Online Algorithms

Framework:

σ = Sequence of requests
Define rules for satisfying requests
Define a cost measure

$OPT(\sigma)$ = The minimum cost of any to satisfy σ.

$C_A(\sigma)$ = The cost for algorithm A on σ.

A is <u>c-competitive</u> if

$\forall \sigma : C_A(\sigma) \leq c\, OPT(\sigma)$

Example: Rent or Buy Problem

Renting skis costs $50
Buying skis costs $500

$\sigma = \underbrace{x x x \ldots x}$

$n = |\sigma| =$ #times you go.

What is $OPT(\sigma)$?

ie. what is the optimal strategy if you know n?

---

if $n \geq \frac{500}{50} = 10$ then buy
else rent

Now: Devise an online alg. that is c-competitive for small c.

Answer: Better late than never (BLTN) is 2-competitive.

Theorem: Buying costs b.
Renting costs r.
(b/r = integer)
Then BLTN is

$\left(2 - \frac{r}{b}\right)$ - competitive.

Proof:

Case 1: $n < \frac{b}{r}$

$\Rightarrow OPT(\sigma) = n \cdot r$

$BLTN(\sigma) = n \cdot r$

Compet ratio = 1

Case 2: $n \geq \frac{b}{r}$

$\Rightarrow OPT(\sigma) = b$

$BLTN(\sigma) = \left(\frac{b}{r} - 1\right)r + b$

$= b - r + b = 2b - r$

$\text{Compet ratio} = \frac{2b-r}{b} = 2 - \frac{r}{b}$

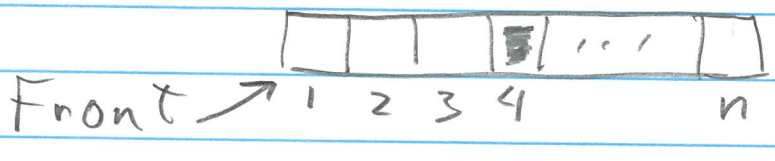
Note: Even if $\frac{b}{r}$ is not an integer BLTN is still 2-competitive

(See lecture notes)

List Update

Groundrules:

- Maintain a list of n distinct items (in a specified initial order.)
- A sequence σ of Access requests to items $\sigma = \sigma_1, \sigma_2, \ldots$
- the cost of Access(σ_i) = the position of item σ_i in the list



- Algorithms can rearrange the list by doing swaps of neighboring items. Cost = 1 per swap.

Challenge: Find a c-competitive online algorithm with small c.

Ideas to try:

- Don't rearrange
- Single Exchange
- Frequency Count
- Move to Front (MTF)

Let's try to evaluate these....

Theorem: MTF is 4-competitive

Proof: We're going to compare MTF to an arbitrary algorithm B.

We'll use amortization on MTF's cost

Φ(MTF List, B's List) =

$2 *$ (# inversions between these lists)

To simplify the analysis we split the state changes into two types:

Access(x): Say the position of x
in MTF's list is k
in B's list is k'

MTF pays k to find x
and k-1 for swaps.

B pays k' to find x.

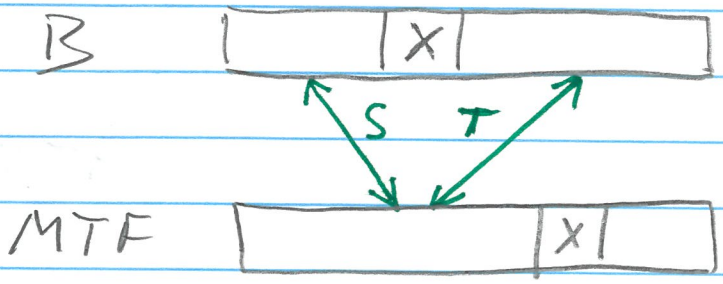
B does a swap: Cost to MTF = 0
but Φ changes.
Cost to B = 1

In both cases we'll show:

Amortized cost to MTF ≤ 4 (B's cost)

ie. $AC_{MTF} \leq 4C_B$

Access(X):



$$C_{MTF} = 2(|S| + |T|) + 1$$

$$\Delta\Phi = -2|T| + 2|S|$$

$$AC_{MTF} = 4|S| + 1$$

$$\leq 4(|S| + 1) \leq 4C_B$$

B does a swap:

$$C_{MTF} = 0$$
$$\Delta\Phi \leq 2$$

$$AC_{MTF} \leq 2 < 4 = 4C_B$$

Total MTF cost $= \sum AC_{MTF} + \underbrace{\Phi_{init}}_{\text{II } 0} - \underbrace{\Phi_{final}}_{\text{VI } 0}$

≤ 4(Total cost to B) ■

Splay Binary Search Trees

Groundrules:

