CS473-Algorithms I

Lecture 12b

Dynamic Tables

Why Dynamic Tables?

In some applications:

- We don't know how many objects will be stored in a table.
- We may allocate space for a table
  - But, later we may 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

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

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, insertions are performed if only

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

Table Insert

```
TABLE-INSERT (T, x)
    if size[T] = 0 then
        allocate table[T] with 1 slot
        size[T] ← 1
    if num[T] = size[T] then
        allocate new-table with 2.size[T] slots
        copy all items in table[T] into new-table
        free table[T]
        table[T] ← new-table[T]
        size[T] ← 2.size[T]
    insert x into table[T]
    num[T] ← 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 num[T] = size[T] = 0

Table Expansion

- Running time of TABLE-INSERT is proportional to the number of elementary insert operations.
- 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 cost c_i of the i-th operation?

- If there is room in the current table (or this is the first operation)
  - $c_i = 1$ (only one elementary insert operation)
- If the current table is full, an expansion occurs, then the cost is $c_i = i$.
  - 1 for the elementary insertion of the new item
  - i-1 for the items that must be copied from the old table to the new table.

Cost of Table Expansion

- If n operations are performed,

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

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

- This bound is not tight, because

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

Amortized Analysis of Insert

The Aggregate Method

Table is initially empty.

Observe:

i-th operation causes an expansion only when i-1 is a power of 2.

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

The Aggregate Method

Therefore the total cost of n TABLE-INSERT operations is

$$\sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j < n + \sum_{j=0}^{\lg n} 2^j = n + 2n = 3n$$

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	...
	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				

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

a) 1 (as an actual cost) for inserting itself into the table

b) 1 (as a credit) for moving itself in the next expansion

