15451 Spring 2023

Online Algorithms

Elaine Shi

Online Algorithm

“one that can process its input **piece-by-piece in a serial fashion**, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.” --- Wikipedia

Online Algorithm

"one that can process its input **piece-by-piece in a serial fashion**, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start." --- Wikipedia

Offline Algorithm

"given the **whole problem data** from the beginning and is required to output an answer which solves the problem at hand" --- Wikipedia

Competitive Ratio

Competitive analysis

An online algorithm ALG is called c-competitive if for all possible inputs σ

$$ALG(\sigma) \le c \cdot OPT(\sigma)$$

online

best offline alg

Rent or Buy

List Update

Online paging

Rent or Buy

Rent ski: $50

Buy ski: $500

Problem:

- should you rent or buy?
- you don't know how many times you will go skiing in advance

Example

Always buy upfront: $c = \$500/\$50 = 10$

Always rent: $c = \infty$

Example

Rent 5 times and then buy

$n=6.$

$Alg = 50 \times 5 + 500 = 750$

$OPT = 50 \times 6 = 300$

$C = \frac{Alg}{OPT} = 2.5$

Example

Rent 5 times and then buy

$n < 6$:

$n = 6$:

Example

Rent 5 times and then buy

$n < 6$: ALG = OPT = $n * \$50$

$n = 6$: ALG = $5 * \$50 + \$500 = \$750$

OPT = $6 * \$50 = \$300$

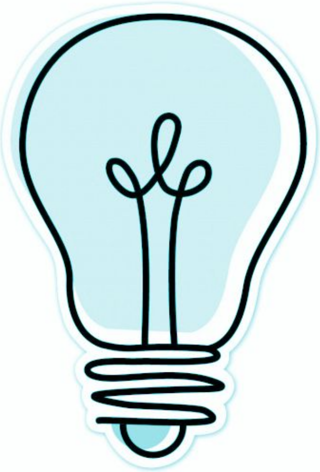
Rent 9 times and then buy?

n < 10: $ALG = OPT = 50 \cdot n$

n >= 10: $ALG = 50 \times 9 + 500 = 950$

$OPT = 500$

$$\frac{ALG}{OPT} = 1.9$$



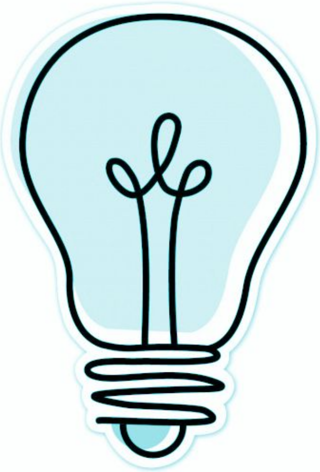
Rent 9 times and then buy?

$n < 10$: OPT = ALG = $\$50 * n$

$n >= 10$: OPT = $500

ALG = $\$50 * 9 + \$500 = \$950$

c = ALG/OPT = 1.9



More generally

Rent ski: r

Buy ski: b

Assume: b/r is integral

BLTN

"Better late than never" algorithm

Rent b/r - 1 times and then buy

Thm: BLTN achieves competitive ratio 2-r/b

$n \cdot r < b$
$r < \frac{b}{n}$

$2 - \frac{r}{b}$

buy on the n-th day $n < b/r$

$ALG = (n-1) \cdot r + b$

$OPT = n \cdot r < b$

$\frac{ALG}{OPT} = \frac{(n-1)r+b}{nr} = \frac{nr - r + b}{nr_{<b}} = 1 - \frac{1}{n} + \frac{b}{nr} > 2 - \frac{r}{b}$

n < b/r: $ALG = OPT = n \cdot r$

n >= b/r: $ALG = \left(\frac{b}{r} - 1\right) \cdot r + b = b - r + b = 2b - r$

$OPT = b$

$\frac{ALG}{OPT} = \frac{2b-r}{b} = 2 - \frac{r}{b}$

Thm: BLTN achieves competitive ratio 2-r/b

n < b/r: OPT = BLTN = r * n

n >= b/r: OPT = b

BLTN = (b/r - 1) * r + b = 2b-r

Thm: BLTN achieves competitive ratio 2-r/b

Thm: BLTN is optimal

Won Nobel for helping found modern portfolio theory: "Efficient frontier"