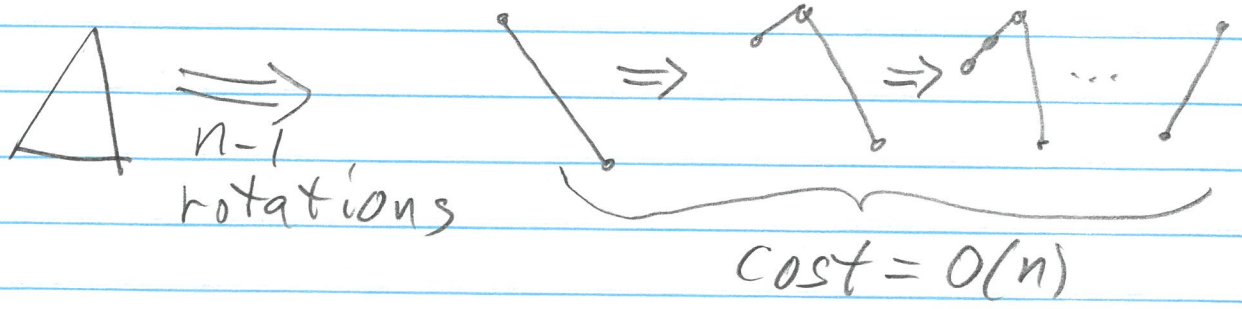
- BST stores keys $\{1, 2 \cdots n\}$
- Sequence of accesses
  Cost = depth in tree
- Do rotations to restructure
  Cost = # rotations

Dynamic Optimality Conjecture:

Splay trees are
c-competitive for some
constant c.

There is a lot of evidence for this.

Eg: Sequential Access



Paging

Ground Rules:

N Pages of memory $1, 2, \cdots, N$
k Pages can be in fast memory (or cache) $k < N$
initially pages $1, 2, 3 \cdots k$ in Cache

σ = Sequence of requests for pages.

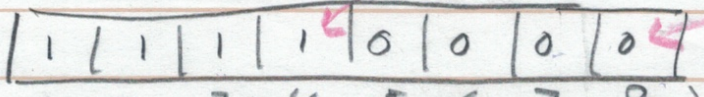
if σ_i is in cache then cost = 0

if σ_i is not, then a page in cache is evicted and σ_i replaces it in cache.
cost = 1. called a page fault

Our goal: a competitive on-line algorithm (with small competitive factor) for the paging problem.

Example with $N = 8, K = 4$

Initial state:



