ADVANCED ALGORITHMS

TOPICS:

01. INTRODUCTION
02. ALGORITHM AS RECIPE
03. DIVIDE AND CONQUER
04. PRAM
05. RANDOMIZATION
06. MINCUT
07. SORTING
08. SELECT i^{TH}
09. PRIMALITY TEST
10. RANDOM DATA STRUCTURES
11. DYNAMIC PROGRAMMING
12. DYNAMIC PROGRAMMING BDD
13. AMORTIZED ANALYSIS
14. COMPETITIVE ANALYSIS

02. ALGORITHM AS RECIPE

ALGORITHM = COMPUTATIONAL PROCEDURE THAT TAKES AN INPUT AND PRODUCES AN OUTPUT IN A FINITE NUMBER OF STEPS

SORTING: $\langle a_1, a_2 \dots, a_n \rangle \longrightarrow$ permutation $\langle a_1', a_2' \dots, a_n' \rangle \mid a_1' \le a_2' \le \dots \le a_n'$

INSERTION SORT

```
InsertionSort (int[] A, int n)
for i←2 to n do
    val ← A[i]
    j←i-1
    while j>0 and A[j]>val do
        A[j+1]← A[j]
        j ← j-1
    A[j+1]← val
```

NOTE: PSEUDOCODE IS 1-BASED!

- WORST CASE (A REVERSE-SORTED): $T(n) = \sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
- AVERAGE CASE: $T(n) = \sum_{i=1}^{n} \Theta(i/2) = \Theta(n^2)$
- BEST CASE (ALREADY SORTED): $T(n) = \Theta(n)$
- SPATIAL: $\Theta(1)$ INPLACE

PERFORMANCE ANALYSES

NOTATIONS: $\Theta, O, \Omega, o, \omega$

MERGE SORT

```
MergeSort (int[] A, int first, int last)
if first<last then
    int mid=⌊(first+last)/2⌋
    MergeSort (A, first, mid)
    MergeSort (A, mid+1, last)
    Merge (A, first, last, mid)
```

- $T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ 2T(n/2) + \Theta(n) & \text{if } n>1 \end{cases}$
- $T(n) = \Theta(n \log n)$
- SPATIAL: $\Theta(n)$ FOR MERGING
- better than INSERTION SORT for $n>30$

SOLVE RECURRENCES ⤳ BOTH TEMPORAL AND SPATIAL!

- RECURSION TREE
- SUBSTITUTION : GUESS, VERIFY BY INDUCTION, SOLVE CONSTANTS
- MASTER THEOREM · $T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n>1 \\ d & \text{otherwise} \end{cases}$ $a \geq 1, b > 1, f(n)$ asymptotically positive
  - $\exists \varepsilon > 0 : f(n) = O(n^{\log_b a - \varepsilon}) \longrightarrow T(n) = \Theta(n^{\log_b a})$
  - $\exists k \geq 0 : f(n) = \Theta(n^{\log_b a} \lg^k n) \longrightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
  - $\exists \varepsilon > 0 : f(n) = \Omega(n^{\log_b a + \varepsilon}),\ \longrightarrow T(n) = \Theta(f(n))$
    $\exists c \in (0,1) : af(n/b) \leq cf(n)$ ⤳ REGULARITY CONDITION

