

# Online Algorithms

## Framework:

$\sigma$  = Sequence of requests

Define rules for satisfying requests

Define a cost measure

$\text{OPT}(\sigma)$  = The minimum cost of any to satisfy  $\sigma$ .

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

$A$  is  $c$ -competitive if

$$\forall \sigma : C_A(\sigma) \leq c \text{OPT}(\sigma)$$

## Example: Rent or Buy Problem ②

Renting skis costs \$50

Buying skis costs \$500

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

$$n = |\sigma| = \text{\#times you go.}$$

What is  $\text{OPT}(\sigma)$ ?

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  $r$ .  
 ( $b/r = \text{integer}$ )  
 Then BLTN is

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

Proof:

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

$$\Rightarrow \text{OPT}(\sigma) = n \cdot r$$

$$\text{BLTN}(\sigma) = n \cdot r$$

$$\text{Compet ratio} = 1$$

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

$$\Rightarrow \text{OPT}(\sigma) = b$$

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

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

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





(4)

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

(See lecture notes)

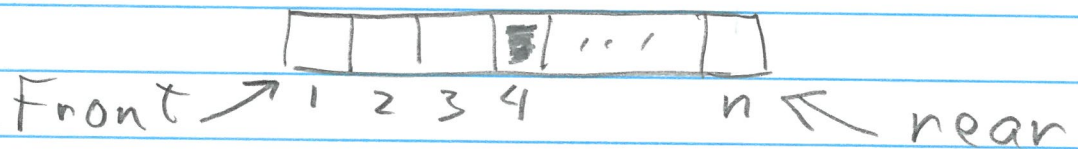


# List Update

5

## Groundrules:

- Maintain a list of  $n$  distinct items (in a specified initial order.)
- A sequence  $\sigma$  of Access requests to items  $\sigma = \sigma_1, \sigma_2, \dots$
- The cost of  $\text{Access}(\sigma_i) =$  the position of item  $\sigma_i$  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

$$\Phi(\text{MTF List}, B's \text{ List}) =$$

$$2 * (\# \text{inversions between these lists})$$

⑦

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  $\Phi$  changes.  
Cost to  $B$  = 1

In both cases we'll show:

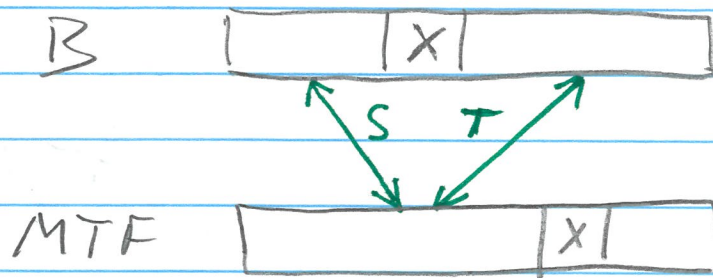
Amortized cost to MTF  $\leq 4(B's \text{ cost})$

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



(8)

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:

$$C_{MTF} = 0$$

$$\Delta \Phi \leq 2$$

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

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

$$\leq 4(\text{Total Cost to } B)$$

# Splay Binary Search Trees

Ground rules:

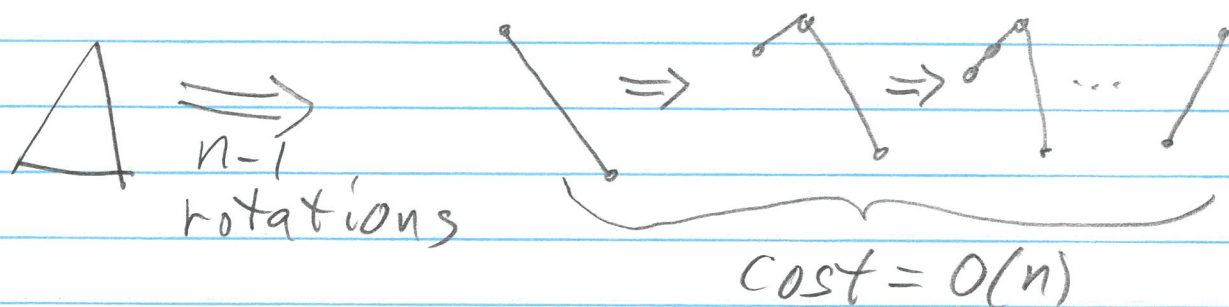
- BST Stores Keys  $\{1, 2, \dots, n\}$
- Sequence of accesses  
Cost = depth in tree
- Do rotations to restructure  
Cost = # rotations