in cache

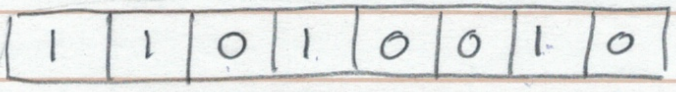
not in cache

$\sigma_1 = 7$

evict 3

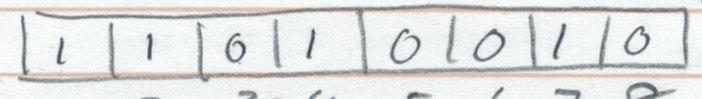
Put 7 in cache

cost = 1



cost = 0

$\sigma_2 = 4$

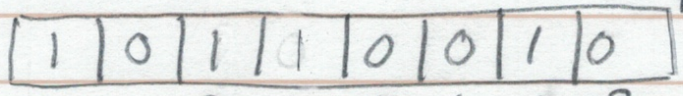


evict 2

Put 3 in cache

cost = 1

$\sigma = 3$

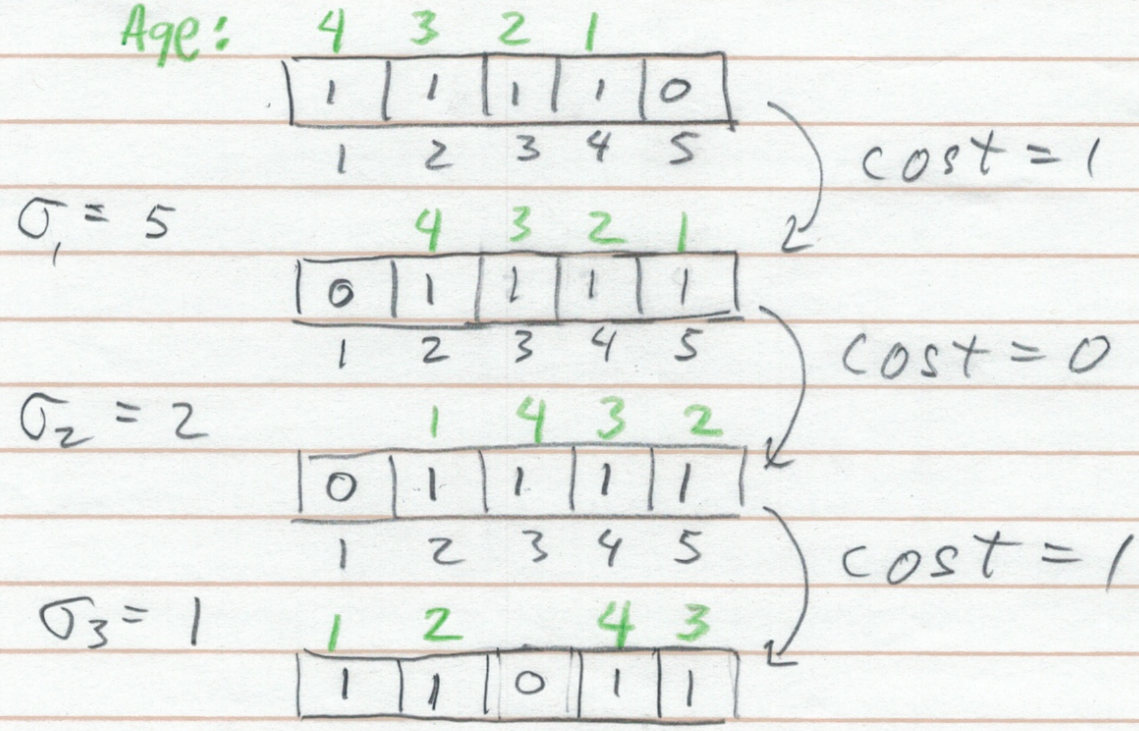


Classic Deterministic Algorithm for Paging: LRU

LRU: Evict the Least Recently Used Page

(Initially 1 is oldest, 2 is second, etc)

Example of LRU with $N=5$, $K=4$



The optimal offline algorithm:
Longest Forward Distance (LFD)

It is: On a page fault, evict the page that is next used later than all others in the cache.

Theorem: LFD is optimal.

No proof given today, but it's not that hard.

What about our goal? A competitive on-line algorithm.

Unfortunately....

theorem: No on-line deterministic algorithm can have a competitive factor $< k$

Proof: We'll prove it for $N = k+1$ (but proof holds for $N > k+1$ too.)

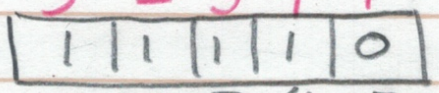
Let A be any on-line algorithm for the paging problem.

let σ = the sequence that causes A to page fault every time. Using only pages $1 \cdots N$.

We will compare this with how LFD performs.

Take this example for $k=4, N=5$

future time: 3 2 5 4 1



As 4 LFD's state

The future times of cache pages are all ≥ 2 and all different.

$\Rightarrow$ there exists one with a future time $\geq$ current time $+k$

e.g. 5 in this case.

So $\sigma_1 = 5$ and LFD evicts page 3. Now the next 3 (i.e. $k-1$) accesses are free for LFD.

Summarize

$\sigma = \sigma_1, \sigma_2 \sigma_3 \cdots \sigma_k \sigma_{k+1}$

σ_1: Page fault for LFD

$\sigma_2 \cdots \sigma_k$: No LFD page faults

σ_{k+1}: Possible next LFD page fault.

$\Rightarrow$ for every k accesses LFD has ≤ 1 fault while A has k. ∎

Randomization to the Rescue!

Marking Algorithm:

