

What You Should Know About Algorithm Design

Hengfeng Wei

hfwei@nju.edu.cn

April 19, 2018











Problem P Algorithm A

Inputs: \mathcal{X}_n of size n

Problem P

Algorithm A

Inputs: \mathcal{X}_n of size n

$$W_A(n) = \max_{x \in \mathcal{X}_n} T_A(x)$$

$$B_A(n) = \min_{x \in \mathcal{X}_n} T_A(x)$$

$$A_A(n) = \sum_{x \in \mathcal{X}_n} T_A(x) \cdot P(x) = \mathbb{E}[T_A] = \sum_{t \in T_A(\mathcal{X}_n)} t \cdot P(T = t)$$

Problem P

Algorithm A

Inputs: \mathcal{X}_n of size n

$$W_A(n) = \max_{x \in \mathcal{X}_n} T_A(x)$$

$$B_A(n) = \min_{x \in \mathcal{X}_n} T_A(x)$$

$$A_A(n) = \sum_{x \in \mathcal{X}_n} T_A(x) \cdot P(x) = \mathbb{E}[T_A] = \sum_{t \in T_A(\mathcal{X}_n)} t \cdot P(T = t)$$

$$T_P(n) =$$

Problem P

Algorithm A

Inputs: \mathcal{X}_n of size n

$$W_A(n) = \max_{x \in \mathcal{X}_n} T_A(x)$$

$$B_A(n) = \min_{x \in \mathcal{X}_n} T_A(x)$$

$$A_A(n) = \sum_{x \in \mathcal{X}_n} T_A(x) \cdot P(x) = \mathbb{E}[T_A] = \sum_{t \in T_A(\mathcal{X}_n)} t \cdot P(T = t)$$

$$T_P(n) = \min_{A \text{ solves } P} W_A(n)$$

Problem P

Algorithm A

Inputs: \mathcal{X}_n of size n

$$W_A(n) = \max_{x \in \mathcal{X}_n} T_A(x)$$

$$B_A(n) = \min_{x \in \mathcal{X}_n} T_A(x)$$

$$A_A(n) = \sum_{x \in \mathcal{X}_n} T_A(x) \cdot P(x) = \mathbb{E}[T_A] = \sum_{t \in T_A(\mathcal{X}_n)} t \cdot P(T = t)$$

$$T_P(n) = \min_{A \text{ solves } P} W_A(n) = \min_{A \text{ solves } P} \max_{x \in \mathcal{X}_n} T_A(x)$$



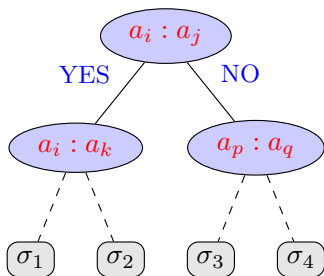
Algorithm Designer



Algorithm Designer



Lower Bound Prover



Decision Trees



Adversary Argument

Lower Bound for Comparison-based Sorting

Prove a lower bound of $\Omega(n \log n)$ on the time complexity of any **comparison-based** sorting algorithm on inputs of size n .

Lower Bound for Comparison-based Sorting

Prove a lower bound of $\Omega(n \log n)$ on the time complexity of any **comparison-based** sorting algorithm on inputs of size n .

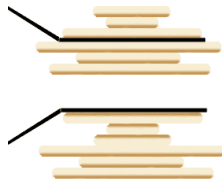
BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

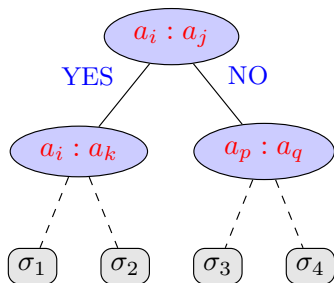
Christos H. PAPADIMITRIOU*†

Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.