03. DIVIDE AND CONQUER

DIVIDE, CONQUER, COMBINE

- MERGE SORT $T(n) = 2T(n/2) + \Theta(n) \rightsquigarrow \Theta(n \log n)$
- BINARY SEARCH $T(n) = T(n/2) + \Theta(1) \rightsquigarrow \Theta(\log n)$
- POWERING A NUMBER $T(n) = \Theta(n)$ NAIVE : $a \cdot a \cdot a \cdots a$
  $T(n) = T(n/2) + \Theta(1) \rightsquigarrow \Theta(\log n)$ Div. et Imp: $a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ even} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ odd} \end{cases}$
- MATRIX MULTIPLICATION
  - NAIVE: $T(n) = \Theta(n^3)$
  - DIVIDE ET IMPERA: $T(n) = 8T(n/2) + \Theta(n^2) \rightsquigarrow \Theta(n^3)$
  - STRASSEN: $T(n) = 7T(n/2) + \Theta(n^2) \rightsquigarrow \Theta(n^{\lg 7}) \sim \Theta(n^{2.81})$
- VLSI LAYOUT $\rightarrow$ AREA $= \Theta(n \log n)$
- H-TREE EMBEDDING $L(n) = 2L(n/4) + \Theta(1) \rightsquigarrow \Theta(\sqrt{n})$
  AREA $= \Theta(n)$

04. PRAM

RAM MACHINE MODEL TO DESIGN ALGO IN A COMPLETE AND COHERENT WAY

- UNBOUNDED NUMBER OF LOCAL CELLS
- UNBOUNDED CELL DIMENSION
- INSTRUCTION SET

  ALL OP. TAKE THE SAME TIME

COMPLEXITIES:

- TIME: # INSTR EXECUTED

  SPACE: # MEM CELLS USED

PRAM ABSTRACTION OF MACHINE FOR DESIGNING ALGO FOR PARALLEL COMPUTING (∞: IN, OUT, SH, PROCESSORS)

- UNBOUNDED COLLECTION OF RAM PROCESSORS SHARED MEM CELLS FOR COMMUNICATION
- ALL OP. EXECUTED SIMULTANEOUSLY BY ALL PROCESSORS
- 5 COMPUTATION STEPS (EACH PROC) RD VAL FROM CELLS. RD 1 SH MEM CELL. COMPUTATION. [WT 1 OUT CELL]. [WT 1 SH MEM]
- TO AVOID CONFLICTS
  - EXCLUSIVE READ

    EXCLUSIVE WRITE
  - CONCURRENT READ
  - CONCURRENT WRITE

    ↳ WHAT? PRIORITY / RANDOM / COMMON (ALL IN)

    ← EFFICIENCY

  → PRAM CLASSIFICATION: EREW, CREW, CRCW →

- COMPUTATIONALLY STRONGER THAN MODEL B, iff ANY ALGO FOR B RUNS UNCHANGED IN SAME PARALLEL TIME, SAME BASIC PROP.
- DEFINITIONS: $SU_p = \dfrac{T^*(m)}{T_p(m)}$ $\quad E_p = \dfrac{T_1(m)}{p\,T_p(m)}$ $\quad COST = P(m) \cdot T(m)$

MATRIX-VECTOR MULTIPLY $y = Ax$ [CREW]

1. CONCURR. READ OF x
2. SIMULTANEOUS READS OF DIFFERENT PARTS OF A
3. COMPUTE RESULT
4. SIMULTANEOUS WRITES (NO CONFLICT)

PERFORMANCE:

- $T_1 = O(m^2)$ → $SU_p = p$ → PERFECTLY PARALLEL!
- $T_p = O(m^2/p)$ → $COST = m^2$

→ $E_p = \dfrac{m^2}{p \cdot \frac{m^2}{p}} = 1$ → PERFECT EFFICIENCY!

LOGARITMIC SUM VECTOR

- INTERNAL NODE: SUM OPERATION
- if $P = n$
  - $T^* = T_1(n) = n$ $SU_p = n/\log n + 2$
  - $T_p = \log n + 2$ $E_p = \frac{n}{\underset{n}{\overset{\shortparallel}{p}} \log n} = \frac{1}{\log n}$
- if $n \gg p$
  - $T^* = T_1 = n$ $SU_p \approx p$
  - $T_p = \frac{n}{p} + \log n$ $E_p \approx 1$

MATRIX MULTIPLY $AB = C$ [CREW]

