Announcement: Homework 2 due on Tuesday, February 5th at 11:59PM (Gradescope)
Recap: Minimum Weight Spanning Trees

**Cut Property:** Minimum weight edge crossing a cut must be in the MST (assume edge weights are distinct)

**Cycle Property:** Maximum weight edge in a cycle must not be in the MST (assuming edge weights are distinct)

**Prim’s Algorithm**
- Repeatedly applies cut property to expand tree
- $O(m \log n)$ time with Binary Heap
- $O(m + n \log n)$ time with Fibonacci Heap

**Kruskal’s Algorithm**
- Consider edges in increasing order of weight
- $O(m \log n)$ running time.

**Union-Find Data Structure**
Divide and Conquer
Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size $n$ into two equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: $n^2$.
- Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
5.1 Mergesort
**Sorting**

*Sorting.* Given $n$ elements, rearrange in ascending order.

**Applications.**

- Sort a list of names.
- Organize an MP3 library.  
  *obvious applications*
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

- Find the median.
- Find the closest pair.  
  *problems become easy once items are in sorted order*
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.  
  *non-obvious applications*
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

...
Mergesort

Mergesort.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

\[
T(n) \leq 2 \, T\left(\frac{n}{2}\right) + O(n)
\]
Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?
- Linear number of comparisons.
- Use temporary array.

Challenge for the bored. In-place merge. [Kronrud, 1969]

using only a constant amount of extra storage
A Useful Recurrence Relation

**Def.** $T(n) =$ number of comparisons to mergesort an input of size $n$.

**Mergesort recurrence.**

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + \frac{n}{2} & \text{otherwise}
\end{cases}
\]

**Solution.** $T(n) = O(n \log_2 n)$.

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume $n$ is a power of 2 and replace $\leq$ with $=$.
Proof by Recursion Tree

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases}
\]

Merge Step Cost

- \( n \)
- \( 2(n/2) \)
- \( 4(n/4) \)
- \( \ldots \)
- \( 2^k(n/2^k) \)
- \( \ldots \)
- \( n/2 \) (2)

\[ n \log_2 n \]
Proof by Telescoping

**Claim.** If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

\[
\text{assuming } n \text{ is a power of } 2
\]

**Pf.** For $n > 1$:

\[
\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1
\]

\[
= \frac{T(n/2)}{n/2} + 1
\]

\[
= \frac{T(n/4)}{n/4} + 1 + 1
\]

\[
\vdots
\]

\[
= \frac{T(n/n)}{n/n} + 1 + \cdots + 1
\]

\[
= \log_2 n
\]
Proof by Induction

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

Pf. (by induction on $n$)

- Base case: $n = 1$.
- Inductive hypothesis: $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n(\log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]}
Analysis of Mergesort Recurrence

Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \log_2 n \rceil \).

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + n & \text{otherwise}
\end{cases}
\]

Pf. (by induction on \( n \))

- Base case: \( n = 1 \).
- Define \( n_1 = \lfloor n/2 \rfloor \), \( n_2 = \lceil n/2 \rceil \).
- Induction step: assume true for 1, 2, ..., \( n-1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lceil \log_{10} n_1 \rceil + n_2 \lceil \log_{10} n_2 \rceil + n \\
\leq n_1 \lceil \log_{10} n_2 \rceil + n_2 \lceil \log_{10} n_2 \rceil + n \\
= n \lceil \log_{10} n_2 \rceil + n \\
\leq n(\lceil \log_{10} n \rceil - 1) + n \\
= n \lceil \log_{10} n \rceil
\]

\[
n_2 = \lceil n/2 \rceil \\
\leq 2 \lceil \log_{10} n \rceil / 2 \\
= 2 \lceil \log_{10} n \rceil / 2 \\
\Rightarrow \log_{10} n_2 \leq \lceil \log_{10} n \rceil - 1
\]
More General Analysis