Decision Tree Model

Decision Tree Model



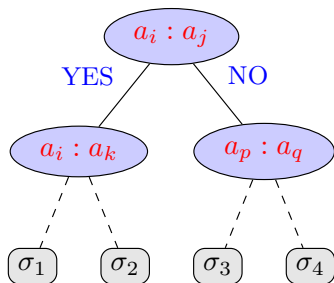
Nodes: comparisons $a_i : a_j$

$<, \leq, =, \geq, >$

Edges: two-way decisions (Y/N)

Leaves: possible permutations

Decision Tree Model



Nodes: comparisons $a_i : a_j$

$<, \leq, =, \geq, >$

Edges: two-way decisions (Y/N)

Leaves: possible permutations

Assumption:

All the input elements are **distinct**.

$$a_i < a_j$$

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree

Decision Tree Model

Any Comparison-based Sorting Algorithm modeled by A Decision Tree

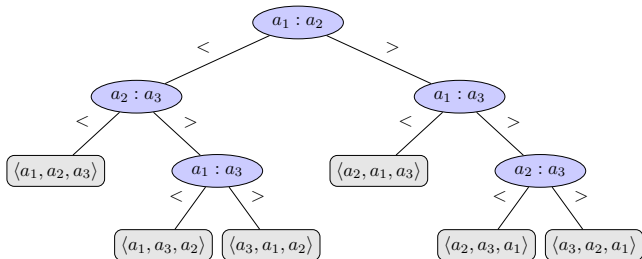


Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree



The decision tree for **insertion sort** on three elements.

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree

```
1: procedure           -SORT( $A, n$ )
2:   for  $i \leftarrow 1$  to  $n - 1$  do
3:     for  $j \leftarrow i + 1$  to  $n$  do
4:       if  $A[j] < A[i]$  then
5:         SWAP( $A[j], A[i]$ )
```

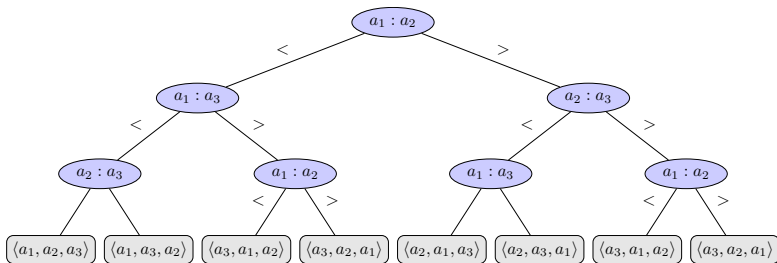
Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree

```
1: procedure SELECTION-SORT( $A, n$ )
2:   for  $i \leftarrow 1$  to  $n - 1$  do
3:     for  $j \leftarrow i + 1$  to  $n$  do
4:       if  $A[j] < A[i]$  then
5:         SWAP( $A[j], A[i]$ )
```

Decision Tree Model

Any Comparison-based Sorting Algorithm $\xRightarrow{\text{modeled by}}$ A Decision Tree



The decision tree for **selection sort** on three elements.

Decision Tree Model

Any Comparison-based Sorting Algorithm \mathcal{A} modeled by A Decision Tree \mathcal{T}

Decision Tree Model

Any Comparison-based Sorting Algorithm \mathcal{A} $\xRightarrow{\text{modeled by}}$ A Decision Tree \mathcal{T}

Algorithm \mathcal{A} on a specific input of size n $\xRightarrow{\text{modeled by}}$ A path through \mathcal{T}

Decision Tree Model

Any Comparison-based Sorting Algorithm \mathcal{A} $\xRightarrow{\text{modeled by}}$ A Decision Tree \mathcal{T}

Algorithm \mathcal{A} on a specific input of size n $\xRightarrow{\text{modeled by}}$ A path through \mathcal{T}

Worst-case time complexity of \mathcal{A} $\xRightarrow{\text{modeled by}}$ The height of \mathcal{T}

Decision Tree Model

Any Comparison-based Sorting Algorithm \mathcal{A} $\xRightarrow{\text{modeled by}}$ A Decision Tree \mathcal{T}

Algorithm \mathcal{A} on a specific input of size n $\xRightarrow{\text{modeled by}}$ A path through \mathcal{T}

Worst-case time complexity of \mathcal{A} $\xRightarrow{\text{modeled by}}$ The height of \mathcal{T}

Worst-case Lower Bound of Comparison-based Sorting
(on inputs of size n)