Harry Markowitz

But, how he invested his own money:
half in bonds and half in stock

List update problem

List of n items, initial ordering fixed

Access(x) pays position of x

Algorithm can rearrange by swapping neighboring elems, each swap costs 1

Example Algorithms

$ALG = n + n \cdots + n + n \cdots$

Never swap

$OPT = n - 1 + n + 1 \cdots + 1 \cdots$

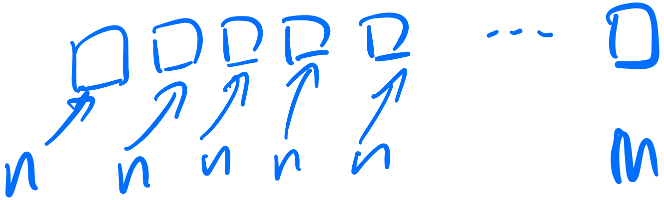
$\frac{ALG}{OPT} = \Omega(n)$

Single exchange: After accessing x, if x is not at the front of the list, swap it with its neighbor toward the front.

$\Omega(n)$

Frequency count: Keep the list ordered by access frequency (large to small)

$\Omega(n)$

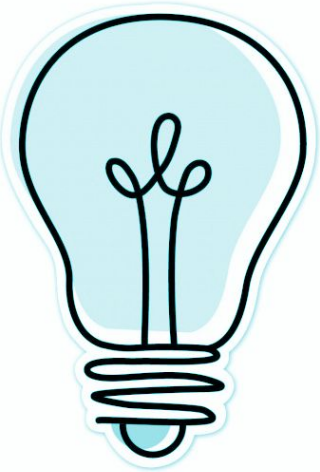


$ALG = n(1+2+3+\cdots n) = \Theta(n^3)$

$MTF \le (2n + (n-1)) \cdot n = \Theta(n^2)$

Move to front algorithm

After accessing x, do a series of swaps to move x to the front



Thm [Sleator-Tarjan'85]:

MTF is 4 competitive

Proof:

Compare MTF and an arbitrary alg B

Potential function

$\Phi(MTF, B) = 2 * \#\text{ inversions}$

$\Phi_{init}(MTF, B) = 0$

$\Phi_{final}(MTF, B) >= 0$

Given any input config and access seq, want to show

$$\text{cost(MTF)} + \underbrace{\Phi_{final}}_{\geq 0} - \underbrace{\Phi_{init}}_{=0} < 4\,\text{cost(B)}$$

$\text{cost(MTF)} < 4\,\text{cost(B)}$

Given any input config and access seq, want to show

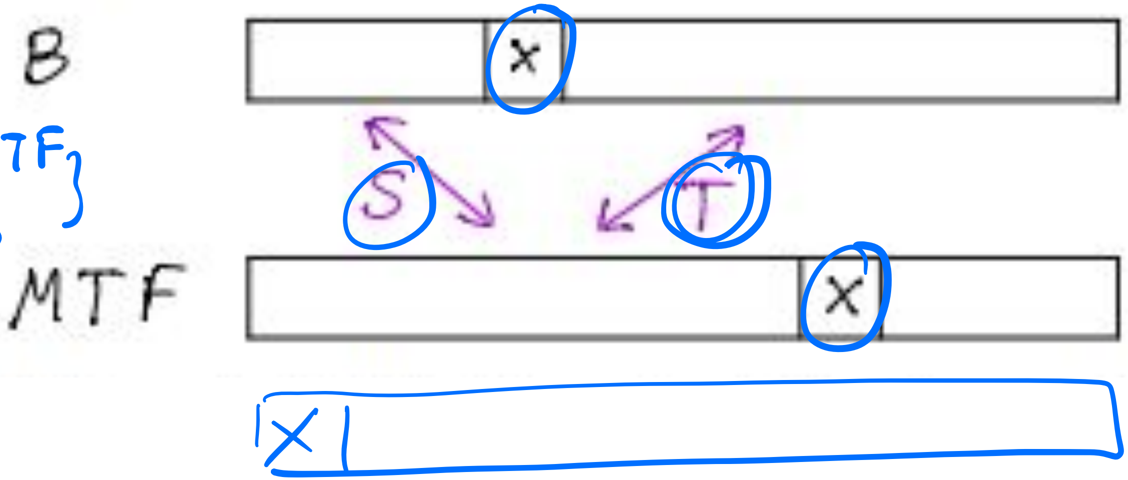
$$\text{cost(MTF)} + \Phi_{\text{final}} - \Phi_{\text{init}} < 4\,\text{cost(B)}$$

