$T(n) = T(n-1) + T(n-2) + 1 \quad || \quad F_n = F_{n-1} + F_{n-2}$

? HINT

both upper bounded and lower bound

$T(n) \simeq \boxed{F_{n-1} + 2F_n + F_{n+1} - 1}$

ind step

ind hyp

$T(n-1) \simeq \boxed{F_{n-2} + 2F_{n-1} + F_n - 1}$

$T(n) \simeq F_{n-1} + 2F_n + F_{n+1} - 1$

ind

$T(n+1) = T(n) + T(n-1) + 1 \simeq F_n + 2F_{n+1} + F_{n+2} - 2$

$+1$

$= \boxed{F_n + 2F_{n+1} + F_{n+2} - 1}$

Amortized Analysis
Fibonacci Heaps

thanks MIT slides

thanks "Amortized Analysis" by Rebecca Fiebrink

thanks Jay Aslam's notes

Objectives

- Amortized Analysis
  - potential method
- Fibonacci Heaps
  - construction
  - operations

running time analysis

- typical: Algorithm uses data-structure and operations
  - structures: table, array, hash, heap, list, stack
  - operations: insert, delete, search, min, max, push, pop
- measure running time by analyzing
  - the sequence of operations,
  - their frequency
  - each operation running time (computation cost)

Running Time Analysis

- determine the c = costliest/longest iteration
  - usually an outer loop of n iterations
  - overall n* (longest cost per iteration) = n*c
- Thats not very accurate!
  - not all iterations have the longest cost
  - perhaps some average technique can work, but how to prove?
- "compensate" : show that for every costly iteration, there must be other "cheap" iterations

Example : binary counter

log n bits

→ to n

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	cost (bits changed)
0	0	0	0	0	0	N/A
0	0	0	0	0	1	1
0	0	0	0	1	0	2
0	0	0	0	1	1	1
0	0	0	1	0	0	3
0	0	0	1	0	1	1
0	0	0	1	1	0	2
0	0	0	1	1	1	1
0	0	1	0	0	0	4

n ops (+1)

$\boxed{\text{naive}}$ R.T

(worst case)

each op $\leq \log n$

total

$O(n \cdot \log n)$

- each row is a binary representation of the counter
  - increasing by one
  - right side: cost = how many bits require changes

$1+2+1+3+1+2+1+4\ldots$

$= n + n/2 + n/4 + n/8 + \ldots$

$n(1+1/2+1/4+1/8+\ldots)$

- naive running time to increment from 0 to n :
  - each row may cost up to O(log n)
  - n iterations/increments would be O(n*logn)

$\Rightarrow 2n = \Theta(n)$

Example : binary counter

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1
0	0	1	0	0	0

cost (bits changed)
N/A
1
2
1
3
1
2
1
4

→ true cost

- true cost for n iterations: 1+2+1+3+1+2+1+4+... = 2n = O(n)
- reason: the iteration cost very rarely is O(log n)
  - O(logn) means changing a good number of bits
  - for one iteration of cost O(logn), there must be many "cheap" iterations

binary counter amortization

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1
0	0	1	0	0	0

- Aggregation method: consider all n iterations
  - bit 0 changes every iteration => cost n
  - bit 1 changes for half of iterations => cost n/2
  - bit 2 changes quarter of iterations => cost n/4
  - bit 3 changes 1/8 of iterations => cost n/8
  - ... etc
- total cost : add up the cost per bit
  - n + n/2 + n/4+ n/8 + ... = 2n

for pedagogy only.

- Aggregation method impractical, only works on toy examples like this

Amortized Analysis

true cost

amortized / avged

+ little higher

- little lower

- c_i = true cost of i-th operation/iteration
- $\hat{c}_i$ = amortized cost of i-th operation/iteration
  - we have to come up with d_i
- the cumulative amortized cant be smaller than the true cumulative cost, up to any iteration k

$$\forall k : \sum_{i=1:k} c_i \leq \sum_{i=1:k} \hat{c}_i$$

Accounting Method

- assign the di amortized cost
- if overcharge some operation (di>ci) use the excess as "prepaid credit",
- use the prepaid credit later for an expensive operation

Potential method

- associate a potential function ϕ with datastructure T
  - ϕ(Ti) = "potential" (or risk for cost) associated with datastructure after i-th operation
  - typically a measure of complexity/risk/size of the datastructure
- require $\hat{c}_i \geq c_i + \phi(T_i) - \phi(T_{i-1})$ for all i
- $\hat{c}_i$ = amortized cost (up to us to define)
- ci = true cost for operation i
- ϕ = potential function
- Ti = datastructure after ith operation

