A study of Dynamic Tables

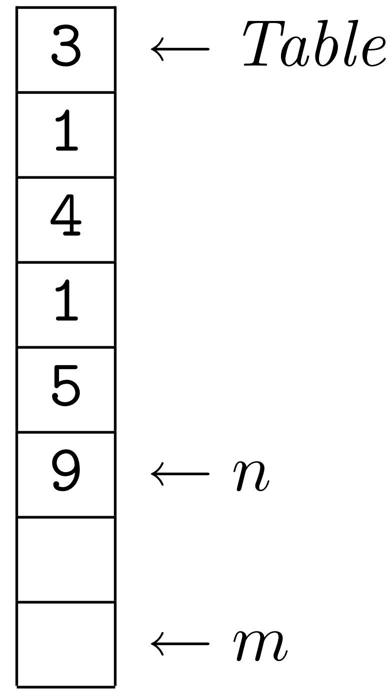
Suppose you want to implement a Table.

For now, let's assume the only operation is **Insert**.

- **Goal:** Keep the table as small as possible.
- **Problem:** Too many elements results in a overflow.
- **Idea:** Allocate more memory when necessary.

Table variables

- $Table$ — pointer to the table.
- n — number of elements, initially 0.
- m — size of table, initially 1.



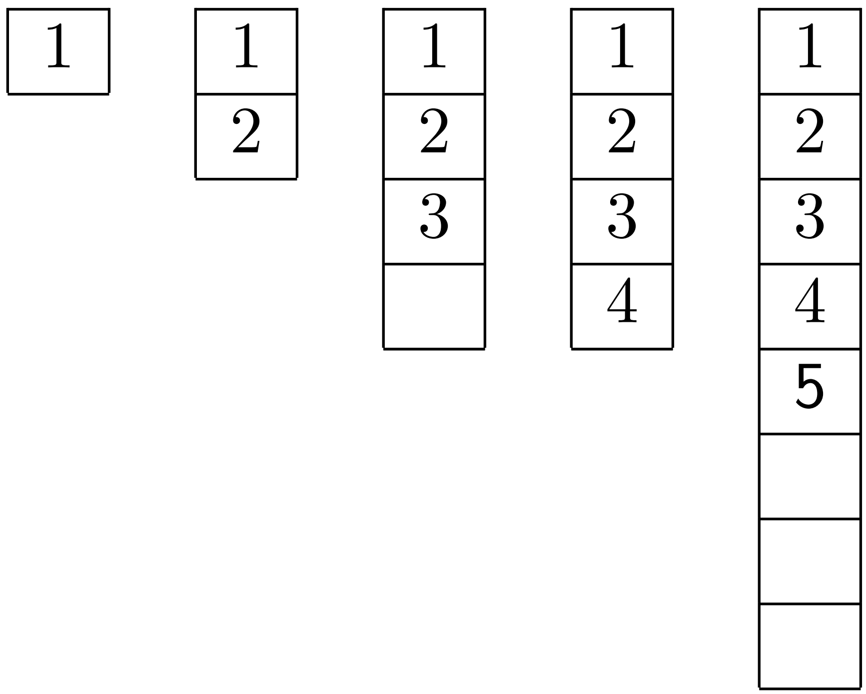
Algorithm for Insert

Insert(x)

if $n = m$
  then $newTable = malloc(2m)$
    move elements from $Table$ to $newTable$
    $Table \leftarrow newTable$
    $m \leftarrow 2m$
$Table[n] \leftarrow x$
$n \leftarrow n + 1$

How much does it cost?

$Table$:



Cost :	1	$1+$	$2+$	1	$4+$
		1	1		1

Cost analysis

Let c_i be the cost of the i:th **Insert**-operation.

$$c_i = \begin{cases} i & \text{if } i - 1 = 2^k \text{ for some } k \\ 1 & \text{otherwise} \end{cases}$$

Accounting analysis

Amortized cost for **Insert** $=$ \$3.

- Use \$1 to insert an element into $Table$.
- Save \$2 for later.

When the table grows. . .

- \$1 moves the element.
- \$1 moves an old element.

\$0	$\longrightarrow$	\$0
\$0		\$0
\$0		\$0
\$0		\$0
\$2		⋮
\$2		
\$2		
\$2		\$0
		\$2
		\$2
		⋮

Expansion and contraction

Now let us implement **Delete**:

- If the table becomes full, double its size (as before).
- Idea: if the table $<$ half full, release half the table.

  Bad idea: may be very expensive. (Why?)
- Better idea: if the table $< 1/4$ full, release half the table.
- Charge \$3 for **Insert** (as before).
- Charge \$2 for **Delete**:
  - \$1 for removing the element.
  - \$1 is stored in the empty cell, used later to move survivors.

Before

$0
$0
$0
$0
Deleted and empty cells

After 2 Delete's

$0
$0
`$1 (dead)`
`$1 (dead)`
Deleted and empty cells

Contraction $\longrightarrow$

$0
$0
Empty cells