- n^3 PROCESSORS n FOR SUM, n^2 TOT SUMS

1. CONCURRENT READ (RAW A, COL B)
2. MULTIPLY AND SUM (LOG, n proc. for n el.)
3. EXCLUSIVE WRITE

- $T_1 = n^3$ $SU = n^3/\log n$
- $T_{P=n^3} = \log n$ $E_p = \frac{1}{\log n}$

PRAM VARIANTS

- BOUNDED # P $\forall p': p' < p$, PROBLEM SOLVED BY P PROC IN T STEPS => CAN BE SOLVED IN $T' = O(T \cdot P/P')$ STEPS
- BOUNDED # MEM CELLS $\forall M': M' < M$ $T' = O(T \cdot M/M')$ STEPS

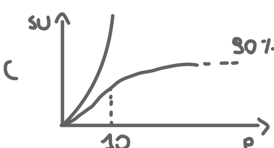
PREFIX SUM ON CREW

example: COMMON CRCW (DNF)

AMDAHL'S LAW [STRONG SCALING] TO CALCULATE SPEEDUP BY PARALLELIZATION

- SERIAL SEGMENTS
- PARALLEL SEGMENTS

$SU(P, f) = \frac{1}{1 - f + \frac{f}{P}}$ $\lim_{P \to +\infty} SU(P, f) = \frac{1}{1-f}$ PESSIMISTIC

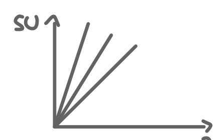


→ USE NO MORE THAN 10p

GUSTAFSON'S LAW [WEAK SCALING]

$SU = s + P(1 - s)$

↳ SERIAL TIME



LINEAR

05. RANDOMIZATION

RANDOMIZED ALGORITHM

HIRING PROBLEM
- WORST CASE: m
- AVERAGE CASE: $\log m$ → INSTEAD OF RELYING ON AN INPUT DISTRIBUTION ASSUMPTION, MAKE YOUR OWN INPUT DISTRIBUTION, PERMUTING INPUT RANDOMLY
- BEST CASE: 1

MONTE CARLO: SOMETIMES INCORRECT SOLUTION — KARGER, PRIMALITY

FOR DECISION PROBLEMS:
- TWO-SIDED = NON-ZERO PROB. IT ERRS YES/NO
- ONE-SIDED = AT LEAST ONE OUTPUT (YES/NO) WITH ZERO PROB. IT ERRS

EFFICIENT IF, FOR ANY INPUT, THE <u>WORST</u> CASE RUNNING TIME BOUNDED BY POLY(m) (INPUT SIZE)

LAS VEGAS: ALWAYS CORRECT SOLUTION = MONTE CARLO WITH ZERO ERROR PROBABILITY — SELECT i^{th}, QUICKSORT

EFFICIENT IF, FOR ANY INPUT, <u>EXPECTED</u> RUNNING TIME BOUNDED BY POLY(m)

example: FIND CHARACTER IN A STRING

06. MIN CUT

CUT

MIN st CUT ⟷ MAX FLOW

KARGER: RANDOMIZED MONTECARLO APPROACH (ON MULTIGRAPH, NO SELF LOOPS)
- CONTRACT (u,v) PRESERVES CUTS WHERE $u,v \in S_1$ (S_2)
- $m-2$ EDGE CONTRACTIONS — ONE RUN TAKES $O(m^2)$ TIME
- $Pr(\text{found min cut}) \geq 1/\binom{m}{2}$ — REPEAT $\ell\binom{m}{2}$
- $\ell = c \ln m \Rightarrow Pr(\text{ERROR}) \leq \frac{1}{m^c} = \frac{1}{\text{POLY}(m)}$ ⇒ TIME: $O(m^4 \log m)$

KARGER-STEIN

- PROBABILITY TO CONTRACT MIN-CUT NODES INCREASES
- $t = n/\sqrt{2}$ => $\Pr$(MIN CUT SURVIVES) $\geq 1/2$ => TWO RUNS ENOUGH
- AT MOST n^2 MIN CUT
- if $n > 6$, REPEAT TWICE:
  - CONTRACT TO $n/\sqrt{2} + 1$ NODES (ORIGINAL ALGO)
  - RECURSE