Accounting Method for binary counter

$\hat{c}_i = c_i + \Delta\phi$

savings $\sum c_i$ $\phi(T_i) - \phi(T_{i-1})$ $\sum_{i=1:k} \hat{c}_i$

Δpotential

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	true cost (c_i)	amortized cost $\hat{c}_i$	
0	0	0	0	0	0	N/A	N/A	
0	0	0	0	0	1	1	2	+1
0	0	0	0	1	0	2	2	0
0	0	0	0	1	1	1	2	+1
0	0	0	1	0	0	3	2	-1
0	0	0	1	0	1	1	2	+1
0	0	0	1	1	0	2	2	0
0	0	0	1	1	1	1	2	+1
0	0	1	0	0	0	4	2	-2

cumulative cost up to k $\leq$ cumul AMORTIZED COST up to k

- assign amortized cost of di=2 for each operation
- verify the amortized condition

$$\boxed{\forall k : \sum_{i=1:k} c_i \leq \sum_{i=1:k} \hat{c}_i}$$

Accounting Method for binary counter

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	true cost (c_i)	amortized cost $\hat{c}_i$	cum true cost $\sum c_i$	cum amortized cost $\sum_{i=1:k} \hat{c}_i$
0	0	0	0	0	0	N/A	N/A	N/A	N/A
0	0	0	0	0	1	1	2	1	2
0	0	0	0	1	0	2	2	3	4
0	0	0	0	1	1	1	2	4	6
0	0	0	1	0	0	3	2	7	8
0	0	0	1	0	1	1	2	8	10
0	0	0	1	1	0	2	2	10	12
0	0	0	1	1	1	1	2	11	14
0	0	1	0	0	0	4	2	15	16

- assign amortized cost of di=2 for each operation Constraint
- verify the amortized condition

$$\forall k : \sum_{i=1:k} c_i \leq \sum_{i=1:k} \hat{c}_i$$

Potential method for binary count

- define the potential $\phi(T_i)$ = the number of "1" bits

T_i = binary counter after i iterations

$i=1:k$

- verify $\boxed{\hat{c}_i \geq c_i + \phi(T_i) - \phi(T_{i-1})}$ for each operation

telescope

$\sum_{i=1:k}$

$$\sum_{i=1}^{k} \hat{c}_i \geq \sum_{i=1}^{k} c_i + \sum_{i=1}^{k} (\phi(T_i) - \phi(T_{i-1}))$$

$$\phi(T_k) - \phi(T_0)$$

≥ 0 $\quad$ $\phi(T_0) = 0$

  - there is only one operation: "increment"
  - di=2 , amortized cost defined by us
  - before the operation i, at T_{i-1}, say there are k trailing 1 ones, before i-th increment
  - ci= true cost = k+1 bit changes: k of "1" bits made "0" (from right to left up to the first "0"); plus the first "0" made "1"
  - $\phi(T_i) - \phi(T_{i-1})$ = "1" gained - "1" lost = 1-k
  - equation becomes 2≥k+1 + 1-k, it checks out! di = 2 is good

...	...	0	1	0	1	1	1	1

k+1

Stack operations - review

- stack is an array with LAST-IN-FIRST-OUT operations
- push(value) : put the new value on the stack (at the top)
  1 elem
- pop(n): take the top n values, return the, delete them from stack
  (or maxstack if $\leq n$)
- naive analysis for n operations : $n*O(n) = O(n^2)$
- better: for pop() to extract many elements, many push() must have happened before, each push is O(1)

	z			
c	c		d	
b	b	b	b	b
a	a	a	a	a
	push(z)	pop(2)	push(d)	pop(1)

Accounting method for Stack

- account each push(x) with \$2:
  - \$1 for the actual push(x) operation, to add x to the stack
  - \$1 credit for the possible later pop() operation that extracts x
- each pop(k) also \$2, for any k
- so each operation is accounted with \$2,
- total running time for n operations is $2*n = O(n)$
- when pop(k) is called, each one of the popped elements have stored \$1 to account for their extraction, $O(k)$ time

Potential method for Stack

- define the potential $\boxed{\phi(\text{stack}) = \text{size}(\text{stack})}$ design for stacks
  - $\phi(T) = |T|$; T = the stack; T_i = stack after i operations
- define the amortized costs: $d_{push}=2$; $\boxed{d_{pop}=2}$
- consider the true costs $c_{push}=1$; $c_{pop(k)}=k$
- prove that for each operation the potential satisfies the fundamental property (exercise)

$$\boxed{\hat{c}_i \geq c_i + \phi(T_i) - \phi(T_{i-1})}$$

universal

- which means the amortized cost d=2 is valid.