$\xRightarrow{\text{modeled by}}$
The Minimum Height of All \mathcal{T} s

Worst-case Lower Bound of Comparison-based Sorting (on inputs of size n)

modeled by

The Minimum Height of All \mathcal{T}_s

Worst-case Lower Bound of Comparison-based Sorting (on inputs of size n)

modeled by

The Minimum Height of All \mathcal{T}_s

To be a correct sorting algorithm:

of leaves $\geq n!$

Worst-case Lower Bound of Comparison-based Sorting
(on inputs of size n)

modeled by


The Minimum Height of All \mathcal{T}_s

To be a correct sorting algorithm:

$$\# \text{ of leaves} \geq n!$$

To be a full binary tree:

$$\# \text{ of leaves} \leq 2^h$$

Lower Bound for Comparison-based Sorting

$$n! \leq \# \text{ of leaves} \leq 2^h$$

Lower Bound for Comparison-based Sorting

$$n! \leq \# \text{ of leaves} \leq 2^h$$

$$h \geq \log n! = \Omega(n \log n)$$

Lower Bound for Comparison-based Sorting

$$n! \leq \# \text{ of leaves} \leq 2^h$$

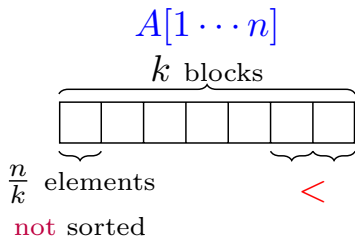
$$h \geq \log n! = \Omega(n \log n)$$

Stirling Formula (by *James Stirling*):

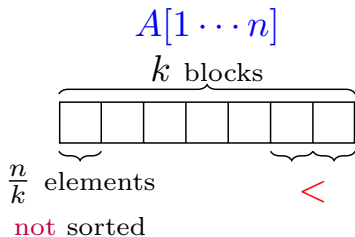
$$n! = \Theta\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n\right)$$



Definition (K -sorting (Problem 6.8))

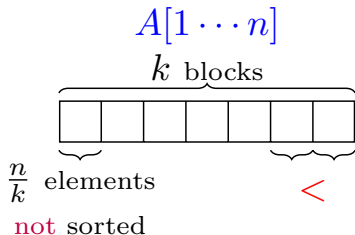


Definition (K -sorting (Problem 6.8))



$$O(n \log k)$$

Definition (K -sorting (Problem 6.8))



$$O(n \log k)$$

$$n = 16, k = 4, \frac{n}{k} = 4$$

1, 2, 4, 3; 7, 6, 8, 5; 10, 11, 9, 12; 15, 13, 16, 14

k -sorted

k -sorted

1-sorted

k -sorted

1-sorted \rightarrow 2-sorted

k -sorted

1-sorted \rightarrow 2-sorted \rightarrow 4-sorted

k -sorted

1-sorted \rightarrow 2-sorted \rightarrow 4-sorted $\rightarrow \dots \rightarrow n$ -sorted

k -sorted

1-sorted \rightarrow 2-sorted \rightarrow 4-sorted $\rightarrow \dots \rightarrow n$ -sorted

Quicksort (with median as pivot) stops after the $\log k$ recursions.

k -sorted

1-sorted \rightarrow 2-sorted \rightarrow 4-sorted $\rightarrow \dots \rightarrow n$ -sorted

Quicksort (with median as pivot) stops after the $\log k$ recursions.

$$\Theta(n \log k)$$

$$\Omega(n \log k)$$

$$\Omega(n \log k)$$

$$L \geq$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k}$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k} = \binom{n}{n/k, \dots, n/k}$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k} = \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k} = \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$H \geq \log \left(\frac{n!}{((\frac{n}{k})!)^k} \right)$$

$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k} = \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

$$H \geq \log \left(\frac{n!}{((\frac{n}{k})!)^k} \right) = \Omega(n \log k)$$

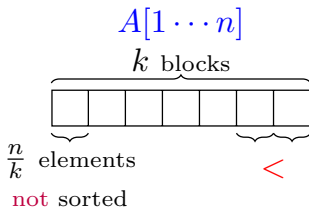
$$\Omega(n \log k)$$

$$L \geq \binom{n}{n/k} \binom{n - n/k}{n/k} \cdots \binom{n/k}{n/k} = \binom{n}{n/k, \dots, n/k} = \frac{n!}{((\frac{n}{k})!)^k}$$

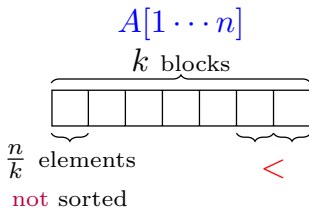
$$H \geq \log \left(\frac{n!}{((\frac{n}{k})!)^k} \right) = \Omega(n \log k)$$

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \implies \log n! \sim n \log n$$