Suffices to show

$$\underbrace{\Delta\text{cost(MTF)} + \Delta\Phi}_{\text{Amortize cost per step}} < 4 \cdot \Delta\text{cost(B)}$$

S = { before x in MTF and B }

T = { before x in MTF, after x in B }



Access(x)

Consider some step, suppose B performs y swaps

$$\Delta\text{cost(MTF)} = \underbrace{|S|+|T|+1}_{\text{find}} + \underbrace{|S|+|T|}_{\text{swap}} = 2(|S|+|T|)+1$$

Amortize cost $\le 4\cdot(|S|+1)+2y$

$$\Delta\Phi \le 2\underbrace{(|S|-|T|)}_{\text{move x to front by MTF}} + \underbrace{2y}_{\text{any swap by B}}$$

$$\Delta\text{cost(B)} \ge \underbrace{|S|+1}_{\text{find}} + y$$

$$\frac{4|S|+1+2y}{|S|+1+y} \le 4$$

Online paging

N slow RAM pages, cache has k < N pages

On access page:

- if not in cache, fetch from memory, put it in cache
- need a cache replacement policy

Used in practice: LRU

Least Recently Used

Theorem: LRU is k-competitive

Used in practice: LRU

Theorem: LRU ~~is~~ does no better than k-competitive

Proof: Can construct an input that causes LRU to have a page fault on every request

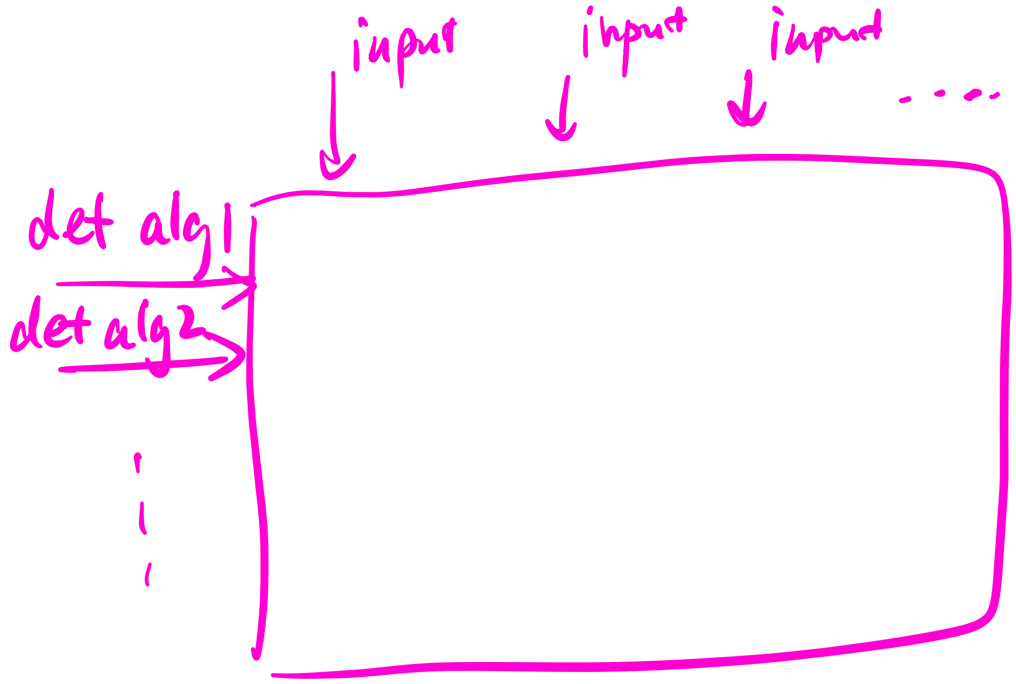
"Farthest in the future"

Theorem: any deterministic algorithm can't have competitive ratio c < k

Proof:

Phase: ~~k distinct pages, 1st page of next phase distinct from all these k~~

Randomized algorand.