push: $2 \geq 1 + \underset{\Delta\phi}{1}$ | pop(k) $\underset{\hat{c}_i}{0} \geq \underset{c_i}{k} \underset{\Delta\phi}{-k}$

Amortized Analysis
Move to Front

Self-organizing lists

- List L of n elements
- The operation ACCESS(x) costs

$$\boxed{rank_L(x)} = \text{distance of } x \text{ from the head of } L.$$

- L can be reordered by swapping adjacent elements at a cost of 1

  choice: move accessed elem to front

- Goal: access to a sequence of n items with minimal cost

List access algorithms

- Off-line Algorithm: if the sequence of access S is known in advance, one can design an optimal algorithm to rearrange the list based on how often items are accessed
- On-line Algorithm: if the sequence is not known in advance, one can design an algorithm based on some heuristics.

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

access

Rank(x)

move to front

Rank(x)

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

$$L \longrightarrow \text{A} \longrightarrow \text{B} \longrightarrow \text{C} \longrightarrow \text{D} \longrightarrow \text{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

$$L \longrightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{C} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

$$L \longrightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{C} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

$$L \longrightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{C} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

$$L \longrightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{C} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{C} \longrightarrow \boxed{D} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \text{A} \longrightarrow \text{B} \longrightarrow \text{C} \longrightarrow \text{D} \longrightarrow \text{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \text{A} \longrightarrow \text{D} \longrightarrow \text{B} \longrightarrow \text{C} \longrightarrow \text{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{A} \longrightarrow \boxed{D} \longrightarrow \boxed{B} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{D} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{D} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

$$L \longrightarrow \boxed{D} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{C} \longrightarrow \boxed{E}$$

- Heuristic: if x is accessed at time t, it is likely to be accessed again soon after time t.
- Cost: MTF always performs within a factor of 4 of the optimal algorithm.

Amortized analysis of MTF

Theorem: $C_{MTF}(S) \leq 4C_{OPT}(S)$
Proof: Let L_i be MTF's list after the ith access, and let L_i^* be OPT's list after the ith access. Let

$$
\begin{aligned}
c_i &= \text{MTF's cost for the } i\text{th operation} \\
&= 2 \cdot rank_{L_{i-1}}(x) \text{ if it accesses } x; \\
c_i^* &= \boxed{\text{OPT's cost for the } i\text{th operation}} \\
&= \underbrace{rank_{L_{i-1}^*}(x)}_{\text{cost}} + \underbrace{t_i}_{\text{after-access swaps}},
\end{aligned}
$$

where t_i is the number of swaps that OPT performs.

Potential function

Inversion $i < j$

$A[i] > A[j]$

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) &= 2 \cdot |\{(x,y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ &= \boxed{2 \cdot \#\text{ inversions}}\end{aligned}$$

between L_i → current MTF list

L_i^* → current OPT list

Example:

after i operations

$$L_i \to \boxed{E} \to \boxed{C} \to \boxed{A} \to \boxed{D} \to \boxed{B}$$

$$L_i^* \to \boxed{C} \to \boxed{A} \to \boxed{B} \to \boxed{D} \to \boxed{E}$$

EC, EA, ED, EB, DB

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$
\begin{aligned}
\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\
=& 2 \cdot \# \text{ inversions}
\end{aligned}
$$

Example:

$$L_i \longrightarrow \boxed{E} \to \boxed{C} \to \boxed{A} \to \boxed{D} \to \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \to \boxed{A} \to \boxed{B} \to \boxed{D} \to \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{\cdots\}|$

Potential function

Define the potential function $\Phi : L_i \rightarrow \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \rightarrow \boxed{C} \rightarrow \boxed{A} \rightarrow \boxed{D} \rightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \rightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{(E, C), \cdots\}|$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{D} \longrightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{E}$$

$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), \cdots\}|$$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{D} \longrightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), \cdots\}|$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x,y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{D} \longrightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{(E,C),(E,A),(E,D),(E,B),\cdots\}|$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{D} \longrightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \longrightarrow \boxed{A} \longrightarrow \boxed{B} \longrightarrow \boxed{D} \longrightarrow \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), (E, B), (D, B)\}|$ $= 10$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Example:

$$L_i \longrightarrow \boxed{E} \rightarrow \boxed{C} \rightarrow \boxed{A} \rightarrow \boxed{D} \rightarrow \boxed{B}$$

$$L_i^* \longrightarrow \boxed{C} \rightarrow \boxed{A} \rightarrow \boxed{B} \rightarrow \boxed{D} \rightarrow \boxed{E}$$

$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), (E, B), (D, B)\}| = 10$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Note that:

- $\boxed{\Phi(L_i) \geq 0}$ for $i = 0, 1, \ldots$
- $\boxed{\Phi(L_0) = 0}$ if MTF and OPT start with the same list.

