ArrayStack: Amortized Analysis

Open Data Structures

ArrayStack

Amortized analysis of resize()

Theorem

An ArrayStack implements the List interface.

Ignoring the time spent in calls to resize(),

- get(i) and set(i, x) each run in $O(1)$ time; and
- add(i, x) and remove(i) each run in $O(1 + n - i)$ time.

ArrayStack

Amortized analysis of resize()

Theorem

An ArrayStack implements the List interface.

Ignoring the time spent in calls to resize(),

- get(i) and set(i, x) each run in $O(1)$ time; and
- add(i, x) and remove(i) each run in $O(1 + n - i)$ time.

ArrayStack

Amortized analysis of resize()

Theorem

An ArrayStack implements the List interface.

Ignoring the time spent in calls to resize(),

- get(i) and set(i, x) each run in $O(1)$ time; and
- add(i, x) and remove(i) each run in $O(1 + n - i)$ time.

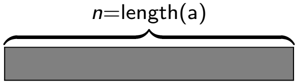
Furthermore, if we start with an empty ArrayStack and perform any sequence of m add(i, x) and remove(i) operations, then the total time spent in all calls to resize() is $O(m)$.

The Plan

resize() during add(i, x)

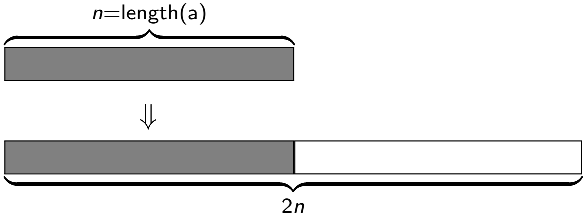
The Plan

resize() during add(i, x)



The Plan

resize() during add(i, x)

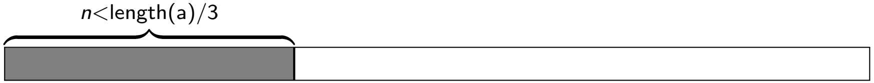


The Plan

resize() during remove(i)

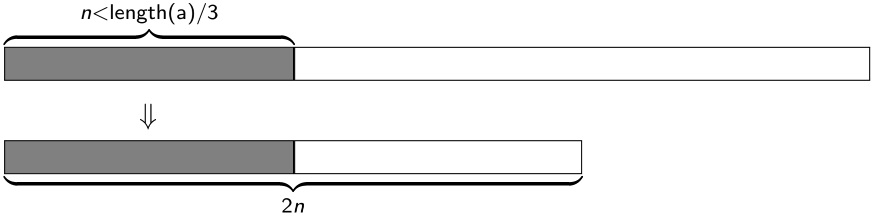
The Plan

resize() during remove(i)



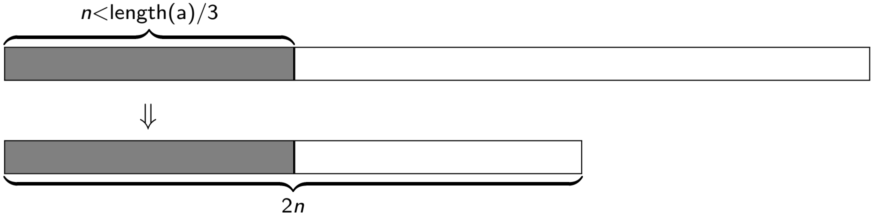
The Plan

resize() during remove(i)



The Plan

resize() during remove(i)

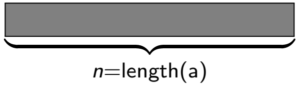


Plan

Show that the total number of items copied by resize() is at most $2m$.

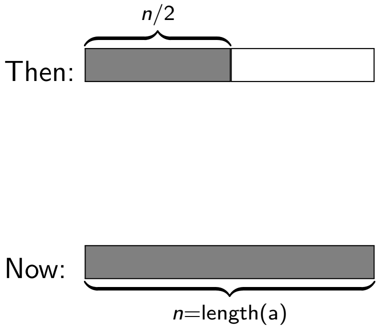
Time Between Resizes

Resize triggered by add(i, x)

Now: 

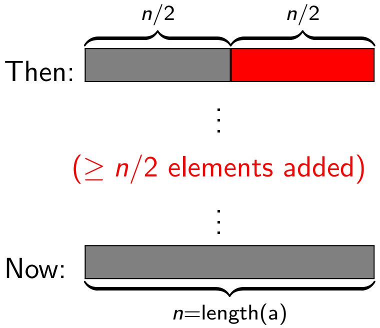
Time Between Resizes

Resize triggered by add(i, x)



