Stable Matching

- Does it terminate - does it progress
- Is it correct
- Running time
- Is this the best running time? Optimal?
- Is this the best answer?
- Is this problem realistic?
- Is there an answer?
Running Time

Big-oh notation - depends on data size
- ignore constants - because they are machine dependent
- looking at the number of elementary operations

\[ n = \Theta(0.5n) \]

is \( n \) better than \( \frac{1}{2} n \)?
maybe!
Big-Oh

does not tell us which algorithm is faster!

\[ f(n) = \Theta(g(n)) \]

is \( f(n) \) better?

\[ f(n) \leq c \cdot g(n) \quad \exists c \quad \forall n \geq n_0 \]

\[ n = O(0.5n) \]

- \( f(n) \) is no worse than \( g(n) \)

\( f(n) \) grows at the same rate or more slowly than \( g(n) \)

\[ f(n) = o(g(n)) \]

- \( f(n) \) is "better"

\[ f(n) \leq c \cdot g(n) \quad \forall c \quad \forall n \geq n_0 \]

- \( f(n) \) grows more slowly than \( g(n) \)
\[ f(n) = o(g(n)) \quad \Rightarrow \quad g(n) = \Omega(f(n)) \]

\[ f(n) \preceq g(n) \quad \Rightarrow \quad f(n) \text{ grows at the same rate or faster than } g(n). \]

\[ f(n) = \omega(g(n)) \]

\[ f(n) \text{ grows faster than } g(n) \]
Graph Algorithms

A graph can be weighted.

\[ a - 1 - b \]
\[ 0 - 6 - 0 - 3 - 0 \]
\[ u \rightarrow v \] directed
\[ 0 \rightarrow 0 \] undirected

\[ 0 \rightarrow v \]
Graph Exploration

- Keep a set of explored vertices and edges
- Never revisit an explored vertex or edge
- Avoids endless loops!
- Our algorithms terminate

BFS - queue - always explore the last seen edge
DFS - stack - always explore the most recently visited edge
Dijkstra's - priority queue - shortest distance so far
Prim's - PA - smallest edge
Dijkstra's may fail on negative edge weights

BFS what are the paths from the start \( s \) \( O(n \cdot m) \)

DFS \( O(n \cdot m) \)

Dijkstra's - shortest paths \( O((n \cdot m) \log n) \)

Prim's - minimum spanning trees \( O((n \cdot m) \log n) \)

Kruskal's - union find structure \( \Theta(n \cdot \log n) \)
Connected components

Strongly connected components

Find cycles

Find odd-length cycles
Greedy Algorithms

- have a set of options
  - pick the one that is "best"
    - maximizes some value

  optimization problems

- proving correct choice
  - stay-ahead - essentially induction
    - induction directly
Interval Scheduling
Shortest Paths
Minimum Spanning Trees

- They don't always work!

\[ \begin{align*}
0 & \rightarrow 1 \\
1 & \rightarrow 0, 6 \\
0 & \rightarrow 0
\end{align*} \]
Divide and Conquer

recursive algorithms

1. break problem into subproblems Divide
2. solve the subproblems Recurse
3. Combine the solutions Conquer

Merge-Sort

split into L and R
merge L and R
Quicksort

\[ m \in \text{median} \]

split into \( L = \{ x \leq m \} \) and \( R = \{ x > m \} \)

recursively on \( L \) and \( R \)

already sorted! \( L \cap R \)
Running Times?

Recurrence relation:

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + f(n) \]

- \( a \) = number of subproblems
- \( b \) = amount we divide the subproblems
- \( f(n) \) = amount of work done in one recursive call ignoring the recursion

merge sort

\[ T(n) = 2 \cdot T \left( \frac{n}{3} \right) + \Theta(n) \]
Solve a recurrence relation?

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

\[ T(n) = 9T\left(\frac{n}{9}\right) + \eta \]

1. Master Theorem
2. Recurrence Tree
3. Substitution
Master Theorem

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \]

Case 1: \( f(n) = O(n^\log_b a - \epsilon) \) \( T(n) = \Theta(n^\log_b a) \) in the recommended number of leaves

Case 2: \( f(n) = \Theta(n^\log_b a) \) \( T(n) = \Theta(f(n)) \log n \)

Case 3: \( f(n) = \Omega(n^\log_b a + \epsilon) \)

Regularity condition: \( a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \)

\[ T(n) = \Theta(f(n)) \]
Recurrence Tree

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

\[ \sum_{i=0}^{\log_2 n} n = cn \log_2 n + bn \]

\[ T(n) = \Theta(n \log n) \]

\[ \log_2 n \]

\[ \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \]

\[ f(n) = n \]

\[ 2f\left(\frac{n}{2}\right) = n \]

\[ 4f\left(\frac{n}{4}\right) = cn \]

\[ 2^i f\left(\frac{n}{2^i}\right) = n \]
\[ T(h) = T \left( \frac{n}{2} \right) + T \left( \frac{n}{3} \right) \]
\[ T(n) = 2T \left( \frac{n}{3} \right) + n \]

**Guess:** \( \mathcal{O}(n \log n) \)

**Base case:** trivial

**Substitution**

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

\[ T(n) \leq 2c \frac{n}{3} \log \frac{n}{3} + n \]

\[ \leq c n \log n - c n \log 3 + n \]

\[ \leq c n \log n + (1 - c)n \]

\[ \leq c n \log n \]
Prove Correctness of Recursive Algorithms

Induction

- Base case - usually trivial
- Inductive hypothesis
  - Assume the algorithm is correct for input size \( \leq N \)
  - Divide step
  - Recurse step - by IH
  - Combine step
Merge sort (A, start, end)

\[ \text{if } \text{start} = \text{end} \]  
\[ \text{middle } m \text{ is middle} \]
\[ \text{if start } \leq \text{end} \]
\[ O(1) \]

Merge sort (A, start, m) \( T(\frac{n}{2}) \)

Merge sort (A, m+1, end) \( T(\frac{n}{2}) \)

merge (A, start, m, end) \( O(n) \)

Correctness:

Inductive hypothesis: Merge sort sorts \( A \) \( n \) for \( |A| < \text{end} - \text{start} \)

Base case: \( \text{end} - \text{start} = 0 \) so \( \text{start} = \text{end} \) and we have a single element which is already sorted
inductive step:

we know by the IH that the left half and right half will be sorted because \( m - \text{start} \leq \text{end} - \text{start} \).

we also know that merge is correct.

\[ A \text{[start, end]} \] is sorted when the function returns.