

# CS473 - Algorithms I



## Lecture 12-b Dynamic Tables

*View in slide-show mode*

# Why Dynamic Tables?

- Assume we need a data structure that needs to reside in contiguous memory (e.g. linear array, etc.).
- But, we don't know how many objects will be stored in the table ahead of time.
- We may allocate space for a table, but later find out that it is **not enough**.
  - Then, the table must be **reallocated with a larger size**.
  - All the objects stored in the original table must be **copied over into the new table**.

# Why Dynamic Tables?

- Similarly, if many objects are deleted from the table:
  - ▣ It may be worthwhile to reallocate the table with a smaller size.
- This problem is called:  
dynamically expanding and contracting a table

# Why Dynamic Tables?

Using **amortized analysis** we will show that,

The amortized cost of **insertion** and **deletion** is  $O(1)$ .

Even though the actual cost of an operation is large when it **triggers** an **expansion** or a **contraction**.

We will also show how to guarantee that

The unused space in a dynamic table never exceeds a **constant fraction of the total space**.

# Operations

## TABLE-INSERT:

**Inserts** into the table an item that occupies a single slot.

## TABLE-DELETE:

**Removes** an item from the table & frees its slot.

# Load Factor

Load Factor of a Dynamic Table T

$$\alpha(T) = \frac{\text{Number of items stored in the table}}{\text{size}(\text{number of slots}) \text{ of the table}}$$

For an empty table

$$\alpha(T) = \frac{0}{0} = 1$$

by definition

# Insertion-Only Dynamic Tables

## Table-Expansion:

- Assumption:
  - Table is allocated as an array of slots
- A table **fills up** when
  - all slots have been used
  - equivalently, when its load factor becomes 1
- **Table-Expansion** occurs when
  - An item is to be **inserted** into a **full table**

# Insertion-Only Dynamic Tables

- A Common Heuristic
  - Allocate a new table that has twice as many slots as the old one.
- Hence, we have:

$$1 / 2 \leq \alpha(T) \leq 1$$



# Table Insert

TABLE-INSERT ( $T, x$ )

if  $\text{size}[T] = 0$  then

    allocate  $\text{table}[T]$  with 1 slot

$\text{size}[T] \leftarrow 1$

if  $\text{num}[T] = \text{size}[T]$  then

    allocate new-table with  $2 \cdot \text{size}[T]$  slots

    copy all items in  $\text{table}[T]$  into new-table

    free  $\text{table}[T]$

$\text{table}[T] \leftarrow \text{new-table}[T]$

$\text{size}[T] \leftarrow 2 \cdot \text{size}[T]$

insert  $x$  into  $\text{table}[T]$

$\text{num}[T] \leftarrow \text{num}[T] + 1$

end

**table[T]** : pointer to  
block of table storage

**num[T]** : number of  
items in the table

**size[T]** : total number of  
slots in the table

Initially, table is empty, so  
 $\text{num}[T] = \text{size}[T] = 0$

# Example: Dynamic Table Insertion

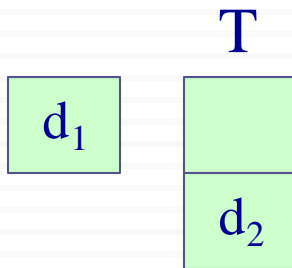
T

$d_1$

INSERT( $d_1$ )

INSERT( $d_2$ )

# Example: Dynamic Table Insertion

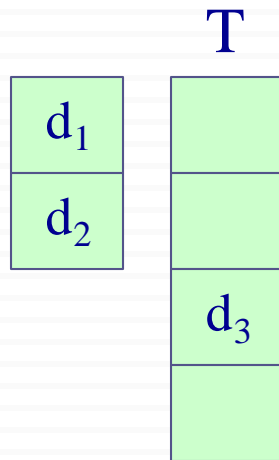


INSERT( $d_1$ )

INSERT( $d_2$ )

INSERT( $d_3$ )

# Example: Dynamic Table Insertion



INSERT( $d_1$ )

INSERT( $d_2$ )

INSERT( $d_3$ )

# Example: Dynamic Table Insertion

T

$d_1$
$d_2$
$d_3$
$d_4$

INSERT( $d_1$ )

INSERT( $d_2$ )

INSERT( $d_3$ )

INSERT( $d_4$ )

INSERT( $d_5$ )

# Example: Dynamic Table Insertion

	T
d <sub>1</sub>	
d <sub>2</sub>	
d <sub>3</sub>	
d <sub>4</sub>	
	d <sub>5</sub>

INSERT(d<sub>1</sub>)

INSERT(d<sub>2</sub>)

INSERT(d<sub>3</sub>)

INSERT(d<sub>4</sub>)

INSERT(d<sub>5</sub>)

# Example: Dynamic Table Insertion

T

$d_1$
$d_2$
$d_3$
$d_4$
$d_5$
$d_6$
$d_7$

INSERT( $d_1$ )

INSERT( $d_2$ )

INSERT( $d_3$ )

INSERT( $d_4$ )

INSERT( $d_5$ )

INSERT( $d_6$ )

INSERT( $d_7$ )

# Table Expansion: Runtime Analysis

- The actual running time of **TABLE-INSERT** is linear in the time to insert individual items.
- Assume that **allocating and freeing** storage is dominated by the **cost of transferring items**.
- Assign a **cost of 1** to each **elementary insertion**.
- Analyze a sequence of **n TABLE-INSERT** operations on an **initially empty table**.



