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.

Competitive Analysis - compare on-line algorithm
with opt 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. alg.s. )
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.

**PF - not trivial**
Modify any opt. soln to this one, bit by bit without changing cost

**Ex.**

\[
\begin{array}{cccccc}
A & B & C & B & D & C & E & A & B & E & D \\
3 & & & & & & & & & & \\
\end{array}
\]

cost 3 evictions

**Optimum**

**Online strategies**

**FIFO** - first in first out

**LRU** - least recently used
- keep track of most recent time each page in cache was used

**LFU** - least frequently used
- keep count of number of times each page in cache was used
LRU - least recently used - like using OPT but pretend future = past.

Ex:

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

\(k=3\)

- Same ex. as previous page.
- 5 evictions
- vs. 3 for OPT.

Theorem [Sleator, Tarjan 1985]

LRU and FIFO have competitive ratio \(k\).

But LRU is better in practice.

Pf: Divide request seq. into phases

\(k=3\)

Cache:

\[
\begin{array}{cccccc}
\end{array}
\]

Assume:

- A phase stops just before we see \(k+1\) different pages.
- The algo \(\leq k\) swaps per phase.

Because LRU and FIFO will not evict a page used in that phase. (Check this out)

Now consider OPT. Let \(p_i\) be first page requested in phase \(i\).

Claim: For requests in phase \(i\) \(\cup p_i \cup p_{i+1}\) \(= p_i\) \(\Rightarrow\) OPT uses \(\geq 1\) swap.

Pf: After request \(p_i\), \(p_i\) will be in cache.

There are \(k+1\) distinct pages in phase \(i \cup p_{i+1}\)

and they don't all fit in the cache.

So \(ALG/OPT \leq k\) + additive term for partial phase at end.
Thm. Any deterministic alg. has competitive ratio $\geq k$.

If adversary argument:

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

Adversary always asks for page not in cache.

$n$ swaps  \quad $n=$ 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 $\text{OPT} \leq n/k$

and $\frac{\text{alg}}{\text{OPT}} \geq k$

\text{Note that there are only $k+1$ pages!}

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 maximal 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 and will be marked by the end of 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

\[ \begin{array}{c}
\text{Start of phase } S_i = \{A, B, C, D\} \\
\text{request } E \text{ new} \\
\text{cost 1} \\
\text{all unmarked}
\end{array} \quad \begin{array}{c}
\text{request } B \text{ old} \\
\text{cost 0} \\
\text{marked}
\end{array} \quad \begin{array}{c}
\text{request } F \text{ new} \\
\text{cost 1} \\
\text{marked}
\end{array} \quad \begin{array}{c}
\text{old} \\
\text{cost 1}
\end{array} \]

Let $n_i = \# \text{ new requests in phase i}$
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 the 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 $p_1, \ldots, p_j$ are in the cache and
are marked. The worst case is that all $n_i$ new
page requests happen before $p_{j+1}$ and cause $n_i$
of the remaining cache slots to be marked.

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

$$\leq n_i \left( \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{k-(k-n_i-1)} \right)$$

$$\leq n_i (H_k - 1) \quad H_k = k^{th} \text{ harmonic number} \quad 1 = O(\log k)$$

Adding the $n_i$ cost of new pages: $n_i \cdot H_k$

To complete the proof we must give a lower bound on OPT

Claim: $OPT \geq \frac{1}{2} \sum n_i$

If in phase $i$ and $i-1$ at least $k+n_i$ distinct pages were requested (the $k$ in the cache at the start of phase $i$ and the $n_i$ new requests)

Thus, except for $i=1$, OPT makes $\geq n_i$ page-faults in phase $i$ and $i-1$ 

Note: $n_i$ was defined relative to $S_i$ (what the Alg. had in the cache) so we cannot just claim that OPT must use $n_i$ faults in phase $i$)

Also $\#$ faults in phase $1$ is $\geq n_i$ (we charge for reading pages into empty cache)

Thus total cost is $\geq \frac{1}{2} \sum n_i$

Thus $\sum n_i \leq 2OPT$

Competitive ratio: alg. costs $H_k \sum n_i \leq 2H_k OPT = O(\log k)OPT$

Fact: There is a (nearly) matching lower bound. It uses an adversary argument where we assume that the adversary does not know the coin tosses.