  RETURN MIN CUT AMONG THE TWO

  $T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + \Theta(n^2) = \Theta(n^2 \log n)$
- $O(\log^2 n)$ RUNS => $\Pr$(ALGO ERROR) $\geq \frac{1}{\text{POLY}(n)}$

  $\Rightarrow O(n^2 \log^3 n)$

```
contract(G=(V,E), t)
while |V| > t do
    e = randomly chosen in G
    G = G/e
return G
```

```
fastMinCut(G=(V,E))
if |V| ≤ 6 then
    return contract(G, 2)
t = ⌈ |V|/√2 + 1 ⌉
G1 = contract(G, t)
G2 = contract(G, t)
return min(fastMinCut(G1), fastMinCut(G2))
```

07. SORTING

QUICKSORT

Partition (A, p, q) $\Theta(n)$

```
x ← A[p]
i ← p
for j ← p+1 to q do
    if A[j] < x then
        i ← i+1
        A[i] ↔ A[j]      // ↔ = EXCHANGE
A[p] ↔ A[i]
return i
```

QuickSort (A, p, r)

```
if p < r then
    q ← Partition(A, p, r)
    QuickSort(A, p, q-1)
    QuickSort(A, q+1, r)
```

- WORST CASE:
  $T(n) = T(0) + T(n-1) + \Theta(n) = \Theta(n^2)$
- BEST CASE:
  $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$

COST ANALYSIS

WOST CASE RARE, BEST CASE ALSO WITH 1:9 ARRAY SPLIT

- EVEN IF ALTERNATE LUCKY-UNLUCKY, $\Theta(n \log n)$ $\quad L(n) = 2U(n/2) + \Theta(n)$
  $U(n) = L(n-1) + \Theta(n)$
- <u>RANDOMIZATION</u>: ON CHOICE OF PIVOT => WORST CASE only depends on output of random generation LAS VEGAS
- ◡̈ TWICE AS FAST AS MERGE SORT
- ◡̈ GOOD WITH CACHING, VIRTUAL MEM

SORTING BOUNDS

COMPARISON SORT: $\Omega(n \log n)$ ⇝ DECISION TREE with $h = \Omega(n \log n)$, $n!$ leaves

=> HEAPSORT, MERGESORT are ASYMPTOTICALLY OPTIMAL SORTING ALGORITHMS

- COUNTING SORT

$$\text{CountingSort}(A, \overset{\text{length}}{m}, \overset{\text{max val}}{k})$$

```
for i ← 1 to k do
    C[i] ← 0
for i ← 1 to m do
    C[A[i]] ← C[A[i]] + 1
for i ← 2 to k do
    C[i] ← C[i-1] + C[i]
for i ← m downto 1 do
    B[C[A[i]]] ← A[i]
    C[A[i]] ← C[A[i]] - 1
```

- TIME: $\Theta(m+k)$
- IF $k = \Theta(m) \Rightarrow \Theta(m)$ ⇝ NOT A COMPARISON SORT!
- **STABLE SORT** = PRESERVE INPUT ORDER (A)

RADIX SORT

- DIGIT-BY-DIGIT SORT LEAST SIGNIFICANT DIGITS FIRST
- AUXILIARY STABLE SORT
- COST ANALYSIS
  - ASSUME COUNTING SORT ON AUXILIARY SORT
  - m WORDS, b BITS SPLIT INTO r BITS
  - $T(m,b) = \Theta\Big(\underbrace{\tfrac{b}{r}}_{\#\text{PASSES}}(m + \underbrace{2^r}_{k})\Big)$

→ r to MINIMIZE: AS LARGE AS POSSIBLE, but $2^r \leq m$ [COUNTING SORT]

