Announcement: Homework 2 due on Tuesday, February 6th at 11:59PM
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

Prim’s Algorithm
- Consider edges in increasing order of weight
- For each edge we can either
  - Discard via Cycle Property, or
  - Add via Cut Property
- \(O(m \log n)\) running time.
Greedy Algorithms
4.7 Clustering
Clustering. Given a set $U$ of $n$ objects labeled $p_1, \ldots, p_n$, classify into coherent groups.

Distance function. Numeric value specifying "closeness" of two objects.

Fundamental problem. Divide into clusters so that points in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster $10^9$ sky objects into stars, quasars, galaxies.
Clustering of Maximum Spacing

**k-clustering.** Divide objects into $k$ non-empty groups.

**Distance function.** Assume it satisfies several natural properties.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

**Spacing.** Min distance between any pair of points in different clusters.

**Clustering of maximum spacing.** Given an integer $k$, find a $k$-clustering of maximum spacing.

$k = 4$
Greedy Clustering Algorithm

**Single-link k-clustering algorithm.**
- Form a graph on the vertex set $U$, corresponding to $n$ clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat $n-k$ times until there are exactly $k$ clusters.

**Key observation.** This procedure is precisely Kruskal's algorithm (except we stop when there are $k$ connected components).

**Remark.** Equivalent to finding an MST and deleting the $k-1$ most expensive edges.

$\text{Graph } G \text{ has } O(n^2) \text{ edges}$

$O(n^2 \log n)$
**Theorem.** Let $C^*$ denote the clustering $C^*_{1}, \ldots, C^*_{k}$ formed by deleting the 
$k-1$ most expensive edges of a MST. $C^*$ is a $k$-clustering of max spacing.

**Pf.** Let $C$ denote some other clustering $C_{1}, \ldots, C_{k}$.

- The spacing of $C^*$ is the length $d^*$ of the $(k-1)^{\text{st}}$ most expensive edge.
- Let $p_{i}, p_{j}$ be in the same cluster in $C^*$, say $C^*_{r}$, but different clusters 
in $C$, say $C_{s}$ and $C_{t}$.
- Some edge $(p, q)$ on $p_{i}-p_{j}$ path in $C^*_{r}$ spans two different clusters in $C$.
- All edges on $p_{i}-p_{j}$ path have length $\leq d^*$ 
since Kruskal chose them.
- Spacing of $C$ is $\leq d^*$ since $p$ and $q$ 
are in different clusters. □
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.  \(\text{obvious applications}\)
- Display Google PageRank results.
- List RSS news items in reverse chronological order.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

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

...
Mergesort.

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

\[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \]
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.** $T(n) = \text{number of comparisons to mergesort an input of size } n.$

**Mergesort recurrence.**

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

**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

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

mergesort steps:

\[
\begin{align*}
T(n) &= T(n/2) + T(n/2) \\
&= 2T(n/2) \\
&= 4T(n/4) \\
&= 2^k T(n/2^k) \\
&= n\log_2 n
\end{align*}
\]
Proof by Telescoping

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

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

assumes \( n \) is a power of 2

**Pf.** For \( n > 1 \):

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

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

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

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

\[
= \log_2 n
\]

\( T(n) = n \log_2 n \)
Proof by Induction

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

**Guess**

$T(n) \leq C \cdot n^d$

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

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

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n(\log_2(2n) - 1) + 2n = 2n \log_2 (2n)
\]

\[
T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{2n}{3}\right) \rightarrow n
\]

assumes $n$ is a power of 2
Analysis of Mergesort Recurrence

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

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

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 \( 1, 2, \ldots, n-1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1\lceil \lg n_1 \rceil + n_2\lfloor \lg n_2 \rfloor + n \\
\leq n_1\lfloor \lg n_2 \rfloor + n_2\lfloor \lg n_2 \rfloor + n \\
= n\lfloor \lg n_2 \rfloor + n \\
\leq n(\lfloor \lg n \rfloor - 1) + n \\
= n\lceil \lg n \rceil
\]

\[
n_2 = \lfloor n/2 \rfloor \\
\leq 2\lceil \lg n \rceil / 2 \\
= 2\lceil \lg n \rceil / 2 \\
\Rightarrow \lg n_2 \leq \lceil \lg n \rceil - 1
\]
**More General Analysis**

\[ T(n) \]

\[ T(n/b) \]

