sequence of requests
- Alg: must handle each request as it comes
  versus off-line - get to look at whole request sequence first.
- Usual scenario for data structures.
- But we will study situations where it makes sense
to compare with "full info." solution, (i.e. best off-line).

Examples
- List accessing - e.g. Move to Front heuristic.
- Paging - LRU, LFU, CS 341, 350
- Splay trees - dynamic optimality conj.
- Bin packing - First Fit, Best Fit.

Ski Rental. - Skiing first time: Rent $30
Buy $300.
In hindsight: if # times you ski is \( \leq 10 \), rent
\( \geq 10 \), buy.

Online alg. (you don't know how many times)
rent 10 times, then buy
Claim: factor \( 2 \) of \( OPT \).
Proof: if # times \( \leq 10 \) - this is \( OPT \)
if \( \cdots \geq 10 \) - you pay \( 2 \times OPT \)

Competitive Analysis - compare online algorithm
with optimal off-line solution (even if \( OPT \) is hard to find)
Alg. A is \( c \)-competitive if \( \exists \) constant \( b \) s.t.
\( A(\sigma) \leq c \cdot OPT(\sigma) + b \)
for minimization.
Note: we allow additive term b always (unlike for approx. algs.)

**Paging**
- fast memory, "cache", holds k pages
- slow memory, n pages \( n \gg k \)

When a page is requested
- if it's in cache, fine.
- otherwise "page fault" must read it into cache
cost 1. Which page do we evict?

Goal: minimize cost

**Optimum off-line strategy**
evict page whose next request is furthest in future.

If - not trivial
modify any opt. soln to this one, bit by bit
without changing cost

**Example**

\[
\begin{array}{ccccc}
A & B & C & B & D \\
E & A & B & E & D \\
\end{array}
\]

\( k = 3 \)

\[
\begin{array}{cccc}
A & B & C & D \\
E & A & D & E \\
B & D & E & E \\
\end{array}
\]

cost 3 evictions

**Online strategies**

- **FIFO** - first in first out
- **LRU** - least recently used
- **LFU** - least frequently used
LRU - least recently used. - like using OPT but pretend future = past.

Ex:

\[
\begin{array}{cccccccc}
A & B & C & B & D & C & E & A & B & E & D \\
\hline
k=3 & A & B & C & D & E & C & D & E & B
\end{array}
\]

- same ex. as previous page.

5 evictions vs. 3 for OPT.

---

Theorem [Sleator, Tarjan 185]
LRU and FIFO have competitive ratio \( k \).

But LRU is better in practice.

Pf. Divide request seq. into phases

\[
k=3 \quad \text{cycle: } \begin{array}{cccccccc}
\end{array}
\]

- phase 1
- phase 2

a phase stops just before we see \( k+1 \) different pages.

The alg. will use \( \leq k \) swaps per phase,

because LRU and FIFO will not evict a page
used in that phase.

And OPT must evict \( \geq 1 \) in each phase + 1 request

because there are \( k+1 \) distinct pages involved.

So \( \text{alg.} / \text{OPT} \leq k \)

Ex. Fill in more detail here.
Thm. Any deterministic alg. has competitive ratio $\geq k$.

*If adversary argument.*

$k=$ cache size  # pages $= k+1$

Adversary always asks for page not in cache.

$n$ swaps  $n =$ length of sequence.

An offline solution that evicts the page with max next request time uses $n/k$ swaps.

Because each time we evict, we're good for next $k$ requests. So $OPT \leq n/k$ and $\text{alg}/OPT \geq k$.

---

A Randomized "Marking" Algorithm

to serve request for page $p$

if $p$ not in cache

if all pages in cache are marked then unmark all

choose a random unmarked page to evict & move $p$ in

mark $p$.

---

Thm. Expected competitive ratio is $O(\log k)$

*Pf.* As before, divide the request sequence into phases in which $k$ different pages are requested.

At the beginning of a phase, all pages in cache are unmarked.

The $k$ requested pages will not be evicted in the phase.

Let $S_i =$ pages in cache at start of phase $i$. 

---
distinguish request for page $p$ as:
- old if $p \in S_i$, new otherwise

Let $n_i =$ # new requests in phase $i$ these all cost 1, (i.e. a page is evicted)
what is the expected cost of old requests? (when we "unluckily" evict a page of $S_i$ and then need it back)

Consider first old request, say for page $p$.
There are $\leq n_i$ new pages $\Rightarrow \leq n_i$ pages were evicted (at random)

$$\text{Prob} \{ p \text{ was evicted} \} \leq \frac{n_i}{k}$$

More generally, let $p_1, p_2, \ldots$ be the distinct old page requests (in order)

When $p_{j+1}$ is requested there were $k-j$ as-yet-unrequested elements of $S_i$.
we have marked $j+$ new pages in the cache
# unmarked pages in cache $\geq k-j-n_i$

$$\text{Prob} \{ p_{j+1} \text{ in cache} \} \geq \frac{k-j-n_i}{k-j}$$

$$\text{Prob} \{ p_{j+1} \text{ not in cache} \} \leq \frac{n_i}{k-j}$$
Sum over \( j = 0 \ldots k-n_i-1 \). Expected cost of old requests
\[
\leq n_i \left( \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{k-(k-n_i-1)} \right)
\leq n_i \left( H_k - 1 \right)
\]

Adding the \( n_i \) cost of new pages: \( n_i \cdot H_k \)

Note \( H_k \) is \( O(\log k) \).

Claim: \( \text{OPT costs} \geq \frac{1}{2} \sum n_i \)

Pf. In phases \( i \) and \( i-1 \) at least \( k + n_i \) distinct pages have been requested (the \( k \) in the cache at the start of phase \( i \) + the \( n_i \) new requests)

\( \ast \ast \) except for \( i=1 \), \( \text{OPT makes} \geq n_1 \) page faults in phase \( i \) and \( i-1 \). \# faults in phase \( i \) \( \leq n_i \)

\( \ast \ast \) total cost \( \geq \frac{1}{2} \sum n_i \), \( \sum n_i \leq 20\text{OPT} \)

Competitive ratio

alg. costs \( H_k \sum n_i \leq 2H_k \text{OPT} = O(\log k) \text{OPT} \)

Fact: There is a (nearly) matching lower bound.

It uses an adversary argument where we assume that the adversary does not know the random coin tosses.

\( \ast \) note that \( n_i \) was defined relative to \( S_i \) (what the algorithm had in the cache) so we cannot just claim that \( \text{OPT} \) must use \( n_i \) faults in phase \( i \).