→ $r = \log m \Rightarrow \Theta\left(\frac{bm}{\log m}\right)$

→ digits $\in (0, m^d - 1) \Rightarrow b = d \log m \Rightarrow \Theta(dm)$

- :) FAST FOR LARGE INPUTS
- :(LITTLE LOCALITY OF REFERENCE

08. SELECT i^{TH}

NAIVE: SORT $\Rightarrow \Theta(n \log n)$

RANDOMIZED DIVIDE and CONQUER

```
RandSelect(A, p, q, i)
if p = q then
 | return A[p]
r ← RandPartition(A, p, q)
k ← r - p + 1
if k = i then
 | return A[k]
else if k > i then
 | return RandSelect(A, p, r-1, i)
else
 | return RandSelect(A, r+1, q, i-k)
```

LINEAR IN AVERAGE

- AVERAGE: $\Theta(n)$
- WORST CASE: $\Theta(n^2)$

(LIKE QUICKSORT)

DETERMINISTIC VERSION (DERANDOMIZATION)

SLOW IN PRACTICE CONSTANT OF n LARGE

LINEAR IN THE WORST CASE

Select (A, i, n)

divide n elements into groups of 5, find median of each	$\Theta(n)$
RECURSIVELY Select median of the medians, x	$T(n/5)$
partition around x. $k = \text{rank}(x)$	$\Theta(n)$
if $k = i$ then return x	
else if $k > i$ then RECURSIVELY Select i^{th} in left part	$T(3n/4)$
else RECURSIVELY SELECT $(i-k)^{TH}$ in right part	

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

$$= \Theta(n)$$

(cn, FOR c LARGE ENOUGH)

09. PRIMALITY TEST

DETERMINISTIC NAIVE

Primality (m) $\Theta(\sqrt{m})$

```
if m = 2 then return true
if m even then return false
for i ← 1 to √m/2 do
    if 2i+1 divides m then
        return false
return true
```

RANDOMIZED: CAN DO BETTER! ONE-SIDED MONTECARLO (FALSE-BIASED)

- K ITERATIONS $p(\text{error}) \leq p^k$
- FERMAT THEOREM: p PRIME => p DIVIDES $2^{p-1}-1$
- FERMAT'S LITTLE THEOREM: p PRIME => p DIVIDES $a^{p-1}-1$, $a \in (0, p)$

```
Primality (m)
a ← random, ∈[2, m-1]
z ← a^(m-1) mod m
if z = 1 then m is POSSIBLY PRIME    // IF NOT: CARMICHAEL NUMBER, FERMAT'S PSEUDOPRIME
else m is NOT PRIME                  // FOR SURE
```

- p PRIME => $a^2 \bmod p = 1$ ONLY FOR $a = 1$, $a = p-1$ OTHERWISE: NON-TRIVIAL SQUARE ROOT
- FAST EXPONENTIATION REDUCES COMPLEXITY TO $\log m$
- m COMPOSITE => AT MOST $(m-9)/4$ PrimalityTest FAIL -> SMALL ERR
- APPLICATION: PUBLIC KEY CRYPTOSYSTEM (RSA)

10. RANDOM DATA STRUCTURES

DICTIONARY

- OPERATIONS search(x,S), insert(x,S), delete(x,S). min(S), max(S), list(S), union(S_1, S_2), split(S, x, S_1, S_2)
- IMPLEMENTATIONS
  - ARRAY
    - ORDERED: O(log n), O(n), O(n)
    - UNORDERED: O(n), O(1), O(n)
  - BINARY SEARCH TREE ☹ MAY LEAD TO A LIST 
  - AVL TREE ROTATION ON UPDATE OPERATION => height (log n)
  - SPLAY TREE ACCESSED NODE → ROOT
    - ☺ AMORTIZE COST O(log n). GOOD FOR FREQUENTLY ACCESSED
    - ☹ ANY OP CAN PERFORM log NUMBER OF ROTATIONS. NO GUARANTEE EACH OP FAST
  - **TREAPS**
    - SEARCH TREE PROPERTY: key(x) < key(r) ? x in LEF SUBTREE : x in RIGHT SUBTREE
    - HEAP PROPERTY: x CHILD OF y => priority(x) > priority(y)