c) 1 (as a credit) for moving another item (in the next expansion) that has already moved in the last expansion

Accounting Method Example

Size of the table: M

Immediately after an expansion (just before the insertion)

num[T] = $M/2$ and 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/2th 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

The Potential Method

Define a potential function Φ that is

- 0 immediately after an expansion
- builds to the table size by the time table becomes full

Next expansion can be paid for by the potential.

Definition of Φ

One possible Φ is:

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

Immediately after an expansion

$$size[T] = 2*num[T] \Rightarrow \Phi(T) = 0$$

Immediately before an expansion

$$size[T] = num[T] \Rightarrow \Phi(T) = num[T]$$

The initial value for the potential is 0

Definition of Φ

Since the table is at least half full

(i.e. $num[T] \geq size[T] / 2$)

$\Phi(T)$ is always nonnegative.

Thus, the sum of the amortized cost of *n* TABLE-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

Φ_i : Potential after the i-th operation

Initially we have $n_i = s_i = \Phi_\mathrm{i} = 0$

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

Insert without Expansion

If the *i*-th TABLE-INSERT does not trigger an expansion,

$$s_i = s_{i-1}\,;\, c_i = 1$$

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

Insert with Expansion

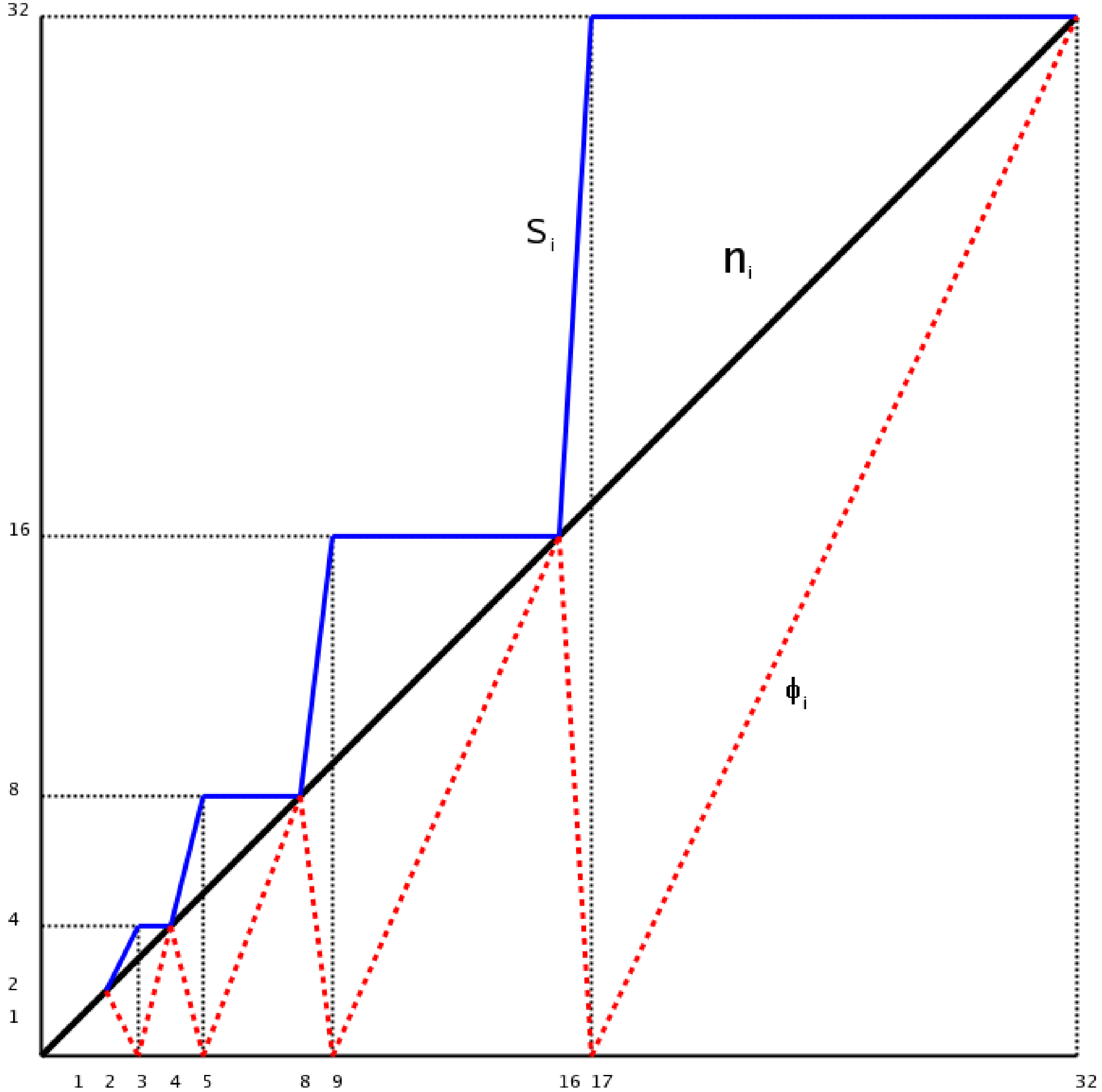
If the i-th TABLE-INSERT does trigger an expansion, then

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

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

$$= (n_{i-1} + 1) + (2(n_{i-1} + 1) + 2s_{i-1}) - (2n_{i-1} - s_{i-1})$$

$$= n_{i-1} + 1 + 2n_{i-1} + 2 - 2n_{i-1} - 2n_{i-1} + n_{i-1} = 3$$


32
16
8
4
2
1
S_i
n_i
ϕ_i
1 2 3 4 5 8 9 16 17 32

Adding Delete Operation

TABLE-DELETE: Remove the specified item from the table. It is often desirable to contract the table.

In table contraction, we would like to preserve two properties

- Load factor of dynamic table is bounded below by a constant
- Amortized cost of a table operation is bounded above by a constant

We assume that the cost can be measured in terms of elementary insertions and deletions

Expansion and Contraction

A natural strategy for expansion and contraction