# Cost of Table Expansion

- What is the cost  $c_i$  of the  $i^{\text{th}}$  operation if there is room in the current table?

$$c_i = 1 \quad (\text{only one elementary insert operation})$$

- What is the cost  $c_i$  of the  $i^{\text{th}}$  operation if the current table is full?

$$c_i = i$$

$i-1$  for the items that must be copied from the old table to the new table.

$1$  for the elementary insertion of the new item

# Cost of Table Expansion

- What is the worst-case runtime of  $n$  INSERT operations?

The worst case cost of 1 INSERT operation is  $O(n)$

Therefore, the total running time is  $O(n^2)$

- **This bound is not tight!**

Expansion does not occur so often in the course of  $n$  INSERT operations

# Amortized Analysis of INSERT: Aggregate Method

- Table is initially empty.
- Compute the total cost of  $n$  INSERT operations.
- When does the  $i^{\text{th}}$  operation require an expansion?

only when  $i-1$  is a power of 2

$i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
$c_i$	1	2	3	1	5	1	1	1	9	1	1	1	1	1	1	1	17	1	1	1	...
elem. ins	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
Expansion cost		1	2		4				8								16				

# Amortized Analysis of INSERT: Aggregate Method

Reminder:  $c_i$  is the actual cost of the  $i^{\text{th}}$  INSERT operation

$$c_i = \begin{cases} i & \text{if } i - 1 \text{ is an exact power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

Therefore the total cost of  $n$  TABLE-INSERT operations is:

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

The amortized cost of a single operation is  $3n/n = 3 = O(1)$

# The Accounting Method

Assign the following **amortized costs**

- **Table-Expansion** : \$0
- **Insertion** of a new item : \$3

**Insertion** of a new item:

\$1 (**as an actual cost**) for inserting itself into the table  
+ \$1 (**as a credit**) for **moving itself** in the next expansion  
+ \$1 (**as a credit**) for **moving another item** (in the next expansion) that has already **moved in the last expansion**

# Accounting Method Example

	T	
\$1	<table><tr><td><math>d_1</math></td></tr></table>	$d_1$
$d_1$		
\$1	<table><tr><td><math>d_2</math></td></tr></table>	$d_2$
$d_2$		

INSERT( $d_2$ ) \$3

Note: Amortized cost of INSERT( $d_2$ ): \$3

\$1 spent for the actual cost of inserting  $d_2$

\$1 credit for moving  $d_2$  in the next expansion

\$1 credit for moving  $d_1$  in the next expansion

# Accounting Method Example



Note: When expansion is needed for the next INSERT operation, we have \$1 stored credit for each item to move it to the new memory location.

# Accounting Method Example

	T
\$1	d <sub>1</sub>
	d <sub>2</sub>
\$1	d <sub>3</sub>

INSERT(d<sub>2</sub>)    \$3

INSERT(d<sub>3</sub>)    \$3

Note: Amortized cost of INSERT(d<sub>3</sub>): \$3

\$1 spent for the actual cost of inserting d<sub>3</sub>

\$1 credit for moving d<sub>3</sub> in the next expansion

\$1 credit for moving d<sub>1</sub> in the next expansion



# Accounting Method Example

	T
\$1	d <sub>1</sub>
\$1	d <sub>2</sub>
\$1	d <sub>3</sub>
\$1	d <sub>4</sub>

INSERT(d<sub>2</sub>)    \$3

INSERT(d<sub>3</sub>)    \$3

INSERT(d<sub>4</sub>)    \$3

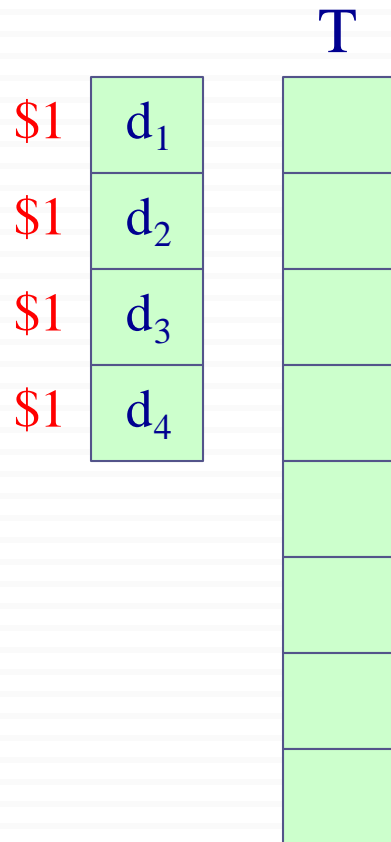
Note: Amortized cost of INSERT(d<sub>4</sub>): \$3