Merge Step Cost

\[ T(n) \]

\[ T(n/b) \]

\[ T(n/b^2) \]

\[ T(n/b^2) \]

\[ T(n/b^2) \]

\[ T(n/b^2) \]

\[ T(n / b^k) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(n) \]

\[ n^c \]

\[ a(n/b)^c = n^c(a/b^c) \]

\[ a^2(n/b^2)^c = n^c(a/b^c)^2 \]

\[ \ldots \]

\[ n^c(a/b^c)^k \]

\[ \ldots \]

\[ n^c(a/b^c)^{\log_b n} \]

\[ T(n) \leq n^c \sum_{i=0}^{\log_b n} \left( \frac{a}{b^c} \right)^i \]
A Helpful Identity

**Fact:** If \( \gamma \neq 1 \) then

\[
1 + \gamma^1 + \gamma^2 \ldots + \gamma^k = \frac{1 - \gamma^{k+1}}{1 - \gamma}
\]

**Proof**

\[
1 + \gamma^1 + \gamma^2 \ldots + \gamma^k = \frac{1 - \gamma}{1 - \gamma} (1 + \gamma^1 + \gamma^2 \ldots + \gamma^k)
\]

\[
= \frac{(1 + \gamma^1 + \gamma^2 \ldots + \gamma^k)}{1 - \gamma} - \frac{\gamma (1 + \gamma^1 + \gamma^2 \ldots + \gamma^k)}{1 - \gamma}
\]

\[
= \frac{1 + \gamma^1 + \gamma^2 \ldots + \gamma^k}{1 - \gamma} + \frac{-\gamma^1 - \gamma^2 \ldots - \gamma^k - \gamma^{k+1}}{1 - \gamma}
\]

\[
= \frac{1 - \gamma^{k+1}}{1 - \gamma}
\]
A Helpful Identity

Fact: If $\gamma \neq 1$ then

$$1 + \gamma^1 + \gamma^2 \ldots + \gamma^k = \frac{1 - \gamma^{k+1}}{1 - \gamma}$$

Observation 1: If $\gamma = 1$ then $1 + \gamma^1 + \gamma^2 \ldots + \gamma^k = k + 1 \in \Theta(k)$

Observation 2: If $0 < \gamma < 1$ then $1 + \gamma^1 + \gamma^2 \ldots + \gamma^k \approx \frac{1}{1 - \gamma} \in \Theta(1)$

Observation 3: If $1 < \gamma$ then $1 + \gamma^1 + \gamma^2 \ldots + \gamma^k \approx \frac{\gamma^{k+1}}{\gamma - 1} \in \Theta(\gamma^k)$

Observation 4: In our case $k = \log_b n$ and $\gamma = \left(\frac{a}{bc}\right)$

$$T(n) \leq n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{bc}\right)^i = n^c \left(\frac{1 - \gamma^{k+1}}{1 - \gamma}\right)$$
More General Analysis

\[
T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
 a \times T \left( \frac{n}{b} \right) + n^c & \text{otherwise}
\end{cases} \\
\text{Case 1: } \gamma = \left( \frac{a}{b^c} \right) = 1 \quad T(n) \leq n^c \sum_{i=0}^{\log_b n} \left( \frac{a}{b^c} \right)^i \\
= n^c \log_b n
\]
More General Analysis

Merge Step Cost

\[ n^c \]
\[ a(n/b)^c = n^c(a/b^c) \]
\[ n^c(a/b^c)^2 \]
\[ \ldots \]
\[ n^c(a/b^c)^k \]
\[ \ldots \]
\[ n^c(a/b^c)^{\log_b n} \]