- Initial state: $\{1, \ldots, k\}$ in cache
  pages $1 \ldots k$ marked

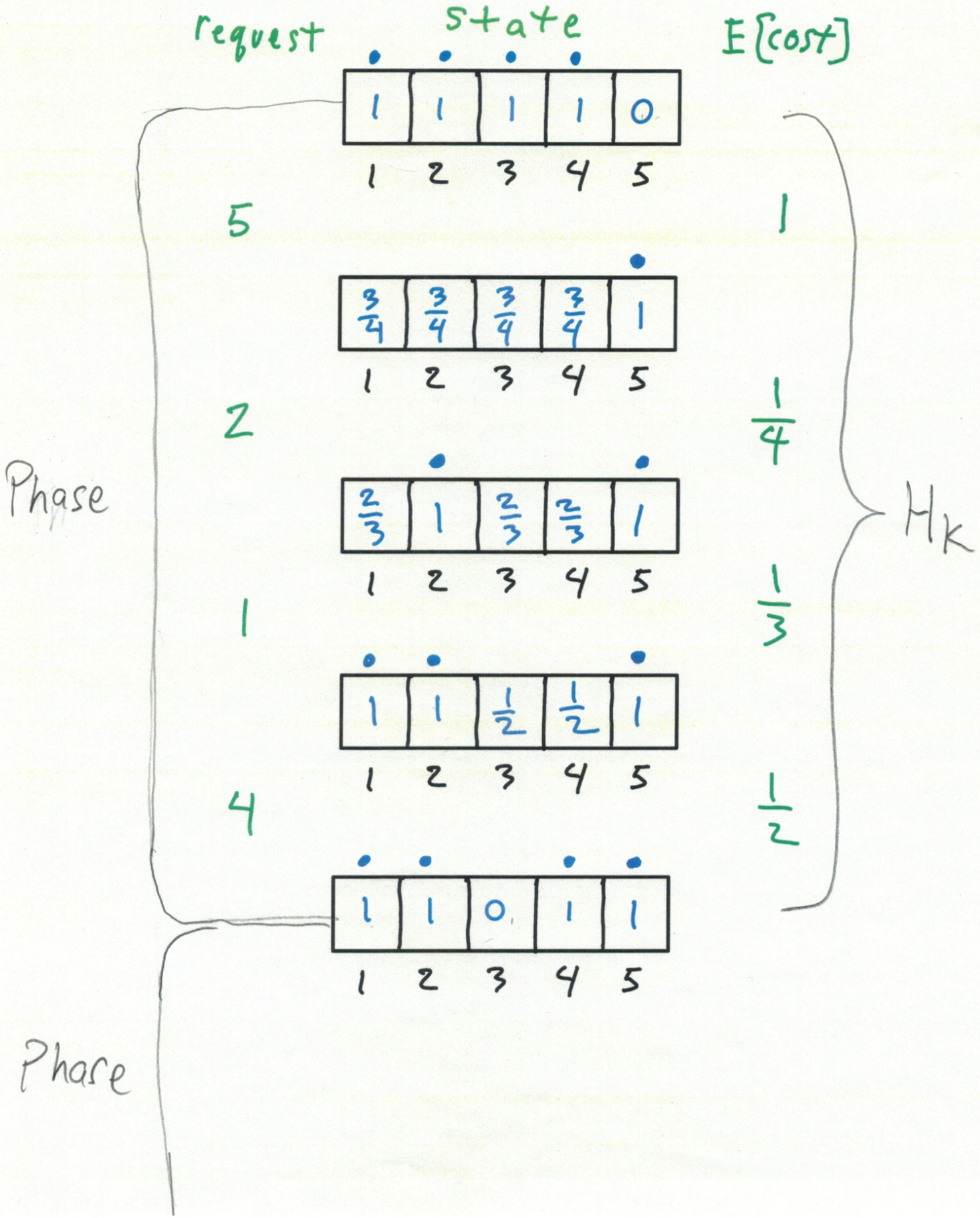
- When a page is requested:

  If in cache, mark it.
  else:

  ① If all pages in cache are marked then unmark all

  ② Pick a random unmarked page in cache and evict. Put the request in cache, mark it.

[Example from lecture notes]


request
state
E[cost]
1 1 1 1 0
1 2 3 4 5
5
1
3/4 3/4 3/4 3/4 1
1 2 3 4 5
2
1/4
Phase
2/3 1 2/3 2/3 1
1 2 3 4 5
1
1/3
1 1 1/2 1/2 1
1 2 3 4 5
4
1/2
1 1 0 1 1
1 2 3 4 5
Hk
Phase

Properties of Marking:

- The marked pages are exactly those with probability 1
- The state at the start of a phase is deterministic (Probs are all 0 or 1)
- During a phase k distinct pages* are requested, including the one not in the cache at the start of phase
- The state at the start and end of a phase differ.

* Ignoring requests to pages that are probability 1.

Theorem:

$$\forall \sigma \quad \frac{E[\text{Marking}(\sigma)]}{OPT(\sigma)} \leq \begin{cases} H_k & \text{if } N = k+1 \\ 2H_k & \text{if } N > k+1 \end{cases}$$

$$\left(H_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k} \leq 1 + \ln k \right)$$

Proof (Just $N = k+1$ case):

The expected cost of a phase for Marking is H_k

Note we can ignore all requests to marked pages.

(This does not affect Marking cost, and may improve OPT.)

$\Rightarrow$ WLOG the first request in a phase is to the unmarked page.

Lemma: If m phases begin, then the cost to OPT is at least m.

Proof of Lemma:

If OPT and Marking start and end a phase in the same state, then cost to OPT ≥ 1.

Because: the first request costs OPT 1.

In general:

	start phase	end phase	cost to OPT $\geq$
✓	$=$	$=$	1
✓	$=$	$\neq$	2
✓	$\neq$	$=$	0
	$\neq$	$\neq$	1

Verify all cases.

the result follows from this table.