How much does Φ change from one swap?

- a swap creates/destroys 1 inversion
- $\Delta\Phi = \pm 2$

What happens on access?

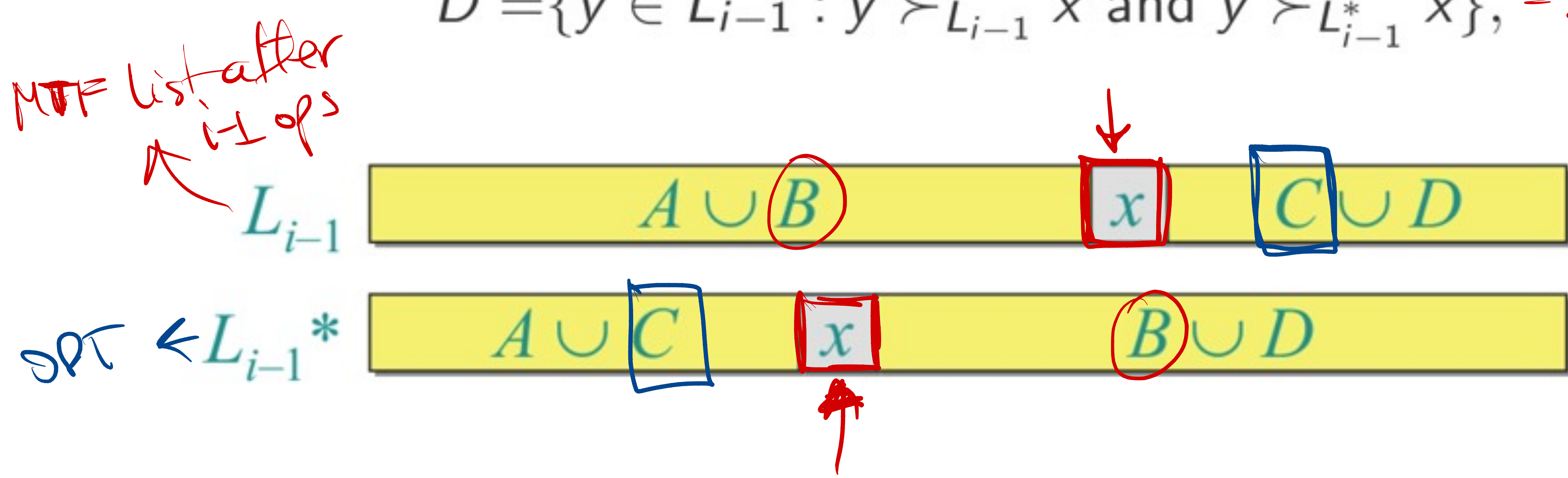
Suppose that operation i access item x, and define

$$A = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \prec_{L^*_{i-1}} x\}, = \{\text{elem in front}(x) \text{ in both lists}\}$$