- Double the table size when an item is inserted into a full table
- Halve the size when a deletion would cause $\alpha(T) < 1/2$

This strategy guarantees $\frac{1}{2} \le \alpha(T) \le 1$

Unfortunately, it can cause the amortized cost of an operation to be quite large

Worst-Case for $\alpha(T) \geq ½$

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 num[T] = 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 $\alpha(T) \geq ½$

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 DELETEs cause contraction
- Two further INSERTs cause expansion ... and so on

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

The cost of each expansion and contraction is ≈ n/2

Worst-Case for $\alpha(T) \geq ½$

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

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

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

The difficulty 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 Expansion – Contraction

We can improve upon this strategy by allowing α(T) to drop below ½

We continue to double the table size when an item is inserted into a full table

But, we halve the table size (perform contraction)

when a deletion causes $\alpha(T) < 1/4$ rather than $\alpha(T) < 1/2$,

Therefore, $1/4 \le \alpha(T) \le 1$

Improving Expansion – Contraction

Hence after an expansion, $\alpha(T) = ½$, thus at least half of the items in the table must be deleted before a contraction can occur.

Similarly, after a contraction $\alpha(T) = ½$, thus the number of items in the table must be doubled by insertions before an expansion can occur.

Potential Method

Define the potential function as follows

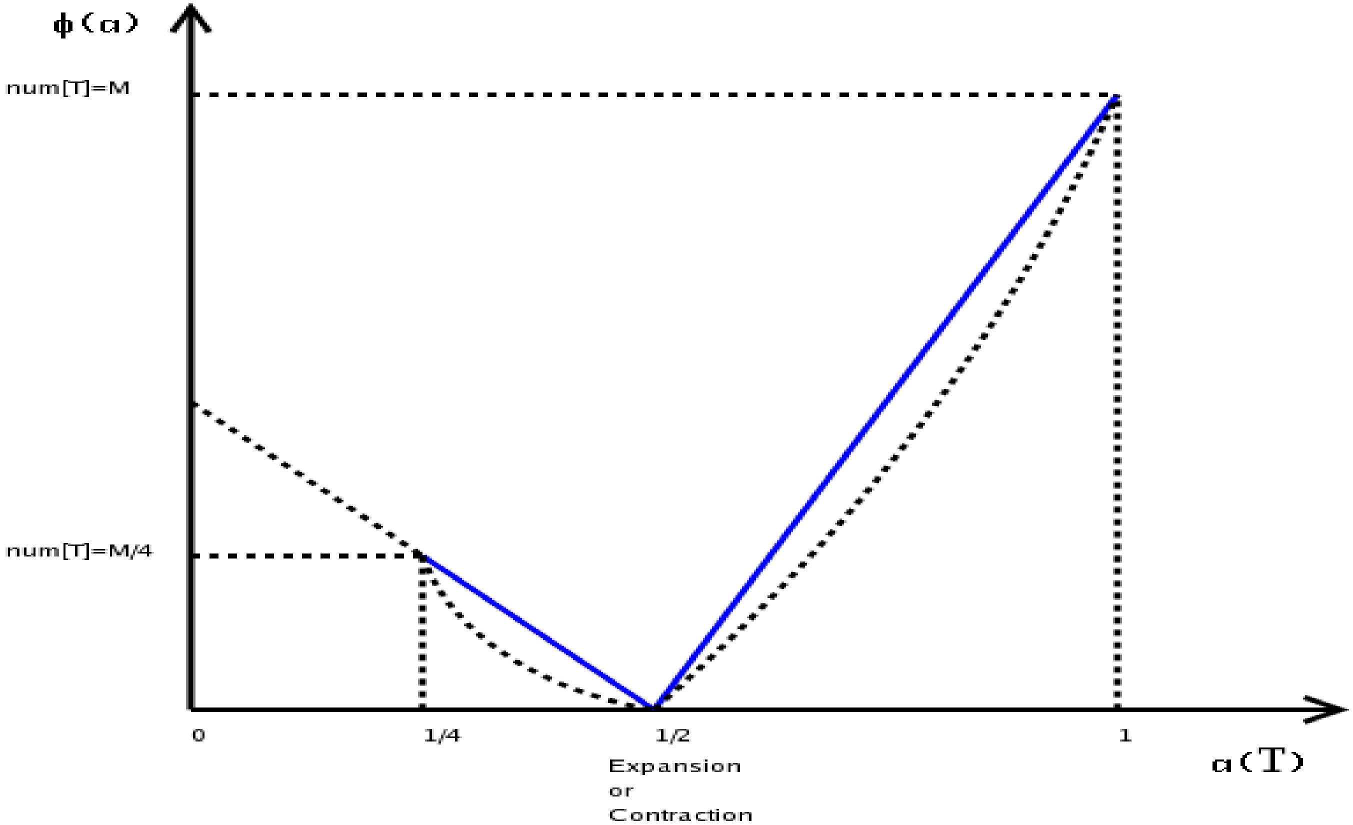
- $\Phi(T) = 0$ immediately after an expansion or contraction
- Recall that, $\alpha(T) = ½$ immediately after and expansion or contraction,

  therefore the potential should build up to num[T]

  as $\alpha(T)$ increases to 1 or decreases to ¼
- So that the next expansion or contraction can be paid by the potential.

$\Phi(\alpha)$ w.r.t. α(T)

M=num[T] when an expansion or contraction occurs



Description of New Φ

One such Φ is

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

or

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

Description of New Φ

- $\Phi = 0$ when $\alpha = ½$
- $\Phi = num[T]$ when $\alpha = ¼$
- $\Phi = 0$ for an empty table