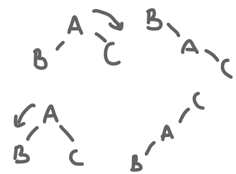
      } UNIQUE TREE
    - if SAME PRIORITY => RANDOMLY EXTEND BITS

searchByKey(root, k)

```
v ← root
while v ≠ NIL do
    if key(v) = k then return v
    else if key(v) < k then v ← v.rightChild
    else if key(v) > k      v ← v.leftChild
return v
```

O(log n) ⇝ BALANCED TREE

insert(x)
choose prio(x)
search(x)
insert x as leaf
restore heap property:
  while prio(parent(x)) > prio(x) do
    if x is leftChild then
      RotateRight(parent(x))
    else
      RotateLeft(parent(x))



→ EXPECTED TIME: $\log n$ ←
EXPECTED ROTATIONS: 2

delete(x)
find x
while x is not a leaf do
  v = child with min priority
  if v = x.leftChild then
    RotateRight(x)
  else
    RotateLeft(x)
delete(x)

split(T, k, T1, T2) $k \notin T$
create new node x : $key(x) = k$, $prio(x) = -\infty$
insert x
T_1 = left subtree, T_2 = right subtree

→ $O(\log n)$ ←

union(T1, T2) $key(x_1) < key(x_2) \quad \forall x_1 \in T_1, \forall x_2 \in T_2$
choose k: $key(x_1) < k < key(x_2)$
create new node x: $key(x) = k$, $prio(x) = -\infty$
generate Treap T: x root, T_1 left, T_2 right
delete x

- **SKIP LISTS**
  - SORTED LINKED LIST → $O(n)$ SEARCH
  - TO SPEED UP SEARCH, SECOND LEVEL WITH $\sqrt{n}$ EVENLY SPACED NODES
  - $\log n$ SORTED LISTS ⇒ LIKE A BINARY TREE ⟶ SEARCH $O(\log n)$ WITH HIGH PROBABILITY → because RANDOMIZATION!

insert (x) ⟶ $\Theta(\log n)$ w.h.p.
→ insert in bottom list
→ promote x to next level with $\frac{1}{2}$ probability

delete (x) ⟶ $\Theta(\log n)$ w.h.p.
remove x from all lists

11. DYNAMIC PROGRAMMING

LCS (x, y)

- BRUTE-FORCE CHECK FOR EACH SUBSEQ. OF x IF THERE'S AN EQUIVALENT SUBSEQUENCE OF y $O(n2^m)$
- OPTIMAL SUBSTRUCTURE $z = LCS(x,y)$ ⇒ any prefix of z = LCS (a prefix of x, a prefix of y)
- RECURSIVELY: $c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j] \\ \max\{c[i-1,j], c[i,j-1]\} & \text{otherwise} \end{cases}$

```
LCS(x, y, i, j)
if c[i,j] = NIL then          // to avoid redoing computation for OVERLAPPING SUBPROBLEMS
 | if x[i] = y[j] then                                  (mn distinct subproblems)
 | | c[i,j] ← LCS (x, y, i-1, j-1) + 1                  ⇝ MEMOIZATION
 | else
 | | c[i,j] ← max {LCS (x, y, i, j-1), LCS (x, y, i-1, j)}
return c[i,j]
```

- RECONSTRUCT LCS by tracing BACKWARDS

12 DYNAMIC PROGRAMMING BDD = BINARY DECISION DIAGRAMS

TO REPRESENT LOGIC FUNCTIONS COMPACTLY

ROBDD

- REDUCED: CHILDREN OF A NODE SAME, TWO NODES ISOMORPHIC BDD
- ORDERED: COFACTORING VARIABLES → SAME ORDER ALONG ALL PATHS

IMPLEMENTATION: WITH HASH (ACCESSED BY KEY)