Sorting the k -sorted array

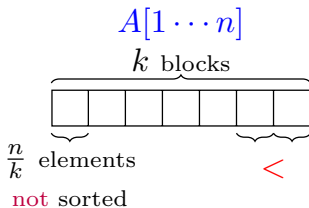


Sorting the k -sorted array



$$O\left(n \log \frac{n}{k}\right)$$

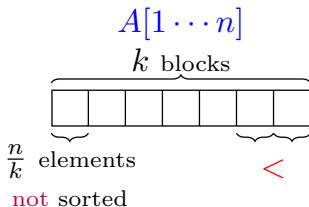
Sorting the k -sorted array



$$L \geq \left(\left(\frac{n}{k}\right)!\right)^k$$

$$O\left(n \log \frac{n}{k}\right)$$

Sorting the k -sorted array



$$L \geq \left(\left(\frac{n}{k}\right)!\right)^k$$

$$O\left(n \log \frac{n}{k}\right)$$

$$H \geq \log\left(\left(\frac{n}{k}\right)!\right)^k = \Omega\left(n \log \frac{n}{k}\right)$$

Bolts and Nuts (Problem 6.9)



Bolts and Nuts (Problem 6.9)



Quicksort

Bolts and Nuts (Problem 6.9)



Quicksort

$$A(n) = O(n \log n)$$

Bolts and Nuts (Problem 6.9)



Quicksort

$$A(n) = O(n \log n)$$

In the worst case:

- ▶ “Matching Nuts and Bolts” by Alon *et al.*, $\Theta(n \log^4 n)$
- ▶ “Matching Nuts and Bolts **Optimality**” by Bradford, 1995, $\Theta(n \log n)$



$$\Omega(n \log n)$$



$$\Omega(n \log n)$$

$$3^H \geq L \geq n!$$



$$\Omega(n \log n)$$

$$3^H \geq L \geq n! \implies H \geq \log n! \implies H = \Omega(n \log n)$$

Repeated Elements (Problem 6.13)

$$R[1 \dots n]$$

$$\# > \lfloor \frac{n}{13} \rfloor$$

To find all $\frac{n}{13}$ -repeated elements

$\text{CHECK}(R[i], R[j])$

$$\# > \lfloor \frac{n}{k} \rfloor$$

$$\# > \lfloor \frac{n}{k} \rfloor$$

$$\Omega(n \log k)$$

$$\# > \lfloor \frac{n}{k} \rfloor$$

$$\Omega(n \log k)$$



Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

Always checking the lower left corner.

Searching in Matrix (Problem 9.8)

$$M : m \times n$$

Row: increasing from left to right

Col: increasing from top to down

$$x \in M?$$

Divide & Conquer

$$T(m, n) = 3T\left(\frac{m}{2}, \frac{n}{2}\right) + 1$$

Always checking the lower left corner.

$$T(m, n) = m + n - 1$$

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals: $i + j = n - 1$ & $i + j = n$

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals: $i + j = n - 1$ & $i + j = n$

No particular ordering requirements on these two diagonals!

Assume $M : n \times n$

$$W(n) \leq 2n - 1$$

$$W(n) \geq 2n - 1$$

By Adversary Argument!

Diagonals: $i + j = n - 1$ & $i + j = n$

No particular ordering requirements on these two diagonals!

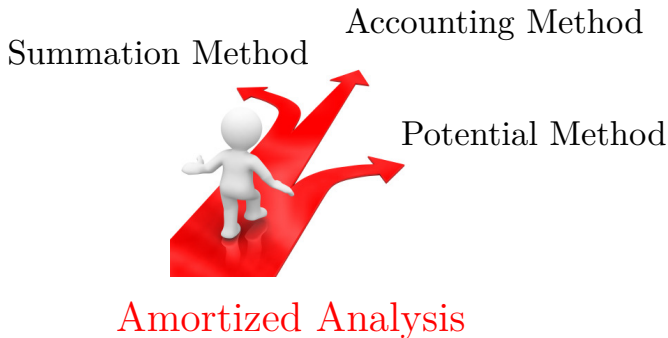
$$i + j \leq n - 1 \implies x > M_{ij}$$

$$i + j > n - 1 \implies x < M_{ij}$$

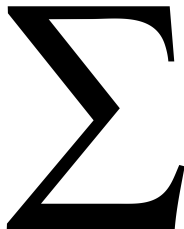
Amortized Analysis



Amortized analysis is
an algorithm analysis technique for
analyzing a sequence of operations
irrespective of the input to show that
the average cost per operation is small, even though
a single operation within the sequence might be expensive.