We say that online ALG has competitive ratio c iff

$$\forall \sigma \quad E[ALG(\sigma)] \leq c \cdot OPT(\sigma)$$

MARK: a randomized algorithm

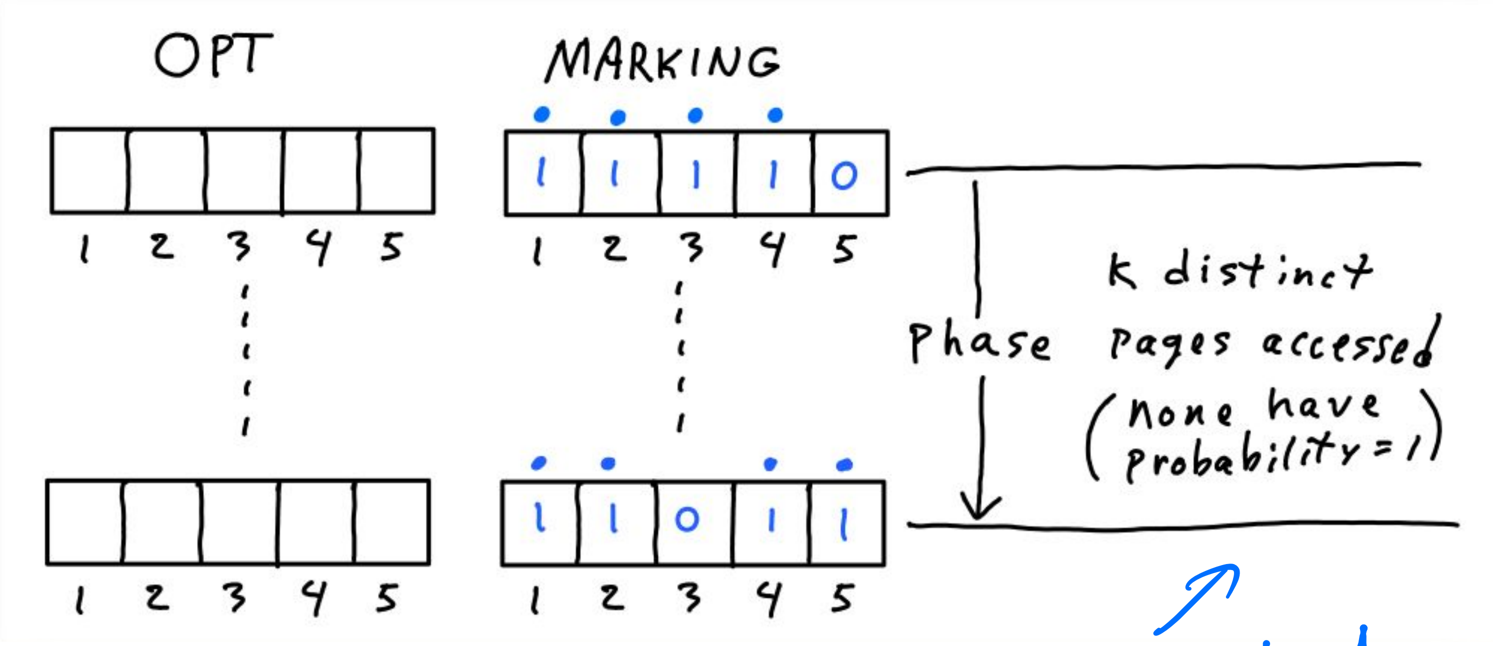
- Initially, pages 1, . . . , k in cache, all marked
- When a page is requested,
  - if in cache, mark it.
  - if not, evict a random unmarked page.
  - if all pages in cache marked, unmark everything first.

