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: 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((n+m 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

- 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: $n \log n$.

Divide et impera.

Veni, vidi, vici.

-Julius Caesar

5.1 Mergesort

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
- Database management
- 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

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

A Useful Recurrence Relation

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

Mergesort recurrence.

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

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

Proof by Telescoping

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

Proof. For \( n = 2^k \):

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

Pf. By induction on \( n \)

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

Copyright 2000, Kevin Wayne
Analysis of Mergesort Recurrence

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

**Proof:** (by induction on $n$)
- Base case: $n = 1$.
- Define $n_1 = \lceil n/2 \rceil$, $n_2 = \lfloor n/2 \rfloor$.
- Induction step: assume true for $1, 2, \ldots, n-1$.

$$T(n) \leq T(n_1) + T(n_2) + n \leq \frac{n}{2} \lceil \lg n \rceil + \frac{n}{2} \lfloor \lg n \rfloor + n \leq n \lceil \lg n \rceil + n \leq n \lfloor \lg n \rfloor + n = n \lceil \lg n \rceil - 1$$

More General Analysis

**Fact:** If $\gamma \neq 1$ then
$$1 + \gamma^1 + \gamma^2 + \cdots + \gamma^k = \frac{1 - \gamma^{k+1}}{1 - \gamma}$$

**Proof**
$$1 + \gamma^1 + \gamma^2 + \cdots + \gamma^k = \frac{1 - \gamma^{k+1}}{1 - \gamma} = \frac{1 - \gamma^{k+1} - \gamma^{k+1}}{1 - \gamma} = \frac{1 - \gamma^{k+1}}{1 - \gamma}$$

**Observation 1:** If $\gamma = 1$ then $1 + \gamma^1 + \gamma^2 + \cdots + \gamma^k = k + 1 \in \Theta(k)$
**Observation 2:** If $0 < \gamma < 1$ then $1 + \gamma^1 + \gamma^2 + \cdots + \gamma^k = \frac{1}{1 - \gamma} \in \Theta(1)$
**Observation 3:** If $1 < \gamma$ then $1 + \gamma^1 + \gamma^2 + \cdots + \gamma^k = \frac{1}{1 - \gamma} \in \Theta(k)$
**Observation 4:** In our case $k = \log_b n$ and $\gamma = \left(\frac{a}{b}\right)$

$$T(n) \leq m \sum_{i=0}^{\log_b n} \left(\frac{a}{b}\right)^i = \Theta\left(\frac{a^n b^{\log_b n}}{b-1}\right)$$
More General Analysis

$T(n) = T(n/b) + T(n/b^2) + \ldots + T(n/b^k)$

Implications for Divide and Conquer Analysis

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

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

Case 3: $\gamma = \left(\frac{n}{b^c}\right) > 1$

$\sum_{i=0}^{\log_{b^c} n} n / b^{ci} = \Theta(n \log n)$

Other Recurrences

- $T(n) = 2^k T(n/k) + n$ (Exponential)
  - 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)/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 ai and aj are in different halves, and return sum of three quantities.

<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>

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

Divide: O(n).
Conquer: 2T(n/2)
Combine: ???

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.

- 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 $B$
  ($r_A$, $A$) \text{ } \Leftarrow \text{ Sort-and-Count($A$)}
  ($r_B$, $B$) \text{ } \Leftarrow \text{ Sort-and-Count($B$)}
  ($r_B$, $L$) \text{ } \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 algorithms 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.

\[ \Theta(n^2) \]

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 \( L \).
- Only check distances of those within 11 positions in sorted list!

\[ s = \min(12, 21) \]

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

Closest Pair Algorithm

```c
Closest-Pair(p_1, ..., p_n) {
    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 \( \delta \).
}
```

Closest Pair: Analysis

Running time.

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

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.

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