The Summation Method



$$O_1, O_2, \dots, O_n$$

$$C_1, C_2, \dots, C_n$$

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$\forall i, \hat{c}_i = \frac{\left(\sum_{i=1}^n c_i \right)}{n}$$

The Summation Method for Array Doubling

The Summation Method for Array Doubling

On **any sequence** of n INSERTs on an **initially empty** array.

The Summation Method for Array Doubling

On **any sequence** of n INSERTs on an **initially empty** array.

$o_i :$	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
$c_i :$	1	2	3	1	5	1	1	1	9	1

The Summation Method for Array Doubling

On **any sequence** of n INSERTs on an **initially empty** array.

$o_i :$	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
$c_i :$	1	2	3	1	5	1	1	1	9	1

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

The Summation Method for Array Doubling

On **any sequence** of n INSERTs on an **initially empty** array.

$$\begin{array}{cccccccccc} o_i : & o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} \\ c_i : & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

The Summation Method for Array Doubling

On **any sequence** of n INSERTs on an **initially empty** array.

$$\begin{array}{cccccccccc} o_i : & o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} \\ c_i : & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 9 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1) + 1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

$$\boxed{\forall i, \hat{c}_i = 3}$$

The Accounting Method



O_1, O_2, \dots, O_n

C_1, C_2, \dots, C_n

a_1, a_2, \dots, a_n

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, \quad a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$O_1, O_2, \dots, O_n$$

$$C_1, C_2, \dots, C_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, \ a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$\forall n, \sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i$$

$$o_1, o_2, \dots, o_n$$

$$c_1, c_2, \dots, c_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, \quad a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$\forall n, \sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i \implies \forall n, \sum_{i=1}^n a_i \geq 0$$

$$O_1, O_2, \dots, O_n$$

$$C_1, C_2, \dots, C_n$$

$$a_1, a_2, \dots, a_n$$

$$\hat{c}_i = c_i + a_i, \quad a_i \geq 0$$

Amortized Cost = Actual Cost + Accounting Cost

$$\forall n, \sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i \implies \forall n, \sum_{i=1}^n a_i \geq 0$$

Key Point: Put the accounting cost on specific objects.

The Accounting Method for Array Doubling

$$Q: \hat{c}_i = 3 \text{ vs. } \hat{c}_i = 2$$

The Accounting Method for Array Doubling

$Q : \hat{c}_i = 3 \text{ vs. } \hat{c}_i = 2$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

The Accounting Method for Array Doubling

$Q : \hat{c}_i = 3 \text{ vs. } \hat{c}_i = 2$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{help move another}}$$

	\hat{c}_i	c_i (actual cost)	a_i (accounting cost)
INSERT (normal)	3	1	2
INSERT (expansion)	3	$1 + t$	$-t + 2$

Simulating a queue Q using two stacks S_1, S_2 (Problem E3)

procedure ENQ(x)

$Push(S_1, x)$

procedure DEQ()

if $S_2 = \emptyset$ **then**

while $S_1 \neq \emptyset$ **do**

$Push(S_2, Pop(S_1))$

$Pop(S_2)$

The Summation Method for Queue Simulation

$$\frac{\left(\sum_{i=1}^n c_i \right)}{n}$$

The Summation Method for Queue Simulation

$$\frac{\left(\sum_{i=1}^n c_i \right)}{n}$$

The operation sequence is *NOT* known.

The Accounting Method for Queue Simulation

<i>item:</i>	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

The Accounting Method for Queue Simulation

<i>item:</i>	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

The Accounting Method for Queue Simulation

<i>item:</i>	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^n a_i \geq 0$$

The Accounting Method for Queue Simulation

item: Push into S_1 Pop from S_1 Push into S_2 Pop from S_2
 1 1 1 1

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\sum_{i=1}^n a_i \geq 0 \iff \sum_{i=1}^n a_i = \#S_1 \times 2$$

The Accounting Method for Queue Simulation

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

$$\#S_1 = t$$

	\hat{c}_i	c_i (<i>actual cost</i>)	a_i (<i>accounting cost</i>)
ENQUEUE	3	1	2
DEQUEUE ($S_2 \neq \emptyset$)	1	1	0
DEQUEUE ($S_2 = \emptyset$)	1	$1 + 2t$	$-2t$

