Today

- Amortized Analysis

Assignments

- HW-3 due fri
- Exercise 5
- Quiz 4 Thurs: resizable arrays, sequence adt, review questions

Two-Week Plan:

- Merge and Quick sort
- HW-4 out Friday (due in 1.5 weeks)
- Exam 1 next week (Thurs)
- Quiz 5 on Tuesday
Cost of Resizable Arrays via Amortized Analysis

What are the Big-O costs of the ArraySeq operations (versus LinkedSeq)?

- contains ... $O(n)$ for both
- access/update (operator[]) ... $O(1)$ (versus $O(n)$)
- erase ... $O(n)$ for both
- insert ... $O(n)$ for both

Q: What does ArraySeq insert() cost when adding items to the end?

- insert is generally more involved for resizable arrays ... why?
- We’ll call this special version of insert insert-end

(1) Note that a call to resize() is $\Theta(n)$ ... Q: Why?

(2) Most insert-end calls are $\Theta(1)$, but periodically we have a $\Theta(n)$ resize call

(3) To help determine the full cost, we use the idea of “Amortized Analysis”

- Instead of looking at just one call to insert-end
- Average the cost of a sequence of calls
- Gives the amortized cost, which averages out less-frequent expensive calls

(4) For $m$ insert-end calls, $a$ the “amortized” and $t_i$ the actual costs, we have:

$$\sum_{i=1}^{m} a \geq \sum_{i=1}^{m} t_i$$

Since the amortized cost is the same for each call, we use it to determine cost
(5) Example of using amortization for **insert-end**

- Assume a non-resize call to **insert-end** has unit cost
- Also assume array starts with 8 empty slots
- First resize call copies 8 values and then inserts at end

\[
\begin{align*}
1 + 1 + \ldots + 1 & \quad \text{1st 8 calls} \\
8 + 1 & \quad \text{9th call}
\end{align*}
\]

- For the first 9 items, we can use an amortized **insert-end** cost of 2

\[
\sum_{i=1}^{9} 2 \geq 17
\]

- Left-hand (amortized) side is 18, actual is 17
- To find the full amortized cost, need to consider for any \( n \)
(6) To simplify, assume \texttt{insert-end} starts as a 1-item array ...

- If the array becomes full on an insert, then resize (not on the next insert)
- Assume resize cost (of copying) for \( n \) insert calls is given by \( f(n) \):
  
  \[
  f(1) = 1 \quad \text{1st resize (copy one elem)} \\
  f(2) = 1 + 2 \quad \text{2nd resize (two copy steps)} \\
  f(4) = 1 + 2 + 4 \quad \text{3rd resize (three copy steps)} \\
  \ldots \\
  f(n) = 1 + 2 + 4 + \cdots + \frac{n}{2} + n = 2n - 1 \quad \text{for } n = 2^k \text{ and } k \geq 0
  \]

- Full cost includes the cost of adding \( n \) elements to the array (not just copying):
  
  \[
  \sum_{i=1}^{n} t_i = f(n) + n = 3n - 1
  \]

  - Note this is an overestimate since we don’t always have to resize:
    - e.g., for \( n = 5 \), cost is \( f(4) + 5 = 12 \) and not \( 3 \cdot 5 - 1 = 14 \)

- Using the definition of amortization wth amortization cost \( a = 3 \), we get:
  
  \[
  \sum_{i=1}^{n} 3 \geq 3n - 1
  \]

- And so, each call to \texttt{insert-end} has (an amortized) cost of \( O(1) \)
  
  - Thus, it is (amortized) constant time even with resize!
  - We’ll use a similar idea for our hash table implementation as well

\footnote{Note that \( \sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \)}