- UNIQUE TABLE WITH COLLISION CHAIN (NODES = FUNCTIONS)
  - NO DUPLICATION OF EXISTING NODES
- COMPUTED TABLE
  - NO RECOMPUTATION OF EXISTING RESULTS ← DYNAMIC PROGRAMMING!
  - KEEP RECORD OF (f, g, h) ALREADY COMPUTED BY ITE

ITE OPERATOR $ite(f, g, h) = fg + \bar{f}h$

- CAN IMPLEMENT ANY 2-VARIABLE LOGIC FUNCTION
- IMPLEMENTED RECURSIVELY
- APPLICATION EXAMPLE: TAUTOLOGY

13. AMORTIZED ANALYSIS

STRATEGY FOR ANALYZING A SEQUENCE OF OPERATIONS, TO SHOW THAT THE AVERAGE COST PER OPERATION IS SMALL, EVEN THOUGH THERE MIGHT BE AN EXPENSIVE OPERATION IN THE SEQUENCE

→ GUARANTEES AVERAGE PERFORMANCE IN THE WORST CASE

example: INSERT IN DYNAMIC TABLE WORST CASE: n INSERTIONS, $O(n)$ EACH ⇒ $O(n^2)$? NO!

AGGREGATE METHOD LESS PRECISE: NO SPECIFIC AMORTIZED COST FOR EACH OP.

- COST OF n INSERTIONS IS $O(n)$ ⇒ $O(n)/n = O(1)$ PER OP!

COST OF i^{TH} = $\begin{cases} i & \text{if } i-1 \text{ exact power of } 2 \\ 1 & \text{otherwise} \end{cases}$

ACCOUNTING METHOD

- CHARGE i^{TH} OPERATION WITH A FICTICIOUS AMORTIZED COST $\hat{c}_i$
- $\hat{c}_i = 3$, COST 1 FOR INSERT, WHAT'S LEFT STORED IN BANK ⇝ NOT NEGATIVE . $\sum_{i=0}^{m} c_i \le \sum_{i=0}^{m} \hat{c}_i$
- TOT AMORTIZED COST IS UPPERBOUND TO TOT TRUE COST ↳ USED IN DOUBLING

POTENTIAL METHOD BANK ACCOUNT → POTENTIAL ENERGY

- INITIAL DATA STRUCTURE D_0, OPERATION i : $D_{i-1} \to D_i$
- POTENTIAL FUNCTION Φ : $\Phi\{D_0\} = 0$, $\Phi\{D_i\} \ge 0 \quad \forall i$
- AMORTIZED COST : $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- POTENTIAL DIFFERENCE $\Delta\Phi_i$
  - $\Delta\Phi_i > 0 \Rightarrow \hat{c}_i > c_i$ ⇝ STORE
  - $\Delta\Phi_i < 0 \Rightarrow \hat{c}_i < c_i$ ⇝ DATA STRUCTURE HELPS PAYING OP.

14. COMPETITIVE ANALYSIS

S SEQUENCE OF OPERATIONS

- **OFFLINE ALGORITHM** : CAN SEE THE WHOLE SEQUENCE IN ADVANCE
- **ONLINE ALGORITHM** (A): MUST EXECUTE OPERATIONS IMMEDIATELY, WITH NO KNOWLEDGE OF FUTURE OPERATIONS
- GOAL: MINIMIZE COST $C_A(S)$

A α-COMPETITIVE if $\exists k : \forall S,\ C_A(S) \le \alpha \underbrace{C_{OPT}}_{\text{OPTIMAL OFFLINE}}(S) + \underbrace{K}_{\text{CONST}}$

SELF-ORGANIZING LISTS ⇝ ACCESSED ELEMENTS CLOSER TO FRONT

- WORST CASE $C_A(S) = \Omega(|S|m)$
- IDEA: MOST FREQ. ACCESSED ELEMENTS FIRST
- **MOVE-TO-FRONT HEURISTICS**
  - ACCESS x, MOVE IT TO FRONT $c = 2\,\mathrm{rank}(x)$
  - 4-COMPETITIVE