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

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

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

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

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

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

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

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

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

\[ \log_b n \]

\[ n^c \]

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

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

\[ \ldots \]

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

\[ \ldots \]

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

\[ T(n) \leq \sum_{i=0}^{\log_b n} n^c(a/b^c)^i \]

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

\[ T(n) \]

\[ T(n/b) \]

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

\[ T(n/b^3) \]

\[ T(n/b^4) \]

\[ \ldots \]

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

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

\[ T(2) \]

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

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

\[ \ldots \]

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

\[ \ldots \]

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

\[ n^c \]

\[ a \times T\left(\frac{n}{b}\right) + n^c \]

if \( n = 1 \)

otherwise

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

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

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

\[ = n^c \log_b n \]
More General Analysis

\[ T(n) \]

\[ T(n/b) \]

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

\[ T(n/b^3) \]

\[ T(n/b^4) \]

\[ \ldots \]

\[ T(2) \]

\[ \ldots \]

\[ \log_b n \]

\[ n^c \]

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

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

\[ \ldots \]

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

\[ \ldots \]

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

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

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

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

\[ = \Theta(n^c) \]
More General Analysis

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

Case 3: \( \left(\frac{a}{b^c}\right) > 1 \)

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

\[
= \Theta\left(n^{\log_b a}\right)
\]
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: a₁, a₂, ..., aₙ.
- Songs i and j inverted if i < j, but aᵢ > aⱼ.

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

<table>
<thead>
<tr>
<th></th>
<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>
<tbody>
<tr>
<td>Divide:</td>
<td>O(1).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<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>
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>

5 blue-blue inversions
5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions
6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Divide: \(O(1)\).

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

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.

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

To maintain sorted invariant

\[
T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left| \frac{n}{2} \right| \right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
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.

\[ \text{fast closest pair inspired fast algorithms for these problems} \]
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. $\rightarrow$ 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.

\[ \delta = \min(12, 21) \]
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.
- Only check distances of those within 11 positions in sorted list!
**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) {

  **Compute** separation line \( L \) such that half the points are on one side and half on the other side.

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

  **Delete** all points further than \( \delta \) from separation line \( L \)

  **Sort** remaining points by \( y \)-coordinate.

  **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 \( \delta \).

  **return** \( \delta \).
}

\( O(n \log n) \)

\( 2T(n / 2) \)

\( O(n) \)

\( O(n \log n) \)

\( O(n) \)
Closest Pair of Points: Analysis

Running time.

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

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.

\[ T(n) \leq 2T(n/2) + O(n) \implies T(n) = O(n \log n) \]
MST Algorithms: Theory

Deterministic comparison based algorithms.

- $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$. [Cheriton-Tarjan 1976, Yao 1975]
- $O(m \beta(m, n))$. [Fredman-Tarjan 1987]
- $O(m \log \beta(m, n))$. [Gabow-Galil-Spencer-Tarjan 1986]
- $O(m \alpha(m, n))$. [Chazelle 2000]

Holy grail. $O(m)$.

Notable.

- $O(m)$ randomized. [Karger-Klein-Tarjan 1995]
- $O(m)$ verification. [Dixon-Rauch-Tarjan 1992]

Euclidean.

- 2-d: $O(n \log n)$. compute MST of edges in Delaunay
- k-d: $O(k n^2)$. dense Prim
Dendrogram. Scientific visualization of hypothetical sequence of evolutionary events.

- Leaves = genes.
- Internal nodes = hypothetical ancestors.

Dendrogram of Cancers in Human

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group

Gene 1

Gene n

| gene expressed | gene not expressed |
Extra Slides