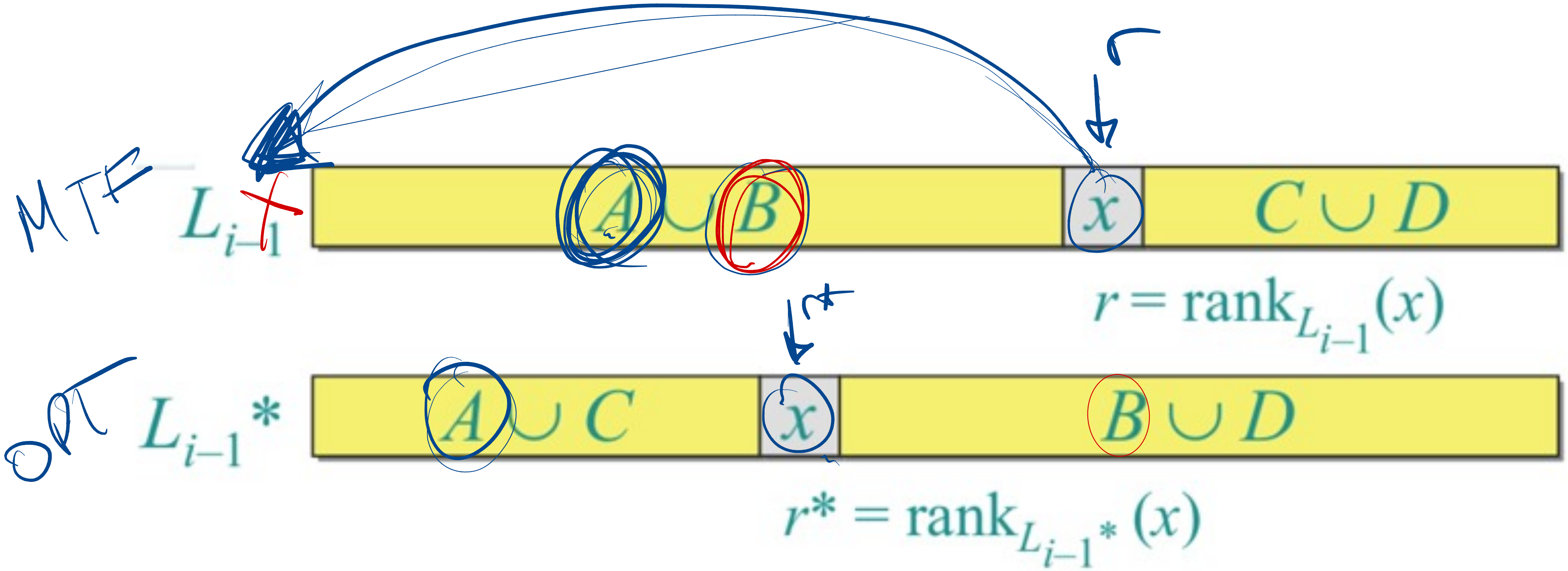
$$B = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \succ_{L^*_{i-1}} x\}, = \{\text{elem infront}(x) \text{ in MTF after}(x) \text{ in OPT}\}$$

$$C = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \prec_{L^*_{i-1}} x\},$$

$$D = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \succ_{L^*_{i-1}} x\}, = \{\text{elem after}(x) \text{ in both lists}\}$$



What happens on access?



We have $r = |A| + |B| + 1$ and $r^* = |A| + |C| + 1$.
When MTF moves x to the front, it creates $|A|$ inversions and destroys $|B|$ inversions. Each swap by OPT creates ≤ 1 inversion.
Thus, we have

$$\boxed{\Phi(L_i) - \Phi_{L_{i-1}}} \leq 2(\underbrace{|A| - |B|}_{\text{MTF}} + \underbrace{t_i}_{\text{OPT}}).$$

→ #swaps OPT does after Access(i)

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\hat{c}_i = c_i + \Phi(L_i) - \Phi(L_{i-1})$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\le 2r + 2\boxed{(|A| - |B| + t_i)}
\end{aligned}
$$

$\to$ upper bound of $\Delta\Phi$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\ &\leq 2r + 2(|A| - |B| + t_i) \\ &= 2r + 2(|A| - (r - 1 - |A|) + t_i)\end{aligned}$$

(since $r = |A| + |B| + 1$)

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\le 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad \text{(since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i
\end{aligned}
$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\le 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= \cancel{2r} + 4|A| - \cancel{2r} + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i
\end{aligned}
$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i \\
&\leq 4(r^* + t_i) \\
&\quad (\text{ since } r^* = |A| + |C| + 1 \geq \boxed{|A| + 1})
\end{aligned}
$$

$\leq 4(\underbrace{|A|+1}_{\leq r^*}) + 4t_i$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\le 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i \\
&\le 4(r^* + t_i) \\
&\quad (\text{ since } r^* = |A| + |C| + 1 \ge |A| + 1) \\
&= 4c_i^*
\end{aligned}
$$

ACC SWAPS

$c^* = r^* + t_i$

WANT $\underbrace{\sum c_i}_{\text{MTF}} \le \underbrace{\sum \hat{c}_i}_{\text{MTF-Amortized}} \le 4 \underbrace{\sum c_i^*}_{\text{OPT}}$

The grand finale

Thus, we have

$$C_{MTF}(S) = \sum_{i=1}^{|S|} c_i$$

The grand finale

Thus, we have

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i))
\end{aligned}
$$

The grand finale

Thus, we have

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i)) \\
&\leq (\sum_{i=1}^{|S|} 4c_i^*) + \Phi(L_0) - \Phi(L_{|S|})
\end{aligned}
$$

The grand finale

Thus, we have

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i)) \\
&\leq (\sum_{i=1}^{|S|} 4c_i^*) + \Phi(L_0) - \Phi(L_{|S|}) \\
&\leq 4C_{OPT}(s)
\end{aligned}
$$

(since $\Phi(L_0) = 0$ and $\Phi(L_{|S|}) \geq 0$)