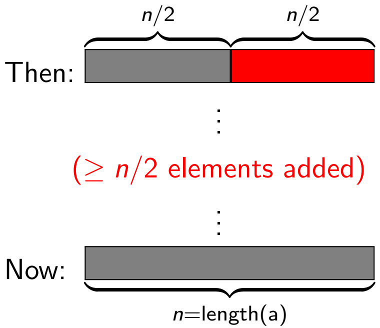
Time Between Resizes

Resize triggered by add(i, x)



Time Between Resizes

Resize triggered by add(i, x)

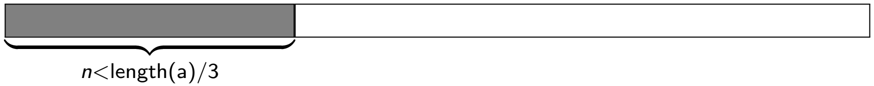


- At least $n/2$ add(i, x) operations between then and now

Time Between Resizes

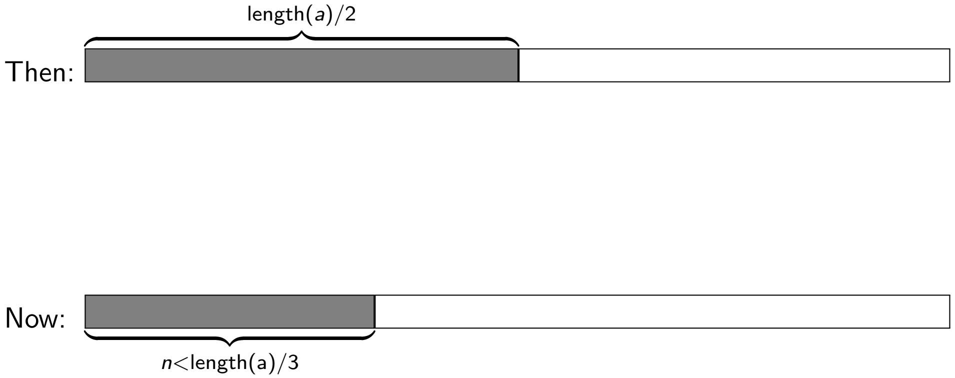
Resize triggered by remove(i)

Now:



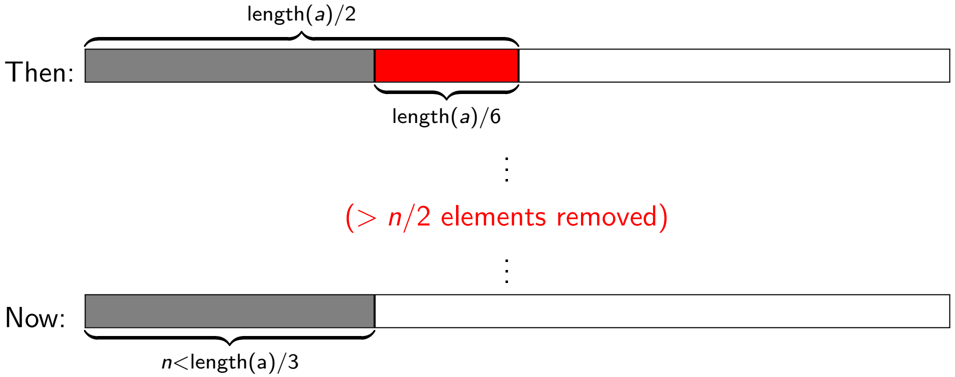
Time Between Resizes

Resize triggered by remove(i)



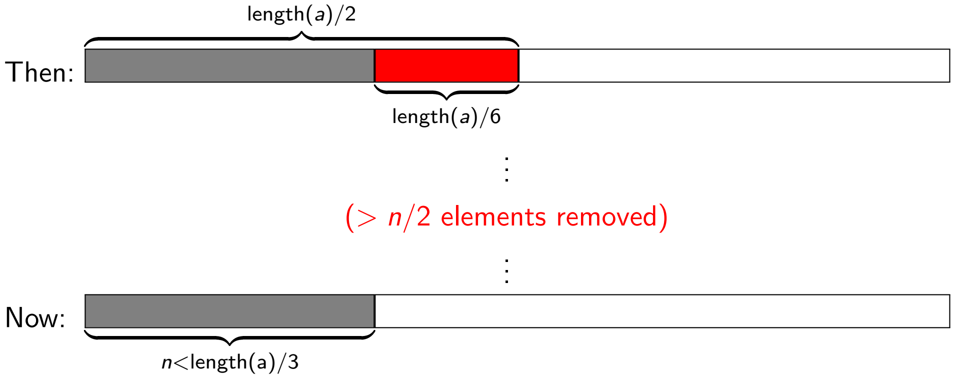
Time Between Resizes

Resize triggered by remove(i)