Case 2: \( \gamma = \left( \frac{a}{b^c} \right) < 1 \)

\[ T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
 a \times T\left(\frac{n}{b}\right) + n^c & \text{otherwise}
\end{cases} \]

\[ n^c \left( \frac{\gamma^i}{(1 - \gamma)} \right) \leq \frac{n^c}{(1 - \gamma)} \]
More General Analysis

\[ T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
\frac{a}{b} T\left(\frac{n}{b}\right) + n^c & \text{otherwise}
\end{cases} \quad \text{Case 3: } \gamma = \left(\frac{a}{b^c}\right) > 1
\]

\[ T(n) \leq \sum_{i=0}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = \Theta\left(n^{\log_b a}\right) \]

\[ n^c \left(\frac{a}{b^c}\right)^{\log_b n} = \left(\frac{n^c}{b^c \log_b n}\right) \times a^{\log_b n} = n^{\log_b a} \]
Implications for Divide and Conquer Analysis

- **Merge Cost:** $O(n^c)$ (want $c$ to be small)
- **Branching Factor:** $a$ (smaller branching factor $\rightarrow$ faster)
- **Reduction in Input Size:** $b$ (bigger is better)
  - **Key Ratio:** $a/b^c$

\[
T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
 a \times T\left(\frac{n}{b}\right) + n^c & \text{otherwise}
\end{cases}
\]

**Case 1:** \(\left(\frac{a}{b^c}\right) < 1\) \(T(n) = \Theta(n^c)\)

**Case 2:** \(\gamma = \left(\frac{a}{b^c}\right) = 1\) \(T(n) = \Theta(n^c \log n)\)

**Case 3:** \(\left(\frac{a}{b^c}\right) > 1\) \(T(n) = \Theta(n^{\log_b a})\)
Implications for Divide and Conquer Analysis

- **Merge Cost:** $O(n^c)$ (want $c$ to be small)

- **Branching Factor:** $a$ (smaller branching factor $\rightarrow$ faster)

- **Reduction in Input Size:** $b$ (bigger is better)
  - **Key Ratio:** $a/b^c$

\[
T(n) \leq \begin{cases} 
1000000 & \text{if } n \leq 100 \\
 a \times T\left(\frac{n}{b} + 50\right) + n^c & \text{otherwise}
\end{cases}
\]

- **Case 1:** $\left(\frac{a}{b^c}\right) < 1 \quad T(n) = \Theta(n^c)$
- **Case 2:** $\gamma = \left(\frac{a}{b^c}\right) = 1 \quad T(n) = \Theta(n^c \log n)$
- **Case 3:** $\left(\frac{a}{b^c}\right) > 1 \quad T(n) = \Theta(n^{\log_b a})$
Other Recurrences

- \( T(n) = T(n - 1) + 1 \) \hspace{1cm} (Unroll: \( T(n) = n \))

- \( T(n) = 2 \times T(n - 10) \) \hspace{1cm} (Exponential)

- \( T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{2n}{3}\right) + n \)
  \hspace{1cm} (Solution: \( T(n) \in \Theta(n) \))

- \( T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n \)
  \hspace{1cm} (Solution: \( T(n) \in \Theta(n \log n) \))
Other Recurrences

- \( T(n) = 2 \times T(n - 10) \) (Exponential)
  - Two branches
  - Only constant reduction in input size

\[
T(n) = \Theta(c^n)
\]

- How to find \( c \)? [Trick]

\[
2 = \frac{T(n)}{T(n - 10)} = \frac{c^n}{c^{n-10}}
\]

\[
\rightarrow c^{10} = 2
\]

\[
\rightarrow c = ^{10}\sqrt{2} \approx 1.07177
\]

- **Must verify solution by induction**
5.3 Counting Inversions
Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

**Similarity metric:** number of inversions between two rankings.

- My rank: 1, 2, ..., n.
- Your rank: \(a_1, a_2, ..., a_n\).
- Songs \(i\) and \(j\) inverted if \(i < j\), but \(a_i > a_j\).