Array Merging Dictionary (Problem E 2)

Array Merging Dictionary (Problem E 2)

$$i \quad s_i = 2^i$$

$$A_0 \quad 1$$

$$A_1 \quad 2$$

$$A_2 \quad 4$$

$$A_3 \quad 8$$

$$\vdots \quad \dots$$

$$A_i \quad 2^i$$

Array Merging Dictionary (Problem E 2)

$$i \quad s_i = 2^i$$

$$A_0 \quad 1$$

$$A_1 \quad 2$$

$$A_2 \quad 4$$

$$A_3 \quad 8$$

$$\vdots \quad \dots$$

$$A_i \quad 2^i$$

$$11 = 2^0 + 2^1 + 2^3$$

$$i \quad e_i$$

$$A_0 \quad [5]$$

$$A_1 \quad [4, 8]$$

$$A_2 \quad []$$

$$A_3 \quad [2, 6, 9, 12, 13, 16, 20, 25]$$

Array Merging Dictionary (Problem E 2)

i	$s_i = 2^i$		$11 = 2^0 + 2^1 + 2^3$
A_0	1		
A_1	2	i	e_i
A_2	4	A_0	[5]
A_3	8	A_1	[4, 8]
\vdots	\dots	A_2	[]
A_i	2^i	A_3	[2, 6, 9, 12, 13, 16, 20, 25]

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

Array Merging Dictionary (Problem E 2)

i	$s_i = 2^i$		$11 = 2^0 + 2^1 + 2^3$
A_0	1		
A_1	2	i	e_i
A_2	4	A_0	[5]
A_3	8	A_1	[4, 8]
\vdots	\dots	A_2	[]
A_i	2^i	A_3	[2, 6, 9, 12, 13, 16, 20, 25]

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

INSERT() : $1 + 2 + 4$;

Array Merging Dictionary (Problem E 2)

i	$s_i = 2^i$		$11 = 2^0 + 2^1 + 2^3$
A_0	1		
A_1	2	i	e_i
A_2	4	A_0	[5]
A_3	8	A_1	[4, 8]
\vdots	\dots	A_2	[]
A_i	2^i	A_3	[2, 6, 9, 12, 13, 16, 20, 25]

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

INSERT() : $1 + 2 + 4$; INSERT() : 1;

Array Merging Dictionary (Problem E 2)

i	$s_i = 2^i$		$11 = 2^0 + 2^1 + 2^3$
A_0	1		
A_1	2	i	e_i
A_2	4	A_0	[5]
A_3	8	A_1	[4, 8]
\vdots	\dots	A_2	[]
A_i	2^i	A_3	[2, 6, 9, 12, 13, 16, 20, 25]

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

INSERT() : $1 + 2 + 4$; INSERT() : 1; INSERT() : $1 + 2$

The Summation Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

The Summation Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

i	c_i
1	1
2	$1 + 2$
3	1
4	$1 + 2 + 4$
5	1
6	$1 + 2$
7	1
8	$1 + 2 + 4 + 8$
\vdots	\dots

The Summation Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

i c_i

1 1

2 $1 + 2$

3 1

4 $1 + 2 + 4$

5 1

6 $1 + 2$

7 1

8 $1 + 2 + 4 + 8$

\vdots \dots

$$\sum_{i=1}^n c_i = \sum_{j=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^j} \rfloor 2^j \leq n(\lfloor \log n \rfloor + 1)$$

The Summation Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

i c_i

1 1

2 $1 + 2$

3 1

4 $1 + 2 + 4$

5 1

6 $1 + 2$

7 1

8 $1 + 2 + 4 + 8$

\vdots \dots

$$\sum_{i=1}^n c_i = \sum_{j=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^j} \rfloor 2^j \leq n(\lfloor \log n \rfloor + 1)$$

$$\forall i, \hat{c}_i = 1 + \lfloor \log n \rfloor$$

The Accounting Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

The Accounting Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$

The Accounting Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$



The Accounting Method for “Array Merging Dictionary”

CREATE : 1 MERGE(A_i, A_i) : $2 \cdot 2^i$

$$\hat{c}_i = 1 + \lfloor \log n \rfloor$$



$$\forall n, \sum_{i=1}^n a_i \geq 0$$

Thank
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn