Competitive Analysis

Beyond Worst Case Analysis

Worst-case analysis.
- Analyze running time as function of worst input of a given size.

Average case analysis.
- Analyze average running time over some distribution of inputs.
  - Ex: quicksort.

Amortized analysis.
- Worst-case bound on sequence of operations.
  - Ex: splay trees, union-find.

Competitive analysis.
- Make quantitative statements about online algorithms.
  - Ex: paging, load balancing.

Online Algorithm and Competitive Analysis

Paging problem: Given two-level store consisting of fast memory (cache) that can hold k pages, and slow memory that can store infinitely many pages.

- Sequence of page requests p:
  - if page p already in cache, no cost incurred
  - otherwise, eject some other page q from cache and replace with p, and pay unit cost for page fault.
- If p not in cache, which page q should you evict?
- Most fundamental and practically important online problem in CS.

Competitive analysis. (Sleator-Tarjan)
- Algorithm A is \( r \)-competitive if there exists some constant \( b \) such that for every sequence of inputs \( \sigma \):
  \[
  \text{cost}_A(\sigma) \leq \rho \text{cost}_{OPT}(\sigma) + b.
  \]
  where OPT is optimal offline algorithm.

- OPT = MIN: evict page whose next access is furthest away.

- A = LRU: evict page whose most recent access was earliest
  - Traditional analysis completely uninformative.
  - We show LRU is \( k \)-competitive.
- A = LIFO: evict page brought in most recently.
  - LIFO can have arbitrarily bad competitive ratio.

- Fact: no online paging algorithm is better than \( k \)-competitive.
Online Algorithm and Competitive Analysis

**Theorem.** LRU is \( k \)-competitive.

**Proof:** Let \( \tau \) be a subsequence of \( \sigma \) on which LRU faults exactly \( k \) times, and \( \tau \) does not contain first access in \( \sigma \). Let \( p \) denote page requested just before \( \tau \).

- **Case 1:** LRU faults in sequence \( \tau \) on \( p \).
  - \( \tau \) requests at least \( k+1 \) different pages \( \Rightarrow \) MIN faults at least once
- **Case 2:** LRU faults on some page, say \( q \), at least twice in \( \tau \).
  - \( \tau \) requests at least \( k+1 \) different pages \( \Rightarrow \) MIN faults at least once
- **Case 3:** LRU does not fault on \( p \), nor on any page more than once.
  - \( k \) different pages are accessed and faulted on, none of which is \( p \)
  - \( p \) is in MIN’s cache at start of \( \tau \) \( \Rightarrow \) MIN faults at least once