\$1 spent for the actual cost of inserting d<sub>4</sub>

\$1 credit for moving d<sub>4</sub> in the next expansion

\$1 credit for moving d<sub>2</sub> in the next expansion

# Accounting Method Example



INSERT(d<sub>2</sub>)    \$3

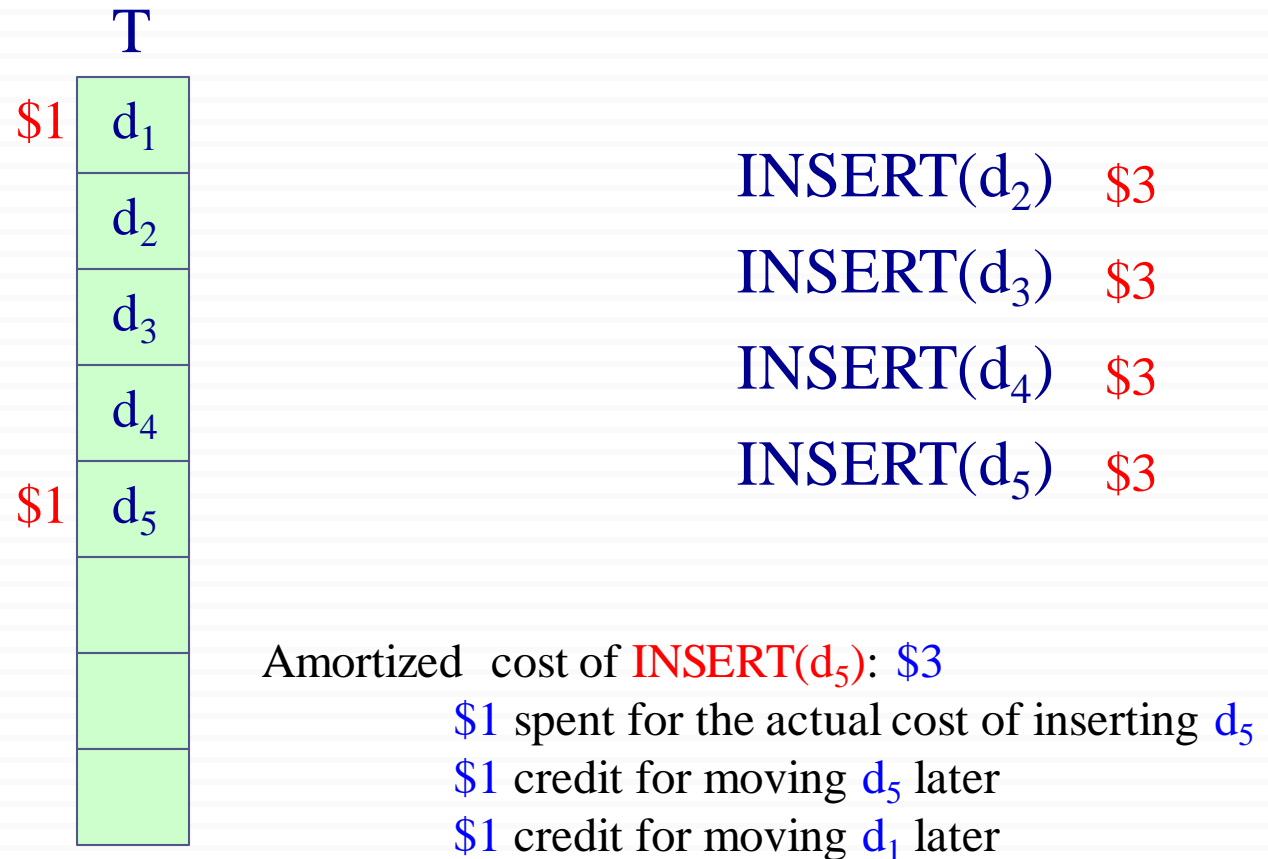
INSERT(d<sub>3</sub>)    \$3

INSERT(d<sub>4</sub>)    \$3

INSERT(d<sub>5</sub>)

*Note:* When expansion is needed for the next INSERT operation, we have \$1 stored credit for each item to move it.

# Accounting Method Example



# Accounting Method Example

T		
\$1	d <sub>1</sub>	INSERT(d <sub>2</sub> ) \$3
\$1	d <sub>2</sub>	INSERT(d <sub>3</sub> ) \$3
	d <sub>3</sub>	INSERT(d <sub>4</sub> ) \$3
	d <sub>4</sub>	INSERT(d <sub>5</sub> ) \$3
\$1	d <sub>5</sub>	INSERT(d <sub>6</sub> ) \$3
\$1	d <sub>6</sub>	
		Amortized cost of INSERT(d <sub>6</sub> ): \$3
		\$1 spent for the actual cost of inserting d <sub>6</sub>
		\$1 credit for moving d <sub>6</sub> later
		\$1 credit for moving d <sub>2</sub> later

# Accounting Method Example

Size of the table:  $M$

Immediately after an expansion (just before the insertion)

$\text{num}[T] = M/2$  and  $\text{size}[T] = M$  where  $M$  is a power of 2.

Table contains **no credits**

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

# Accounting Method Example

## 1st insertion

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X	X	X	X	X	X	X	X	Z							
\$1								\$1							

(c) (b) (a) \$1 for insertion

## 2nd insertion

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X	X	X	X	X	X	X	X	Z	Z						
\$1	\$1							\$1	\$1						

# Accounting Method Example

## $M/2$ th Insertion

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X	X	X	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z
\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1	\$1

Thus, by the time the table contains  $M$  items and is full

- each item in the table has \$1 of credit to pay for its move during the next expansion

# Amortized Analysis of INSERT: Potential Method

## Practical guideline reminder:

*Choose a potential function that increases a little after every cheap operation, and decreases a lot after an expensive operation.*

- Define a potential function  $\Phi$ 
  - that is 0 immediately after an expansion, and
  - that builds to the table size by the time table becomes full.
  
- This way the next expansion can be paid for by the potential.



# Definition of Potential

- One possible potential function  $\Phi$  can be defined as:

$$\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$$

where:

$\text{num}[T]$ : the # of entries stored in table  $T$

$\text{size}[T]$ : the size allocated for table  $T$

- What is the potential value immediately after an expansion?

$$\Phi(T) = 0 \quad \text{because } \text{size}[T] = 2 * \text{num}[T]$$

- What is the potential value immediately before an expansion?

$$\Phi(T) = \text{num}[T] \quad \text{because } \text{size}[T] = \text{num}[T]$$

- The initial value of the potential is 0.

# Definition of Potential

*Potential function:*  $\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$

- Can the potential be ever negative?

No, because the table is always at least half full.

i.e.  $\text{num}[T] \geq \text{size}[T] / 2$

- Since  $\Phi(T)$  is always nonnegative:

The sum of the amortized costs of  $n$  INSERT operations is an upper bound on the sum of the actual costs.

# Analysis of $i$ -th Table Insert

$n_i$  : **num**[T] after the  $i$ -th operation

$s_i$  : **size**[T] after the  $i$ -th operation

$\Phi_i$  : **Potential** after the  $i$ -th operation

Initially we have  $n_i = s_i = \Phi_i = 0$

Note that,  $n_i = n_{i-1} + 1$  always holds.

# Amortized Cost of TABLE-INSERT

*Potential function:*  $\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$

If the  $i^{\text{th}}$  **TABLE-INSERT** does not trigger an expansion:

*Intuitively:*

$\text{size}[T]$  remains the same

$\text{num}[T]$  increases by 1

$\Rightarrow$  **potential change** = 2

**amortized cost** = **real cost** + **potential change**

$= 1 + 2 = 3$

# Amortized Cost of TABLE-INSERT

*Potential function:*  $\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$

If the  $i^{\text{th}}$  **TABLE-INSERT** does not trigger an expansion:

Formally:

$$s_i = s_{i-1} \text{ and } c_i = 1$$

$$\begin{aligned}\hat{c}_i &= c_i + f_i - f_{i-1} = 1 + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= 1 + (2(n_{i-1} + 1) - s_{i-1}) - (2n_{i-1} - s_{i-1}) \\ &= 1 + 2n_{i-1} + 2 - s_{i-1} - 2n_{i-1} + s_{i-1} = 3\end{aligned}$$

# Amortized Cost of TABLE-INSERT

*Potential function:*  $\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$

If the  $i^{\text{th}}$  **TABLE-INSERT** triggers an expansion:

*Intuitively:*

$\text{size}[T]$  is doubled, i.e. increases by  $n_{i-1}$

$\text{num}[T]$  increases by 1

$\Rightarrow$  **potential change**  $= 2 - n_{i-1}$

**real cost:**  $n_{i-1} + 1$  (*copy  $n_{i-1}$  entries to new memory*  
*+ insert the new element*)

**amortized cost**  $=$  **real cost**  $+$  **potential change**  
 $= n_{i-1} + 1 + 2 - (n_{i-1}) = 3$

# Amortized Cost of TABLE-INSERT

*Potential function:*  $\Phi(T) = 2 * \text{num}[T] - \text{size}[T]$

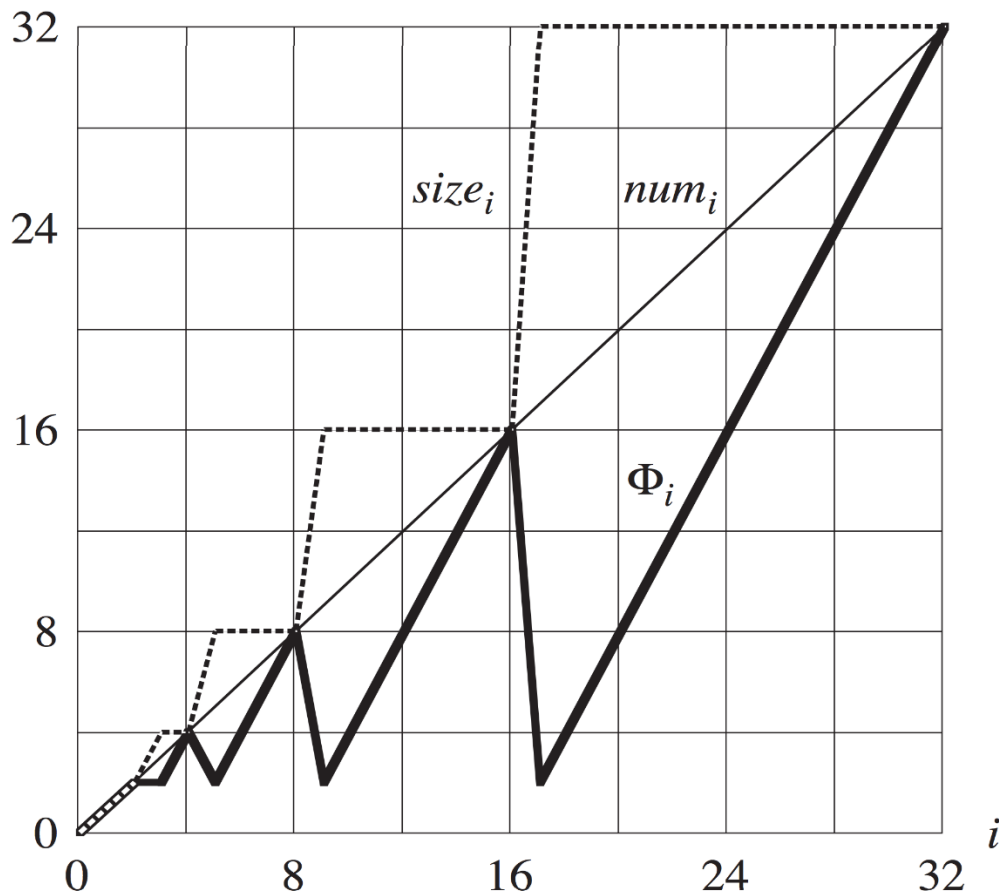
If the  $i^{\text{th}}$  **TABLE-INSERT** triggers an expansion:

Formally:

$$n_{i-1} = s_{i-1}; \quad s_i = 2s_{i-1}; \quad c_i = n_i = n_{i-1} + 1$$

$$\begin{aligned} \hat{c}_i &= c_i + \phi_i - \phi_{i-1} = n_i + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= (n_{i-1} + 1) + (2(n_{i-1} + 1) + 2s_{i-1}) - (2n_{i-1} - s_{i-1}) \\ &= n_{i-1} + 1 + 2n_{i-1} + 2 - 2n_{i-1} - 2n_{i-1} + n_{i-1} = 3 \end{aligned}$$

# A Sequence of TABLE-INSERT Operations



Size of the table is doubled when  $i-1$  is a power of 2.

The potential function **increases gradually** after every INSERT that **doesn't require table expansion**.

The potential function **drops to 2** after every INSERT that **requires table expansion**.



# Supporting Insertions and Deletions

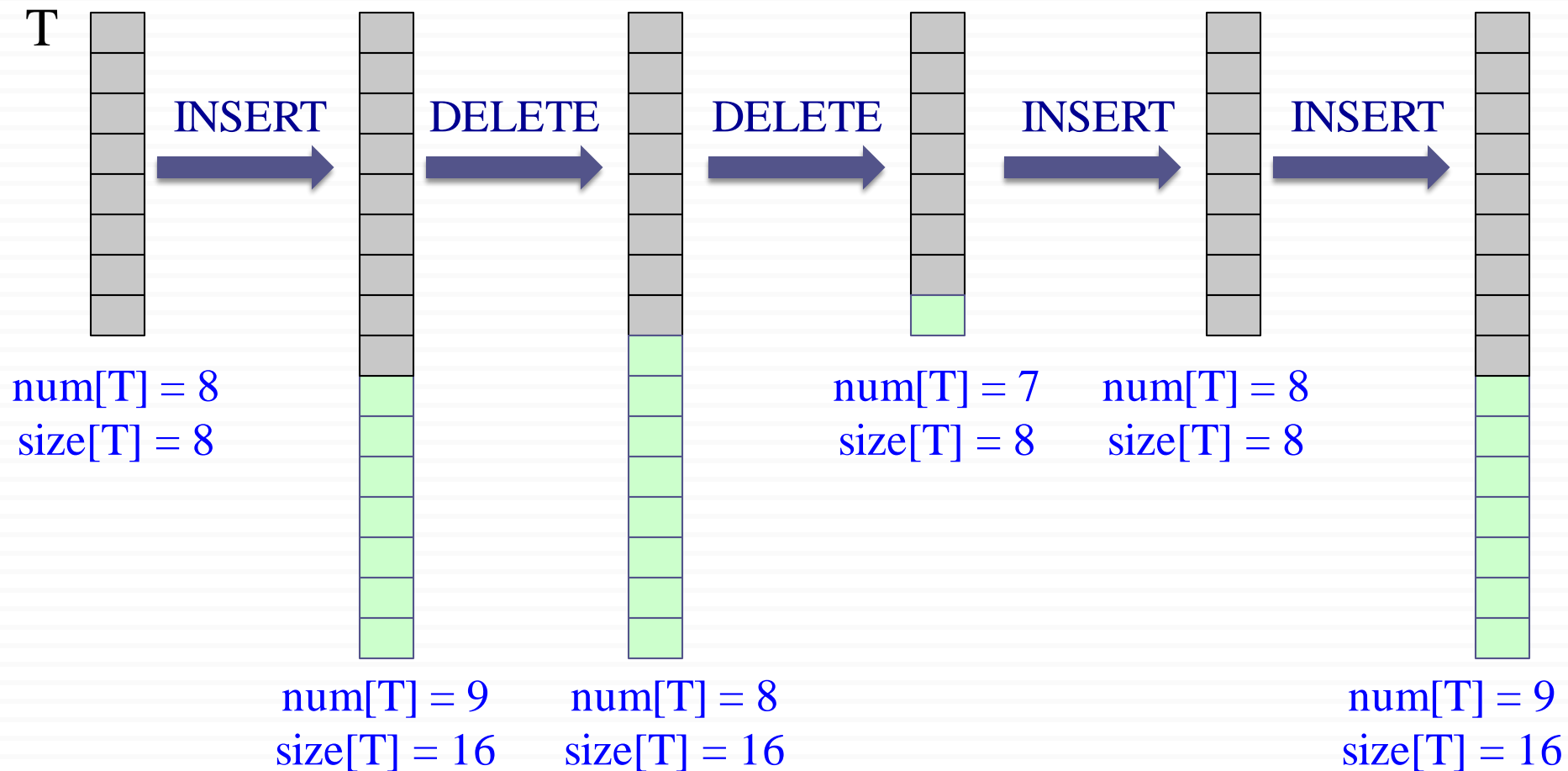
- So far, we have assumed that we only **INSERT** elements into the table. Now, we want to support **DELETE** operations as well.
- **TABLE-DELETE**: Remove the specified item from the table. Contract the table if needed.
- In table contraction, we want to preserve two properties:
  - ▣ The **load factor** of the table is bounded below by a constant.
  - ▣ **Amortized cost** of an operation is bounded above by a constant.
- As before, we assume that the cost can be measured in terms of elementary insertions and deletions.

# Expansion and Contraction

*Load factor reminder:*  $\alpha(T) = \frac{\text{Number of items stored in the table}}{\text{size(number of slots) of the table}}$

- An intuitive strategy for expansion and contraction:
  - ▣ Double the table size when an item is to be inserted into a full table.
  - ▣ Halve the size when a deletion would cause  $\alpha(T) < \frac{1}{2}$
  
- What is the problem with this strategy?
  - ▣ **Good:** It guarantees  $\frac{1}{2} \leq \alpha(T) \leq 1.0$
  - ▣ **Bad:** Amortized cost of an operation can be quite large.

# Worst-Case Behavior for $\alpha(T) \geq \frac{1}{2}$



# Worst-Case for $\alpha(T) \geq 1/2$

Consider the following **worst case** scenario

- We perform  $n$  operations on an **empty table** where  $n$  is a power of 2
- First  $n/2$  operations are all **insertions**, cost a total of  $\Theta(n)$

at the end: we have  $\text{num}[T] = \text{size}[T] = n/2$

- Second  $n/2$  operations repeat the sequence

**I D D I**

that is **I D D I I D D I I D D I ...**

# Worst-Case for $\alpha(T) \geq 1/2$

Example:  $n=16$

$i$ :	1	2	...	7	8		9	10	11	12	13	14	15	16
oper:	I	I	...	I	I		I	D	D	I	I	D	D	I
$n_i$	1	2	...	7	8		9	8	7	8	9	8	7	8
$s_i$	1	2	...	8	8		16	16	8	8	16	16	8	8
							E		C		E		C	

In the second  $n/2$  operations

- The first **INSERT** cause an expansion
- Two further **DELETE**s cause contraction
- Two further **INSERT**s cause expansion ... and so on

Hence there are  $n/8$  expansions and  $n/8$  contractions

The cost of each expansion and contraction is  $\approx n/2$

# Worst-Case for $\alpha(T) \geq 1/2$

Thus the total cost of  $n$  operations is  $\Theta(n^2)$  since

- First  $n/2$  operations :  $3n/2$
- Second  $n/2$  operations :  $(n/4)*(n/2)=n^2/8$

The amortized cost of an operation is  $\Theta(n)$

The problem with this strategy is

- After an expansion, we do not perform enough deletions to pay for a contraction
- After a contraction, we do not perform enough insertions to pay for an expansion

# Improving Amortized Time of Expansion and Contraction

- We saw that if we enforce  $\frac{1}{2} \leq \alpha(T) \leq 1$ , the amortized time becomes  $O(n)$  in the worst case.

- To improve the amortized cost:

Allow  $\alpha(T)$  to drop below  $\frac{1}{2}$ .

- Basic idea:

- ▣ Expansion: Double the table size when an item is inserted into a full table (same as before).

- ▣ Contraction: Halve the table size when a deletion causes:

$$\alpha(T) < \frac{1}{4}$$

# Improving Amortized Time of Expansion and Contraction

- In other words, we enforce:  $\frac{1}{4} \leq \alpha(T) \leq 1$
- Intuition:
  - ▣ Immediately after an expansion, we have  $\alpha(T) = \frac{1}{2}$   
 $\Rightarrow$  At least half of the items in the table must be deleted before a contraction can occur (i.e. when  $\alpha(T) < \frac{1}{4}$ )
  - ▣ Immediately after a contraction, we have  $\alpha(T) = \frac{1}{2}$   
 $\Rightarrow$  The number of items in the table must be doubled before an expansion can occur (i.e. when  $\alpha(T)=1$ ).



# Potential Method for INSERT & DELETE

□ We want to define the potential function  $\Phi(T)$  as follows:

▣ Immediately after an expansion or contraction:

$$\Phi(T) = 0$$

▣ Immediately before an expansion or contraction:

$$\Phi(T) = \text{num}[T]$$

because we need to copy over  $\text{num}[T]$  elements, and the cost of expansion or contraction should be paid by the decrease in potential.

# Potential Method for INSERT & DELETE

- **Reminder:** Immediately after an expansion or contraction,

$$\alpha(T) = 1/2$$

- So, we want to define a potential function  $\Phi(T)$  such that:

$\Phi(T)$  **starts at 0** when  $\alpha(T) = 1/2$

$\Phi(T)$  **gradually increases to**  $\text{num}[T]$ ,

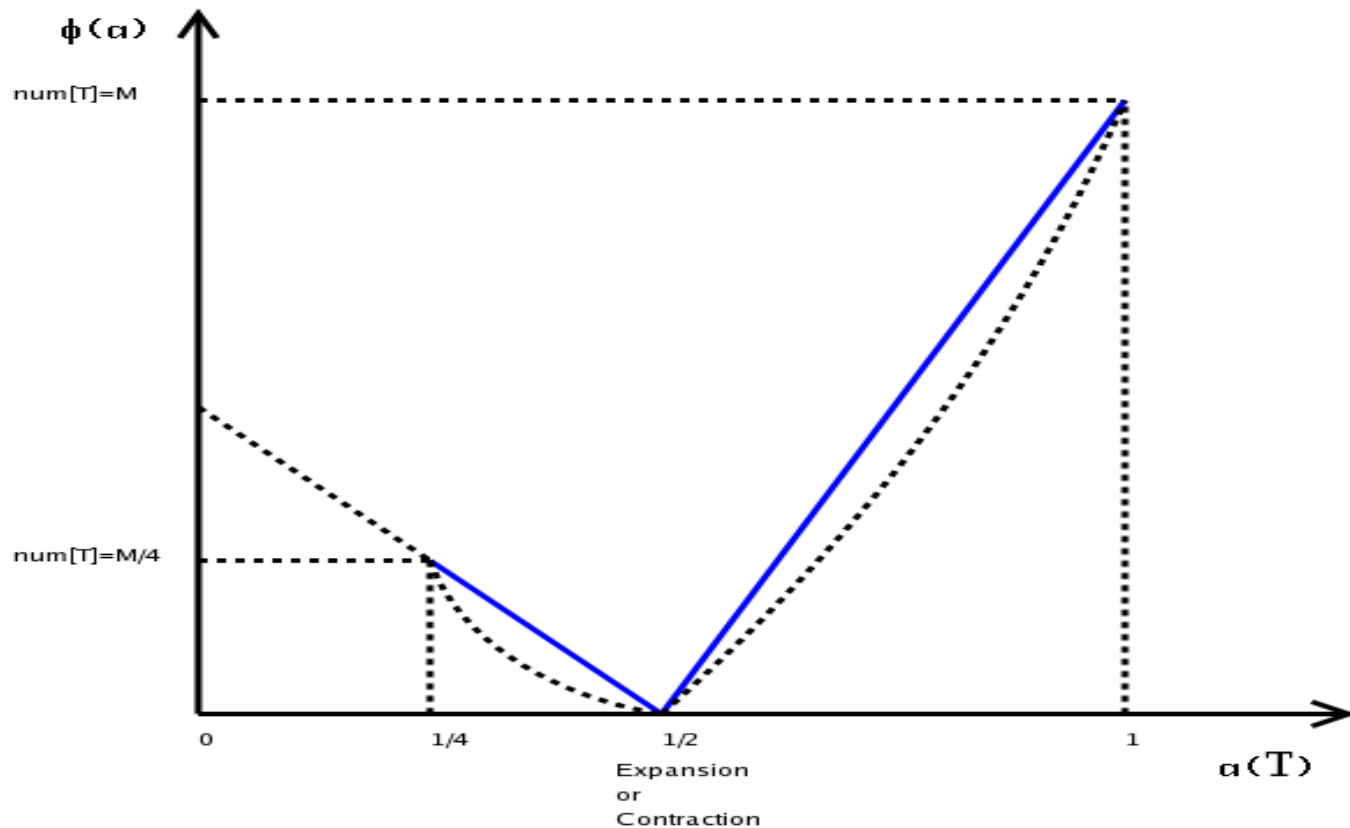
**when**  $\alpha(T)$  **increases to 1**, or

**when**  $\alpha(T)$  **decreases to**  $1/4$

- This way, the next expansion or contraction can be paid by the decrease in potential.

# $\Phi(\alpha)$ w.r.t. $\alpha(T)$

$M = \text{num}[T]$  when an **expansion** or **contraction** occurs



# Definition of New $\Phi$

One such  $\Phi$  is

$$\Phi(T) = \begin{cases} 2num[T] - size[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{size[T]}{2} - num[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

or

$$\Phi(T) = \begin{cases} num[T](2 - 1/\alpha) & \text{if } \alpha(T) \geq \frac{1}{2} \\ num[T](1/2\alpha - 1) & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

# Description of New $\Phi$

$$\Phi(T) = \begin{cases} 2\text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

$$a = \frac{\text{num}[T]}{\text{size}[T]}$$

- ✓  $\Phi = 0$  **when**  $\alpha = \frac{1}{2}$
- ✓  $\Phi = \text{num}[T]$  **when**  $\alpha = \frac{1}{4}$
- ✓  $\Phi = \text{num}[T]$  **when**  $\alpha = 1$
- ✓  $\Phi = 0$  **when** the table is empty  
( $\text{num}[T] = \text{size}[T] = 0, \alpha(T) = 0$ )
- ✓  $\Phi$  is always nonnegative

# Amortized Analysis

We need to analyze the operations:

**TABLE-INSERT** and **TABLE-DELETE**

Notations:

$c_i$ : Actual cost of the  $i^{\text{th}}$  operation

$\hat{c}_i$ : Amortized cost of the  $i^{\text{th}}$  operation

$\Phi_i$ : Potential  $\Phi(T)$  after the  $i^{\text{th}}$  operation

$n_i$ : Number of elements  $\text{num}[T]$  after the  $i^{\text{th}}$  operation

$s_i$ : Table size  $\text{size}[T]$  after the  $i^{\text{th}}$  operation

$\alpha_i$ : Load factor  $\alpha(T)$  after the  $i^{\text{th}}$  operation

# Amortized Analysis: Table Insert – Case 1

- There is no possibility of contraction in any case.
- In all cases:  $n_i = n_{i-1} + 1$

Case 1:  $\alpha_{i-1} \geq \frac{1}{2}$

Analysis is identical to the one we did before for only TABLE-INSERT operation.

