Recap: Minimum Weight Spanning Trees

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

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 10^6 sky objects into stars, quasars, galaxies.
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.

Greedy Clustering Algorithm: Analysis

**Theorem.** Let $C_1^* \ldots C_k^*$ denote the clustering, formed by deleting the k-1 most expensive edges of a MST, $C$ is a k-clustering of max spacing.

**Proof.** Let $C$ denote some other clustering $C_1 \ldots C_k$.
- The spacing of $C$ is the length $d^*$ of the k-th most expensive edge.
- Let $p, q$ be in the same cluster in $C$, say $C_r$, but different clusters in $C$. Let $C_s$ and $C_t$.
- Any edge on the path between $p$ and $q$ will have length $\geq d^*$ since Kruskal’s rule.
- Spacing of $C \leq d^*$ since $p$ and $q$ are in different clusters.

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 $\frac{n}{2}$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

**Consequence.**
- Brute force $n^2$.
- Divide and conquer: $\log n$. — Julius Caesar

5.1 Mergesort

**Divide and Conquer**

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

**Applications.**
- Sort a list of names.
- Organize an MP3 library.
- Do you need to sort some data?
- List RSS news items in reverse chronological order.
- 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.

Greedy Clustering Algorithm: Analysis

**Theorem.** Let $C_1^* \ldots C_k^*$ denote the clustering, formed by deleting the k-1 most expensive edges of a MST, $C$ is a k-clustering of max spacing.

**Proof.** Let $C$ denote some other clustering $C_1 \ldots C_k$.
- The spacing of $C$ is the length $d^*$ of the k-th most expensive edge.
- Let $p, q$ be in the same cluster in $C$, say $C_r$, but different clusters in $C$. Let $C_s$ and $C_t$.
- Any edge on the path between $p$ and $q$ will have length $\geq d^*$ since Kruskal’s rule.
- Spacing of $C \leq d^*$ since $p$ and $q$ are in different clusters.

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 \( \frac{n}{2} \).
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

**Consequence.**
- Brute force $n^2$.
- Divide and conquer: $\log n$. — Julius Caesar
Mergesort

Divide array into two halves.

Recursively sort each half.

Merge two halves to make sorted whole.

A Useful Recurrence Relation

Def. \( T(n) = \) number of comparisons to mergesort an input

of size \( n \).

Mergesort recurrence.

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

Assorted proofs. We describe several ways to prove this

currence. Initially we assume \( n \) is a power of 2 and replace

\( \leq \) with =.

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

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

Proof by Recursion Tree

Proof by Telescoping

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

Pf. For \( n = 1 \):

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

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

\[ T(n) = \frac{n \log_2 n}{\log_2 2} = \frac{n \log_2 (2n)}{2} \]

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

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. [Kraus, 1965]

Proof by Induction

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

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

- Base case: \( n = 1 \)
- Inductive hypothesis: \( T(2k) = 2k \log_2 (2k) \)

Goal: show that \( T(2k) + 2k \log_2 (2k) \)
Analysis of Mergesort Recurrence

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

Proof (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, ..., $n-1$.

$T(n) \leq T(n_1) + T(n_2) + n$

$\leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$

$\leq n_1 \log_2 n_1 + n_2 \log_2 n_2 + n$

$\leq n \log_2 n$

More General Analysis

Case 1: $\frac{n}{b^k} = n/2$.

Case 2: $\frac{n}{b^k} < n/2$.

Case 3: $\frac{n}{b^k} > n/2$.

5.3 Counting Inversions
Counting Inversions

You rank \( n \) songs.

Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: \( a_1, a_2, \ldots, a_n \).
- Your rank: \( b_1, b_2, \ldots, b_n \).

Songs \( i \) and \( j \) inverted if \( i < j \), but \( a_i > b_j \).

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

### Applications

- Voting theory.
- Collaborative filtering.
- 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.

Divide:
- separate list into two pieces.

Conquer:
- recursively count inversions in each half.

Combine:
- count inversions where \( a_i \) and \( b_j \) are in different halves, and return sum of three quantities.

<table>
<thead>
<tr>
<th>Songs</th>
<th>Inversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3, 2, 4, 2</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

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

---

**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 \( b_j \) are in different halves, and return sum of three quantities.

Total: \( 5 \times 8 + 9 = 47 \)
### 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.

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

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

- Count: \( O(n) \)
- Merge: \( O(n) \)

### Counting Inversions: Implementation

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

```python
def 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:
    (rA, A) = Sort-and-Count(A)
    (rB, B) = Sort-and-Count(B)
    (rB, L) = Merge-and-Count(A, B)
    return r = rA + rB + r and the sorted list L
```

### 5.4 Closest Pair of Points

#### 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: First Attempt

- **Divide**: sub-divide region into 4 quadrants.

#### Closest Pair of Points

- **Closest pair**: given \( n \) points in the plane, find a pair with smallest Euclidean distance between them.
- **Pre-condition**: \( n \) points in the plane
- **Post-condition**: \( n \) points in the plane

### Closest Pair of Points

- **Brute force**: check all pairs of points \( p \) and \( q \) with \( \Omega(n^2) \) comparisons.
- **1-D version**: \( O(n \log n) \) easy if points are on a line.
- **Assumption**: no two points have same \( x \) coordinate.

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

- **Fast closest pair inspirations**:
  - Brute force: check all pairs of points \( p \) and \( q \) with \( \Omega(n^2) \) comparisons.
  - 1-D version: \( O(n \log n) \) easy if points are on a line.
  - Assumption: no two points have same \( x \) coordinate.

- **Fast closest pair inspired fast algorithms**:
  - Brute force: check all pairs of points \( p \) and \( q \) with \( \Omega(n^2) \) comparisons.
  - 1-D version: \( O(n \log n) \) easy if points are on a line.
  - Assumption: no two points have same \( x \) coordinate.

- **Fast closest pair**: given \( n \) points in the plane, find a pair with smallest Euclidean distance between them.

### Closest Pair of Points

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

```python
def 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:
    (rA, A) = Sort-and-Count(A)
    (rB, B) = Sort-and-Count(B)
    (rB, L) = Merge-and-Count(A, B)
    return r = rA + rB + r and the sorted list L
```
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. Return best of 3 solutions.

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.
- Only check distances of those within 11 positions in sorted list!

\( \delta = \min(12, 21) \)

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

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 return two lists: all points sorted by y coordinate,
  and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

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

MST Algorithms: Theory

Deterministic comparison based algorithms.
- \( O(m \log n) \) \([Jarník, Prim, Dijkstra, Kruskal, Boruvka]\)
- \( O(m \log \log n) \). \([Cheriton-Tarjan 1976, Yao 1975]\)
- \( O(m \beta(m, n)) \). \([Fredman-Tarjan 1987]\)
- \( O(m \log \beta(m, n)) \). \([Gabow-Gall- Spencer-Tarjan 1986]\)
- \( O(m = (m, n)) \). \([Chazelle 2000]\)

Holy grail. \( O(n) \).

Notable.
- \( O(m) \) randomized. \([Karger-Klein-Tarjan 1995]\)
- \( O(m) \) verification. \([Dixon-Rausch-Tarjan 1992]\)

Euclidean.
- 2-d: \( O(n \log n) \). compute MST of edges in Delaunay
- k-d: \( O(n \log^k n) \), dense Prim

Dendrogram

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

Extra Slides