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

$\sigma$  = Sequence of requests  
 for pages.

if  $\sigma_i$  is in cache then cost = 0

if  $\sigma_i$  is not, then a page in  
 cache is evicted and  $\sigma_i$   
 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: 

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |   |

in cache  
not in cache

$\sigma_1 = 7$  evict 3  
Put 7 in cache  
cost = 1

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

cost = 0

$\sigma_2 = 4$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

evict 2  
Put 3 in cache  
cost = 1

$\sigma = 3$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

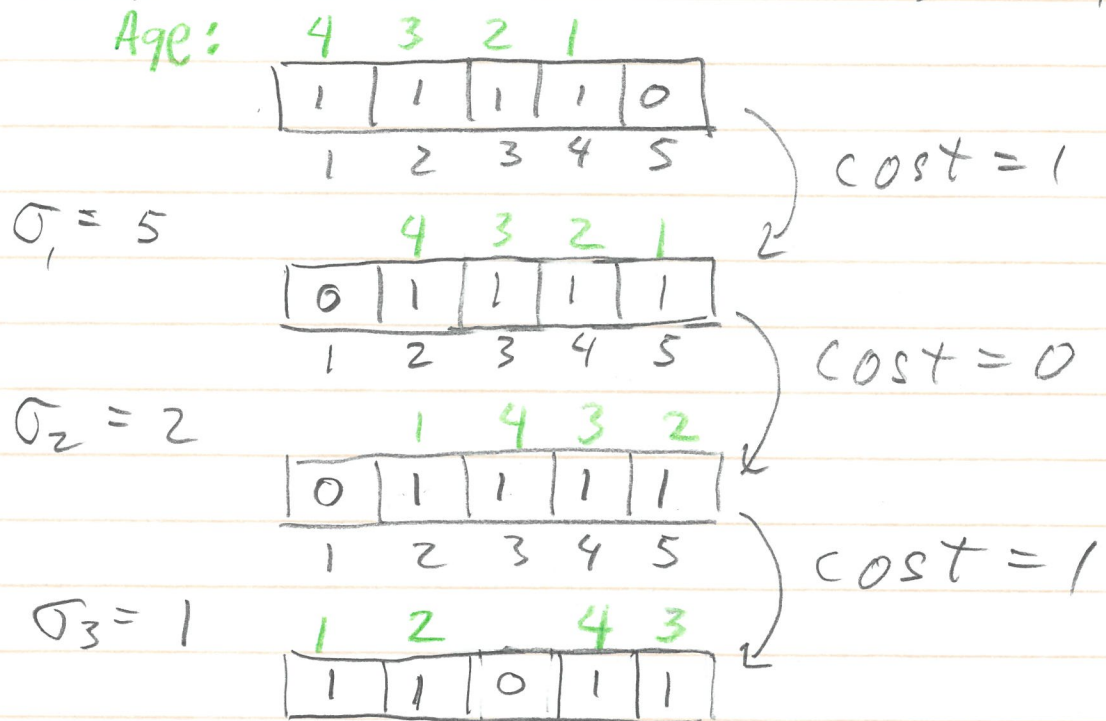
# Classic Deterministic Algorithm for Paging: LRU

LRU: Evict the Least Recently Used Page.

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

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

Age:



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  $N > k+1 + \infty$ .)

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

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

We will compare this with how LFD performs.

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

future time: 3 2 5 4 1

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 1 | 2 | 3 | 4 | 5 |

As LFD's state

The future times of cache pages are all  $\geq 2$  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 (ie  $k-1$ ) accesses are free for LFD.

Summarize

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

$\uparrow$  Page fault for LFD  
 $\underbrace{\hspace{10em}}$  No LFD Page faults  
 $\uparrow$  Possible next LFD Page fault.

$\Rightarrow$  for every  $k$  accesses LFD has  $\leq 1$  fault while A has  $k$ . ■

# Randomization To the Rescue!

## Marking Algorithm:

- Initial State:  $1, \dots, k$  in cache  
pages  $1 \dots k$  marked

- When a page is requested:

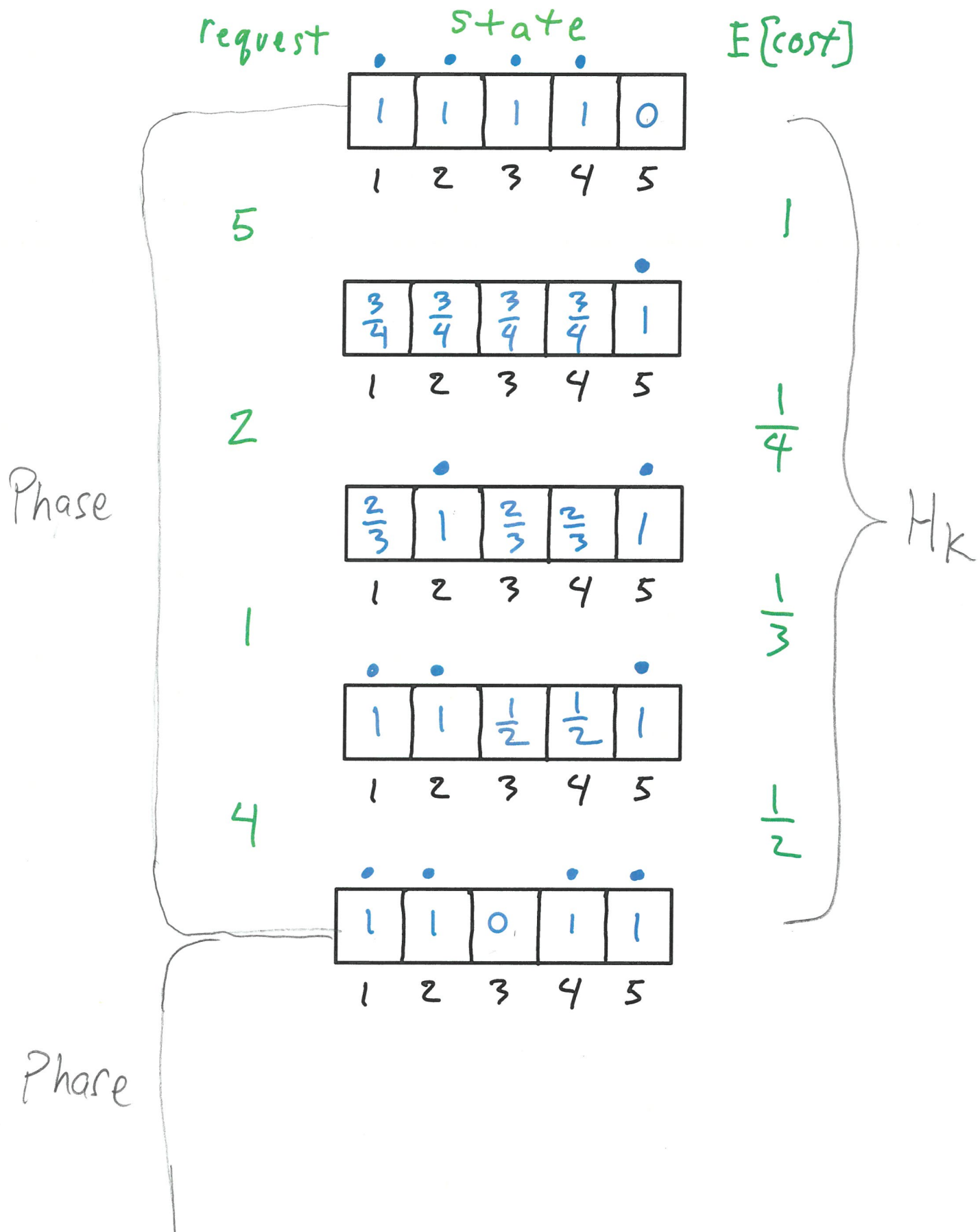
If in cache, mark it.  
else:

- 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]





## 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<sup>\*</sup> 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:

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

$$(H_k = 1 + \frac{1}{2} + \dots + \frac{1}{k} \leq 1 + \ln k)$$

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  $\text{cost to OPT} \geq 1$ .

Because: the first request costs OPT 1.

In general:

|   | Start phase | end phase | Cost to OPT $\geq$ |
|---|-------------|-----------|--------------------|
| ✓ | =           | =         | 1.                 |
| ✓ | =           | ≠         | 2.                 |
| ✓ | ≠           | =         | 0                  |
|   | ≠           | ≠         | 1                  |

Verify all cases.

the result follows from this table