### Counting Inversions

<table>
<thead>
<tr>
<th>Songs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Inversions:

\[3-2, 4-2\]

**Brute force:** check all \(\Theta(n^2)\) pairs \(i\) and \(j\).
Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

1 5 4 8 10 2 6 9 12 11 3 7

Divide: $O(1)$.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide:** separate list into two pieces.
- **Conquer:** recursively count inversions in each half.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: \( O(1) \).

Conquer: \( 2T(n/2) \)

- 5 blue-blue inversions
- 8 green-green inversions
- 5-4, 5-2, 4-2, 8-2, 10-2
- 6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.
- **Combine**: count inversions where $a_i$ and $a_j$ are in different halves, and return sum of three quantities.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: $O(1)$.

Conquer: $2T(n/2)$

Combine: ???

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = $5 + 8 + 9 = 22$. 
Counting Inversions: Combine

**Combine:** count blue-green inversions

- Assume each half is *sorted*.
- Count inversions where \( a_i \) and \( a_j \) are in different halves.
- **Merge** two sorted halves into sorted whole.

\[
\begin{array}{cccccc}
3 & 7 & 10 & 14 & 18 & 19 \\
\end{array}
\begin{array}{cccccc}
2 & 11 & 16 & 17 & 23 & 25 \\
6 & 3 & 2 & 2 & 0 & 0 \\
\end{array}
\]

13 blue-green inversions: \( 6 + 3 + 2 + 2 + 0 + 0 \)  
Count: \( O(n) \)

\[
\begin{array}{cccccccccccc}
2 & 3 & 7 & 10 & 11 & 14 & 16 & 17 & 18 & 19 & 23 & 25 \\
\end{array}
\]

Merge: \( O(n) \)

\[
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)
\]
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    (r_A, A) ← Sort-and-Count(A)
    (r_B, B) ← Sort-and-Count(B)
    (r, L) ← Merge-and-Count(A, B)

    return r = r_A + r_B + r and the sorted list L
}
5.4 Closest Pair of Points
Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.
  - Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
  - Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

\[ \uparrow \]
  to make presentation cleaner
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.
Closest Pair of Points: First Attempt

Divide.  Sub-divide region into 4 quadrants.
Obstacle. Impossible to ensure n/4 points in each piece.
Closest Pair of Points

Algorithm.
- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
Closest Pair of Points

Algorithm.

- **Divide**: draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.
Closest Pair of Points

**Algorithm.**
- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$. 

$\delta = \min(12, 21)$
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < \( \delta \).
- Observation: only need to consider points within \( \delta \) of line \( L \).

\[ \delta = \min(12, 21) \]
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance \( \delta \).

- Observation: only need to consider points within \( \delta \) of line \( L \).
- Sort points in \( 2\delta \)-strip by their y coordinate.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < \( \delta \).

- Observation: only need to consider points within \( \delta \) of line L.
- Sort points in \( 2\delta \)-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!

\[ \delta = \min(12, 21) \]
Closest Pair of Points

**Def.** Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest $y$-coordinate.

**Claim.** If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

**Pf.**
- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$.

**Fact.** Still true if we replace 12 with 7.
Closest Pair Algorithm

Closest-Pair(p₁, ..., pₙ) {
    \textbf{Compute} separation line \( L \) such that half the points are on one side and half on the other side.

    \( \delta_1 = \text{Closest-Pair(left half)} \)
    \( \delta_2 = \text{Closest-Pair(right half)} \)
    \( \delta = \min(\delta_1, \delta_2) \)

    \textbf{Delete} all points further than \( \delta \) from separation line \( L \)

    \textbf{Sort} remaining points by y-coordinate.

    \textbf{Scan} points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).

    \textbf{return} \( \delta \).
}
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n) \]

Q. Can we achieve \( O(n \log n) \)?

A. Yes. Don't sort points in strip from scratch each time.
   - Each recursive returns two lists: all points sorted by \( y \) coordinate, and all points sorted by \( x \) coordinate.
   - Sort by merging two pre-sorted lists.

\[ T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]