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**
  - 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: \(n \log n\).

---

**5.1 Mergesort**

**Sorting**

- **Sorting** Given \(n\) elements, rearrange in ascending order.

- **Applications**
  - Sort a list of names.
  - Organize an MP3 library.
  - 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.

---

Copyright 2000, Kevin Wayne
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.

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

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

Proof by Telescoping

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

Pf. For \( n = 1 \):

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

obtained n is a power of 2

Proof by Induction
Analysis of Mergesort Recurrence

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

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 $1, 2, \ldots, n-1$.

$$T(n) \leq T(n_1) + T(n_2) + n \leq n_1 \lg n_1 + n_2 \lg n_2 + n \leq n_1 \lg n_2 + n_2 \lg n_2 + n = \lg n \leq \lg n_2 \leq \lceil \lg n \rceil.$$  

More General Analysis

$$T(n) \leq \sum_{i=0}^{\log_2 n} n \cdot \frac{1}{2^i}$$

A Helpful Identity

Fact: If $\gamma \leq 1$ then

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

Proof

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

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 $1 + \gamma^1 + \gamma^2 + \ldots + \gamma^k = \frac{1 - \gamma^{k+1}}{1 - \gamma} \in \Theta(1)$

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

Observation 4: In our case $k = \log_2 n$ and $\gamma = \left(\frac{1}{2}\right)$

$$T(n) \leq \sum_{i=0}^{\log_2 n} n \cdot \frac{1}{2^i} = \Theta(n \log_2 n)$$
More General Analysis

$$T(n) = T(n/b) + T(n/b^2) + \cdots + T(n/b^k) + \text{merge cost}$$

Mergesort - Divide & Conquer - Implications for Divide and Conquer Analysis

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

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ a \times T\left(\frac{n}{b}\right) + w' & \text{otherwise} \end{cases}$$

Case 1: $\frac{n}{b^n} \leq 1 \Rightarrow T(n) = \Theta(n^c)$
Case 2: $\gamma = \frac{c}{b} < 1 \Rightarrow T(n) = \Theta(n^c \log n)$
Case 3: $\gamma > 1 \Rightarrow T(n) = \Theta(n^c \log n)$

Other Recurrences

- $T(n) = 2 \times T(n - 10)$ (Exponential)
- $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$ (Solution: $T(n) \in \Theta(n \log n)$)

5.3 Counting Inversions
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.

<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>

Brute force: check all n(n⁻¹) pairs i and j.

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

Divide: separate list into two pieces.

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

Conquer: recursively count inversions in each half.

Divide: O(1).

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

Conquer: 2T(n / 2)

5 blue-blue inversions
5-4, 5-2, 4-2, 8-6, 8-2, 10-2

8 green-green inversions
6-5, 9-5, 7-9, 7-12, 3-12, 12-9, 11-3, 11-7

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

Total = 5 + 8 + 9 = 22.

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: 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.

\[
\begin{array}{cccccc}
1 & 7 & 10 & 14 & 18 & 19 \\
2 & 11 & 16 & 17 & 23 & 25 \\
\end{array}
\]

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

To maintain sorted invariant

Count Inv. = \( O(n) \)

Combine: \( O(n \log n) \)

Count Inv. = \( O(n^2) \)

\[
T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) = \Omega(n^2) = \Theta(n^2)
\]

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 \( B \)
  \( (r_A, A) \leftarrow \text{Sort-and-Count}(A) \)
  \( (r_B, B) \leftarrow \text{Sort-and-Count}(B) \)
  \( (r, L) \leftarrow \text{Merge-and-Count}(A, B) \)
- return \( r = r_A + r_B + r \) and the sorted list \( L \)

5.4 Closest Pair of Points

Closest pair. 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.

Fast closest pair inspired fast algorithm for these problems

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.

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.
- **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 $L$.
- 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 < δ.
- Observation: only need to consider points within δ of line L.
- Only check distances of those within 11 positions in sorted list!

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.
  δ1 = Closest-Pair(left half)
  δ2 = Closest-Pair(right half)
  δ = min(δ1, δ2)
  Delete all points further than δ 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 δ, update δ.
  return δ.
}

Closest Pair of Points: Analysis

Running time.

T(n) ≤ 2T(n/2) + O(n log n) ⇒ T(n) = O(n log² n)

Can we achieve O(n log n)?

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) ≤ 2T(n/2) + O(n) ⇒ T(n) = O(n log n)