Time Between Resizes

Resize triggered by remove(i)



- At least $n/2$ remove(i) operations between then and now

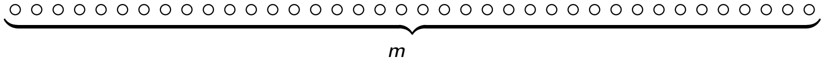
Time Between Resizes

Summary

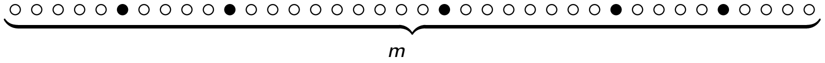
Operations Between Resizes

If a resize() operation copies n elements, then there have been at least $n/2$ add(i, x) or remove(i) operations since the preceding resize() operation.

Overview

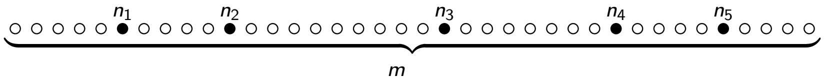


Overview



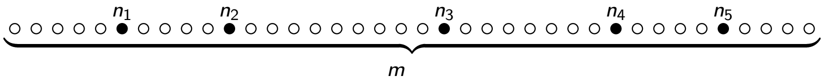
Overview

- $n_i =$ number of items copied by the i^{th} resize() operation



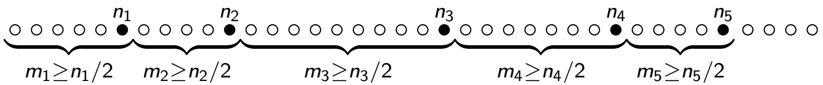
Overview

- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



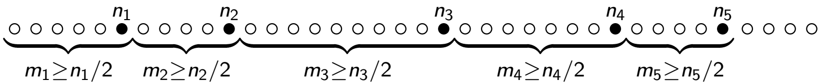
Overview

- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



Overview

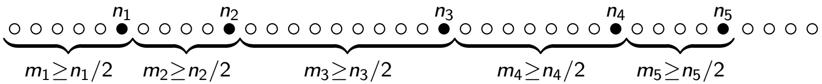
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$

Overview

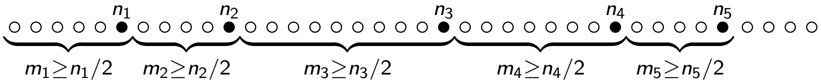
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $m \geq m_1 + m_2 + m_3 + \cdots$

Overview

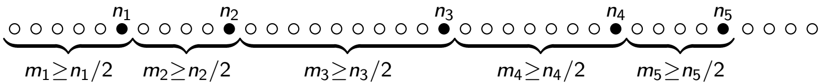
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $m \geq n_1/2 + n_2/2 + n_3/2 + \cdots$

Overview

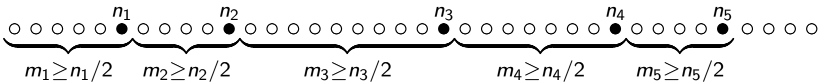
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $m \geq N/2$

Overview

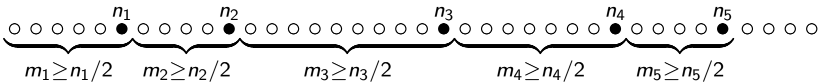
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $N \leq 2m$

Overview

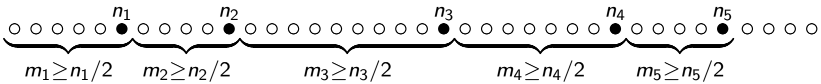
- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $N \leq 2m$
- The total number, N, of items copied by resize() is at most $2m$.

Overview

- $n_i =$ number of items copied by the i^{th} resize() operation
- $m_i =$ number of add/remove operations between the $(i-1)^{\text{th}}$ and i^{th} resize()



- Total number of items copied: $N = n_1 + n_2 + n_3 + \cdots$
- Total number of add/remove operations: $N \leq 2m$
- The total number, N, of items copied by resize() is at most $2m$.
- The total time spent in all calls to resize() is $O(m)$

ArrayStack

Summary

Theorem

An ArrayStack implements the List interface.

Ignoring the time spent in calls to resize(),

- get(i) and set(i, x) each run in $O(1)$ time; and
- add(i, x) and remove(i) each run in $O(1 + n - i)$ time.

Furthermore, if we start with an empty ArrayStack and perform any sequence of m add(i, x) and remove(i) operations, then the total time spent in all calls to resize() is $O(m)$.

End of Lesson