  $(num[T] = size[T]=0, \alpha[T] = 0)$
- Φ is always nonnegative

Amortized Analysis

Operations are:

– TABLE-INSERT

– TABLE-DELETE

c_i : Actual Cost $\quad \hat{c}_i$: Amortized Cost $\quad \Phi_i$: $\Phi(T)$

n_i : $num[T]$ $\quad s_i$: $size[T]$ $\quad \alpha_i$: $\alpha(T)$

after the *i*-th operation

Table Insert

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

Table contraction may not occur.

- $\alpha_{i-1} \geq ½$

  Analysis is identical to that for table expansion

  Therefore, $\hat{c}_i = 3$ whether the table expands or not.

- $\alpha_{i-1} < ½$ and $\alpha_i < ½$

  Expansion may not occur ($\hat{c}_i = 1$, $s_i = s_{i-1}$)

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (3i/2 - n_i) - (s_{i-1}/2 - n_{i-1})$$

$$= 1 + \frac{s_i}{2} - n_i - \frac{s_i}{2} + (n_i - 1) = 0$$

Table Insert

- $\alpha_{i\text{-}1} < ½$ and $\alpha_i \geq ½$

  $\Rightarrow \alpha_i = ½$

  Expansion may not occur ($c_i = 1$; $s_i = s_{i\text{-}1}$; $n_i = s_i / 2$)

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

Thus, amortized cost of a TABLE-INSERT operation is at most 3.

Table Delete

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

Table expansion may not occur.

- $\alpha_{i-1} \leq ½$ and $¼ \leq \alpha_i < ½$ (It does not trigger a contraction)

$s_i = s_{i-1}$ and $c_i = 1$ and $\alpha_i < ½$

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

$$= 1 + s_i / 2 - n_i - s_i / 2 + (n_i + 1) = 2$$

Table Delete

- $\alpha_{i\text{-}1} = ¼$ (It does trigger a contraction)
  $s_i = s_{i\text{-}1}/2$; $n_i = s_{i\text{-}1}/2$; and $c_i = n_i + 1$
  $$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = (n_i + 1) + (s_i / 2 - n_i) - (s_{i-1} / 2 - n_{i-1})$$
  $$= n_i + 1 + s_i / 2 - n_i - s_i + s_i / 2 = 1$$
- $\alpha_{i\text{-}1} > ½$ ($\alpha_i \geq ½$)
  Contraction may not occur (c_i=1 ; $s_i = s_{i\text{-}1}$)
  $$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (2n_i - s_i) - (2n_{i-1} - s_{i-1})$$
  $$= 1 + 2n_i - s_i - 2(n_i + 1) + s_i = -1$$

Table Delete

- $\alpha_{i\text{-}1} = ½$ ($\alpha_i < ½$)

  Contraction may not occur

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

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (s_i/2 - n_i) - 0$$

$$= 1 + s_i/2 - n_i \quad \text{but} \quad n_{i+1} = s_i/2$$

$$= 1 + (n_i + 1) - n_i = 2$$

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 $O(n)$