$\Rightarrow$  Amortized cost  $\hat{c}_i = 3$  whether the table expands or not.

# Amortized Analysis: Table Insert – Case 2

$$\Phi(T) = \begin{cases} 2\text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

Case 2:  $\alpha_{i-1} < \frac{1}{2}$  and  $\alpha_i < \frac{1}{2}$

There is no possibility of expansion.

Intuitively:

Potential change: -1

Real cost: 1

Amortized cost =  $1 - 1 = 0$



# Amortized Analysis: Table Insert – Case 2

$$\Phi(T) = \begin{cases} 2\text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

Case 2:  $\alpha_{i-1} < 1/2$  and  $\alpha_i < 1/2$

There is no possibility of expansion.

Formally:  $c_i = 1$ ;  $s_i = s_{i-1}$ ;  $n_i = n_{i-1} + 1$

$$\begin{aligned} \hat{c}_i &= c_i + F_i - F_{i-1} = 1 + (s_i / 2 - n_i) - (s_{i-1} / 2 - n_{i-1}) \\ &= 1 + \frac{s_i}{2} - n_i - \frac{s_i}{2} + (n_i - 1) = 0 \end{aligned}$$

# Amortized Analysis: Table Insert – Case 3

$$\Phi(T) = \begin{cases} 2\text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

Case 3:  $\alpha_{i-1} < 1/2$  and  $\alpha_i \geq 1/2$  (which means  $\alpha_i = 1/2$  because  $\text{size}[T]$  is even)

There is no possibility of expansion.

Intuitively:  $n_i = s_i/2$ ;  $n_{i-1} = s_i/2 - 1$

Old potential: 1

New potential: 0

Real cost: 1

Amortized cost =  $1 - 1 = 0$

# Amortized Analysis: Table Insert – Case 3

$$\Phi(T) = \begin{cases} 2\text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < \frac{1}{2} \end{cases}$$

Case 3:  $\alpha_{i-1} < 1/2$  and  $\alpha_i \geq 1/2$  (which means  $\alpha_i = 1/2$  because  $\text{size}[T]$  is even)

There is no possibility of expansion.

Formally:  $c_i = 1$ ;  $n_i = s_i/2$ ;  $n_{i-1} = s_i/2 - 1$ ;  $s_i = s_{i-1}$

$$\begin{aligned} \hat{c}_i &= c_i + F_i - F_{i-1} = 1 + (2n_i - s_i) - (s_{i-1}/2 - n_{i-1}) \\ &= 1 + (2(s_i/2) - s_i) - (s_i/2 - (s_i/2 - 1)) \\ &= 0 \end{aligned}$$

# Amortized Analysis: Table Insert - Summary

Case 1:  $\alpha_{i-1} \geq \frac{1}{2}$

Amortized cost of **TABLE-INSERT** = 3

Case 2:  $\alpha_{i-1} < \frac{1}{2}$  and  $\alpha_i < \frac{1}{2}$

Amortized cost of **TABLE-INSERT** = 0

Case 3:  $\alpha_{i-1} < \frac{1}{2}$  and  $\alpha_i \geq \frac{1}{2}$

Amortized cost of **TABLE-INSERT** = 0

So, the amortized cost of **TABLE-INSERT** is at most 3

# Table Delete

$$n_i = n_{i-1} - 1 \Rightarrow n_{i-1} = n_i + 1$$

Table **expansion** cannot occur.

- $\alpha_{i-1} \leq 1/2$  and  $1/4 \leq \alpha_i < 1/2$  (It does not trigger a **contraction**)

$$s_i = s_{i-1} \text{ and } c_i = 1 \text{ and } \alpha_i < 1/2$$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} = 1 + (s_i / 2 - n_i) - (s_{i-1} / 2 - n_{i-1}) \\ &= 1 + s_i / 2 - n_i - s_i / 2 + (n_i + 1) = 2\end{aligned}$$

# Table Delete

- $\alpha_{i-1} = 1/4$  (It does **trigger** a **contraction**)

$$s_i = s_{i-1}/2 ; n_i = s_{i-1}/2; \text{ and } c_i = n_i + 1$$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} = (n_i + 1) + (s_i / 2 - n_i) - (s_{i-1} / 2 - n_{i-1}) \\ &= n_i + 1 + s_i / 2 - n_i - s_i + s_i / 2 = 1\end{aligned}$$

- $\alpha_{i-1} > 1/2$  ( $\alpha_i \geq 1/2$  )

**Contraction** cannot occur ( $c_i=1 ; s_i = s_{i-1}$ )

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} = 1 + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= 1 + 2n_i - s_i - 2(n_i + 1) + s_i = -1\end{aligned}$$

# Table Delete

- $\alpha_{i-1} = 1/2$  ( $\alpha_i < 1/2$ )

Contraction cannot occur

$$c_i = 1 ; s_i = s_{i-1} ; n_i = s_{i-1}/2; \text{ and } \Phi_{i-1}=0)$$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} = 1 + (s_i / 2 - n_i) - 0 \\ &= 1 + s_i / 2 - n_i \quad \text{but} \quad n_{i+1} = s_i / 2 \\ &= 1 + (n_i + 1) - n_i = 2\end{aligned}$$

# Table Delete

Thus, the amortized cost of a **TABLE-DELETE** operation is at most 2

Since the amortized cost of each operation is bounded above by a constant

The actual time for any sequence of  $n$  operations on a Dynamic Table is  $O(n)$