Alternate Analysis

Use amortized analysis on multiple calls to push, pop:

Cannot pop more elements than get pushed onto the stack.

After many pushes, a single pop has high potential.

Once that potential has been expended, it is not available for future pop operations.

The cost for \( m_1 \) pushes and \( m_2 \) pops:

\[ m_1 + (m_2 + m_1) = O(m_1 + m_2) \]

Creative Design of Algorithms by Induction

Analogy: Induction ↔ Algorithms

Begin with a problem:
- “Find a solution to problem Q.”

Think of Q as a set containing an infinite number of problem instances.

Example: Sorting
- Q contains all finite sequences of integers.

Solving Q

First step:
- Parameterize problem by size: \( Q(n) \)

Example: Sorting
- \( Q(n) \) contains all sequences of \( n \) integers.

Q is now an infinite sequence of problems:
- \( Q(1), Q(2), ..., Q(n) \)

Algorithm: Solve for an instance in \( Q(n) \) by solving instances in \( Q(i), i < n \) and combining as necessary.

Induction

Goal: Prove that we can solve for an instance in \( Q(n) \) by assuming we can solve instances in \( Q(i), i < n \).

Don’t forget the base cases!

Theorem: \( \forall n \geq 1 \), we can solve instances in \( Q(n) \).
- This theorem embodies the correctness of the algorithm.

Since an induction proof is mechanistic, this should lead directly to an algorithm (recursive or iterative).

Just one (new) catch:
- Different inductive proofs are possible.
- We want the most efficient algorithm!
Interval Containment

Start with a list of non-empty intervals with integer endpoints.

Example:
[6, 9], [5, 7], [0, 3], [4, 8], [6, 10], [7, 8], [0, 5], [1, 3], [6, 8]

---

**Interval Containment (cont)**

Problem: Identify and mark all intervals that are contained in some other interval.

Example:
- Mark [6, 9] since [6, 9] ⊆ [6, 10]

---

**Interval Containment (cont)**

- **Q(n):** Instances of n intervals
- **Base case:** Q(1) is easy.
- **Inductive Hypothesis:** For n > 1, we know how to solve an instance in Q(n - 1).
- **Induction step:** Solve for Q(n).
  - Solve for first n - 1 intervals, applying inductive hypothesis.
  - Check the nth interval against intervals i = 1, 2, · · ·
    - If interval i contains interval n, mark interval n. (stop)
    - If interval n contains interval i, mark interval i.
- **Analysis:**
  \[ T(n) = T(n - 1) + cn \]
  \[ T(n) = \Theta(n^2) \]

---

“Creative” Algorithm

Idea: Choose a special interval as the nth interval.

Choose the nth interval to have rightmost left endpoint, and if there are ties, leftmost right endpoint.
1. No need to check whether nth interval contains other intervals.
2. nth interval should be marked iff the rightmost endpoint of the first n - 1 intervals exceeds or equals the right endpoint of the nth interval.

Solution: Sort as above.
“Creative” Solution Induction

Induction Hypothesis: Can solve for \( Q(n - 1) \) AND interval \( n \) is the “rightmost” interval AND we know \( R \) (the rightmost endpoint encountered so far) for the first \( n - 1 \) segments.

Induction Step: (to solve \( Q(n) \))
- Sort by left endpoints
- Solve for first \( n - 1 \) intervals recursively, remembering \( R \).
- If the rightmost endpoint of \( n \)th interval is \( \leq R \), then mark the \( n \)th interval.
- Else \( R \leftarrow \) right endpoint of \( n \)th interval.

Analysis: \( \Theta(n \log n) + \Theta(n) \).

Lesson: Preprocessing, often sorting, can help sometimes.

Maximal Induced Subgraph

Problem: Given a graph \( G = (V, E) \) and an integer \( k \), find a maximal induced subgraph \( H = (U, F) \) such that all vertices in \( H \) have degree \( \geq k \).
Example: Scientists interacting at a conference. Each one will come only if \( k \) colleagues come, and they know in advance if somebody won’t come.
Example: For \( k = 3 \).

Solution:

Max Induced Subgraph Solution

\( Q(s, k) \): Instances where \(|V| = s \) and \( k \) is a fixed integer.

Theorem: \( \forall s, k > 0 \), we can solve an instance in \( Q(s, k) \).

Analysis: Should be able to implement algorithm in time \( \Theta(|V| + |E|) \).

Celebrity Problem

In a group of \( n \) people, a celebrity is somebody whom everybody knows, but who knows no one else.

Problem: If we can ask questions of the form “does person \( i \) know person \( j \)?” how many questions do we need to find a celebrity, if one exists?

How should we structure the information?
Celebrity Problem (cont)

Formulate as an $n \times n$ boolean matrix $M$.

$$M_{ij} = 1 \text{ iff } i \text{ knows } j.$$ 

Example:

$$\begin{bmatrix}
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

A celebrity has all 0’s in his row and all 1’s in his column.

There can be at most one celebrity.

Clearly, $O(n^2)$ questions suffice. Can we do better?

Efficient Celebrity Algorithm

Appeal to induction:

- If we have an $n \times n$ matrix, how can we reduce it to an $(n - 1) \times (n - 1)$ matrix?

What are ways to select the $n$'th person?

Efficient Celebrity Algorithm (cont)

Eliminate one person if he is a non-celebrity.

- Strike one row and one column.

$$\begin{bmatrix}
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

Does 1 know 3? No. 3 is a non-celebrity.

Does 2 know 5? Yes. 2 is a non-celebrity.

Observation: Each question eliminates one non-celebrity.

Celebrity Algorithm

Algorithm:

- Ask $n - 1$ questions to eliminate $n - 1$ non-celebrities. This leaves one candidate who might be a celebrity.

- Ask $2(n - 1)$ questions to check candidate.

Analysis:

- $\Theta(n)$ questions are asked.

Example:

- Does 1 know 2? No. Eliminate 2
- Does 1 know 3? No. Eliminate 3
- Does 1 know 4? Yes. Eliminate 1
- Does 4 know 5? No. Eliminate 5

4 remains as candidate.