. Express \( T(n) \) recursively in terms of \( T(k) \) for \( k < n \).
**Merge Sort (L)**

- if \( \|L\| = 1 \) return \( L \)
- \( L_e \leftarrow \) left half of \( L \)
- \( L_r \leftarrow \) right half of \( L \)

\( T(n) \)

- \( sL_e \leftarrow \text{Merge Sort} (L_e) \)
- \( sL_r \leftarrow \text{Merge Sort} (L_r) \)

\( T(n/2) \)

\( T(n/2) \)

- Return Combine \((sL_e, sL_r)\)

\( O(n) \)

\( \leq r \)

\( l \leq r \)
Merge Sort \quad \text{Recurrence.}

\[ T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n \]

\[ T(1) = \Theta(1). \]
Merge Sort Recurrence:

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

\[ T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + cn \]

\[ T(1) = O(1) \]

Analyzing the Recurrence:

Total work per level:

\[ C \cdot n \]

\# of levels:

\[ \log_2 n \]

\[ \log_2 n = O(\log_2 n) \cdot \log_2 n \]
Another Recurrence.

\[ T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + n^2 \]

\[ T(n) \leq n^2 \log n \]

\[ T(n) \leq O(n^2) \]

\[ \frac{2 \cdot \left(\frac{n}{2}\right)^2 + n^2}{\frac{n^2}{2}} \leq 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right) + n^2 \]

\[ h^2 + \frac{n^2}{2} - \frac{n^2}{4} + \cdots \leq 2h^2 \]
Slow Recurrences.

\[ T(n) \geq 2 \cdot T(n-1) \]

\[ \geq 2 \cdot (2 \cdot T(n-2)) \]

\[ \geq 2^{2^{\cdots \cdots}} \cdot T(1) \]

\[ = 2^n \]

\[ T(n) \geq T(n-1) + T(n-2) \]

\[ \underbrace{\text{Fibonacci}} \]

\[ \geq 2 \cdot T(n-2) \]

\[ \geq 2^{n/2} \]

\[ \geq 2 \]
Announcements.

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

* Prelim Review Sessions
  > Saturday 1-3:30pm
  > Sunday 2-4:30pm Gates GO1

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 $\pi_1, \ldots, \pi_3$

Find. The number of "inversions" in $S$
Given. A permutation $S$ of $\mathbb{Z}_1, \ldots, n$.

Find. The number of "inversions" in $S$.

1 2 3
0 inversions

1 3 2
1

2 1 3
1

2 3 1
2

3 1 2
2

3 2 1
3
Given. A permutation $S$ of $\pi_1, \ldots, \pi_3$

Find. The number of "inversions" in $S$

$S: \begin{array}{c} 5 \ 1 \ 3 \ 2 \ 4 \ 5 \end{array}$

How many crossings?
Given. A permutation \( S \) of \( \mathbb{Z}_1, \ldots, n \).

Find. The number of "inversions" in \( S \).

\[
\begin{array}{ccccccc}
1 & 2 & 3 & \downarrow & 1 & 3 & 2 & \downarrow & 2 & 1 & 3 & \downarrow & 2 & 3 & 1 & \downarrow & 3 & 1 & 2 & \downarrow & 3 & 2 & 1 \\
& & & 0 \text{ inversions} & & & 1 & \text{ inversions} & & & 1 & \text{ inversions} & & & 2 & \text{ inversions} & & & 2 & \text{ inversions} & & & 3 & \text{ inversions} \\
\end{array}
\]

\( S : 5 & 1 & 3 & 2 & 4 \)

\( i : 1 & 2 & 3 & 4 & 5 \)

How many crossings?

\[
\text{\# inversions}(S) = \left| \sum_{i \in \mathbb{Z}_1 \ldots n} \sum_{j \in \mathbb{Z}_1 \ldots n} \text{ s.t. } i < j \right| S_i > S_j
\]
Naive Algorithm?
Check All Pairs \( S \)

\[
\text{count} = 0
\]

For \( i = 1 \rightarrow n-1 \)

\[
\text{For } j = i+1 \rightarrow n
\]

\[
\text{if } S_{i} > S_{j}
\]

\[
\text{count} \leftarrow \text{count} + 1
\]

Return \( \text{count} \).
Check All Pairs (S)

Count = 0

For i = 1 --- n - 1
  For j = i + 1 --- 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 \( i = 1 \) \( \rightarrow \) \( n-1 \)
  
  For \( j = i+1 \) \( \rightarrow \) \( n \)
    
    If \( S_i > S_j \)
    
    Count \( \leftarrow \) Count \( + 1 \).

Return Count.
Observation. Sorting "inverts inversions"

For every permutation $S$

\[ \text{Sort}(S) \rightarrow I = \{1, 2, \ldots, n\} \]
Observation. Sorting "inverts inversions"

For every permutation \( S \)

\[
\text{Sort}(S) \rightarrow I = \{1, 2, \ldots, n\}
\]

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

Combine

$L$

$R$

$N_L = \# \text{inversions}(L)$

$N_R = \# \text{inversions}(R)$

$N_S = N_L + N_R + \boxed{??}$
Given $S$

Divide

Recurse

Combine

$\# \text{inversions} \left( S \right) = \# \text{inversions} \left( L \right) + \# \text{inversions} \left( R \right) + \left( \text{??} \right)$