Theorem: MARK is $O(\lg k)$-competitive

Proof for the special case $N = k + 1$

$N > k+1$

"Phase"



maximal

next request will be
(k+1)-th distinct page (prob ≠ 1)

$N=5$

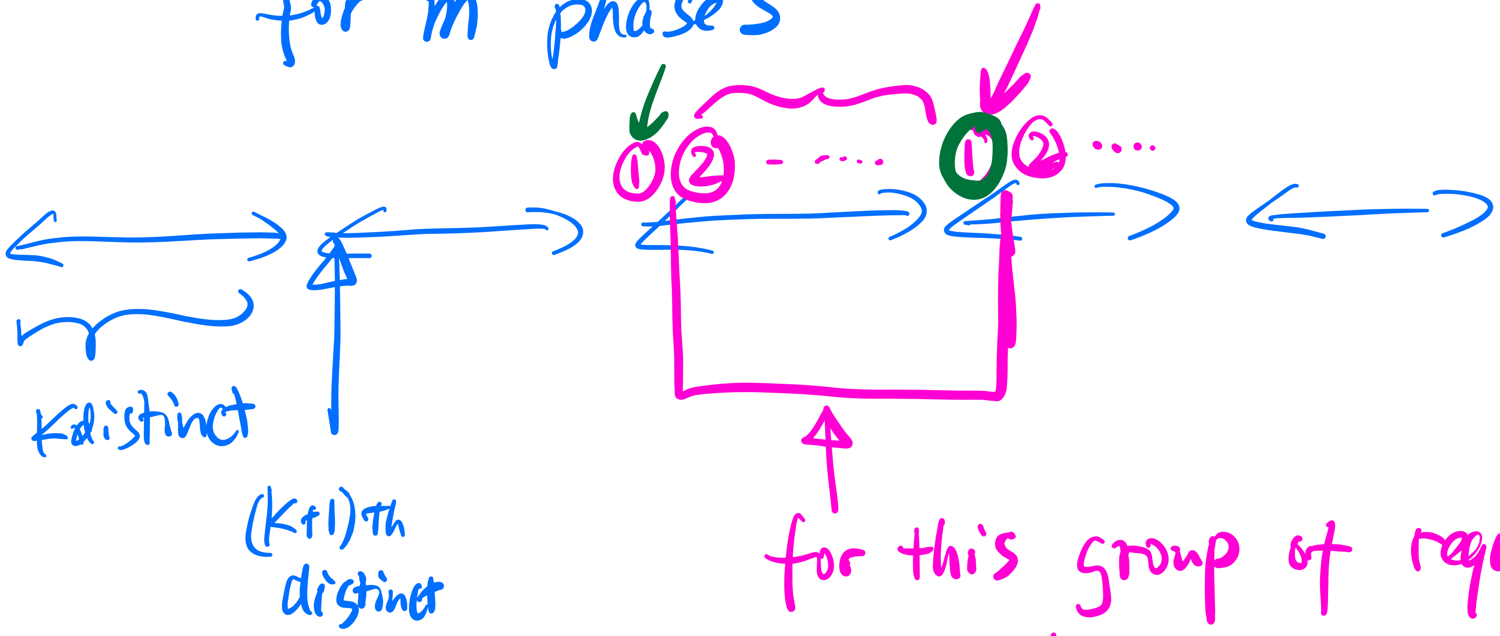
$k=4$

request	1	2	3	4	5	E[cost]
	1	1	1	1	0	
5	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1	1
2	$\frac{2}{3}$	1	$\frac{2}{3}$	$\frac{2}{3}$	1	$\frac{1}{4}$
1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{3}$
4	1	1	0	1	1	$\frac{1}{2}$

$$\underbrace{1+\frac{1}{k}+\frac{1}{k-1}+\dots+\frac{1}{2}}$$

$$=\Theta(\log k)$$

Claim: any algorithm must pay a cost of at least m for m phases



for this group of requests, any alg must pay at least 1 cost.