$\# \text{inversions} \text{ across } L \leftrightarrow R$

$\left\{ i \in \left| L \right| \text{ s.t. } R_j < L_i \right\}$
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 \((L, R)\)

\[
\text{Count} = 0 \\
\text{i} = 1, \quad \text{j} = 1 \]

For \(k = 1 \rightarrow |L| + |R|\):

\[
\begin{align*}
\text{if } R_j < L_i : \\
\quad \text{count} & \leftarrow \text{count} + 1 + (|L| - i) \\
\quad j & \leftarrow j + 1 \\
\text{else:} \\
\quad i & \leftarrow i + 1
\end{align*}
\]

Return \(\text{count}\)

// Note: need to handle case where \(i/j\) go off end of \(L/R\)
Count Across Sorted Lists $(L, R)$

\[
\text{Count} = 0, \\
i = 1, j = 1. \\
\text{For } k = 1 \rightarrow |L| + |R|.
\]

\[
\begin{align*}
\text{if } R_j < L_i : \\
\quad \text{count} & \leftarrow \text{count} + 1 + (|L| - i) \\
\quad j & ++ \\
\text{else} : \\
\quad i & ++
\end{align*}
\]

Return count

// Note: need to handle case where $i$/$j$ go off end of $L/R$
Count Across Sorted Lists \((L, R)\)

\[
\begin{align*}
\text{Count} &= 0 \\
i &= 1, \quad j = 1. \\
\text{For } k = 1 \rightarrow |L| + |R|: \\
&\quad \text{if } R_j < L_i : \\
&\quad \quad \text{count} \leftarrow \text{count} + 1 + (|L| - i) \\
&\quad \quad j++ \\
&\quad \text{else} : \\
&\quad \quad i++
\end{align*}
\]

\text{Return count}

\text{Running Time?}

\begin{align*}
\phantom{= \# \text{ elems in } L} \\
\phantom{\geq L_i}
\end{align*}

// Note: need to handle case where \(i/j\) go off end of \(L/R\)
Claim. If L and R are sorted, then Count Across Sorted Lists 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 $R$.

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

Proof. 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,

$\Rightarrow i$ is the least index s.t. $R_j < L_i$. 
Claim. If \( L \) and \( R \) are sorted, then CountAcrossSortedLists returns count of \# inversions across \( L \) and \( R \).

Proof. 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,

\[ i \] 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.
Count Inversions Merge \((L, R)\)

\[ S = [\ ] \quad // \text{array of } |L| + |R| \]

Count = 0

i = 1, j = 1.

For \( k = 1 \rightarrow |L| + |R| \),

\[
\text{if } R_j < L_i : \\
\quad \text{count} \leftarrow \text{count} + 1 + (|L| - i) \\
\quad S_k \leftarrow R_j \\
\quad j++
\]

\text{else}:

\[
\quad S_k \leftarrow L_i \\
\quad i++
\]

Return \((S, \text{count})\)  // Note: need to handle case where i/j go off end of L/R
Claim: If L and R are sorted, then Count Inversions Merge returns S in sorted order.

Pf: By induction. Follows by correctness of Merge Sort.
Count Inversions Sort \((S)\).

if \(|S| \leq 1\), return \(\emptyset\).

\((L_{sorted}, n_L) \leftarrow \text{Count Inversions Sort} \((L)\)\)

\((R_{sorted}, n_R) \leftarrow \text{Count Inversions Sort} \((R)\)\)

\((S_{sorted}, n_X) \leftarrow \text{Count Inversions Merge} \((L_{sorted}, R_{sorted})\)\)

Return \((S_{sorted}, n_L + n_R + n_X)\)
Correctness. By induction,

Claim. \( \text{Count InversionsSort}(S) \) returns \( S \) in sorted order, and

\[ \text{count} = \# \text{inversions in } S \]
Correctness. By induction.

Claim: \( \text{Count InversionsSort}(S) \) returns \( S \) in sorted order, and \( \text{count} = \# \text{ inversions in } S \).

Base Case: \( |S| \leq 1 \), already sorted.
\( 0 \) inversions.
Correctness. By induction.

Claim. $\text{Count Inversions Sort}(s)$ returns $S$ in sorted order, and
\[ \text{count} = \# \text{ inversions in } S \]

Base Case. $|S| \leq 1$, already sorted.
\[ 0 \text{ inversions} \]

Induction. Follows by correctness of $\text{Count Inversions Merge}$.
Correctness. By induction.

Claim. \(\text{Count-InversionsSort}(S)\) returns \(S\) in sorted order, and \(\text{count} = \#\) inversions in \(S\).

Base Case. \(|S| \leq 1\), already sorted.

Induction. Follows by correctness of \(\text{Count-InversionsMerge}\).

Total \# inversions = \# on left + \# on right

+ \# across \(L\&R\) By IH

By correctness of \(\text{Count-InversionsMerge}\).
Running Time Analysis.

**Count Inversions Sort** ($S$)

if $|S| \leq 1$, return 0.

($L_{\text{sorted}}, n_L$) ← Count Inversions Sort ($L$) \hspace{1cm} T(n/2)

($R_{\text{sorted}}, n_R$) ← Count Inversions Sort ($R$) \hspace{1cm} T(n/2)

($S_{\text{sorted}}, n_X$) ← Count Inversions Merge ($L_{\text{sorted}}, R_{\text{sorted}}$) \hspace{1cm} ?

Return ($S_{\text{sorted}}, n_L + n_R + n_X$)
Claim. Count Inversions Sort runs in $O(n \log n)$ time.

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