CS180 - Vectors

Announcements

- Vector HW - due next Monday
- Lab tomorrow
- Review next Thursday, test next Friday
Vector: "nice" version of array implementation.

- data

  ↑

  size

  ↑

  capacity
Runtimes

\[
\begin{align*}
\text{insert: } & \mathcal{O}(n) \\
\text{erase: } & \text{ crossed out} \\
\text{push-back: } & \text{ crossed out} \\
\text{operator []: } & \mathcal{O}(1) \\
\text{push-back } & \text{amortized runtime of } \mathcal{O}(1)
\end{align*}
\]
Analysis

Consider push-back in a vector

Running time? (worst case)

$O(n)$

making $k$ push-backs
Amortization

Every time we have to rebuild the array we get a bunch of extra spots.
Need to formalize this idea:

amortization: finding average running time per operation over a long series of operations
$n$ calls to $O(n)$ function

$O(n^2)$

$O(1) \  O(n) \  O(1) \  O(n)$

$\Rightarrow \quad n = O(n^2)$
Claim: The total time to perform a series of \( n \) push-back operations into an initially empty vector is \( O(n) \).

Proof: Think of a bank account. Each constant time operation costs $1 to run.

So each non-overflow push costs $1. Overflow inserts? $k$, where \( k=\text{size} \)
Key idea: overcharge the non-overflow pushes

\[ 3^k - 1^k = 2k \]

4k push backs

\[ \text{Overflow costs } \$2k \]
Analysis: array has $2^i$ elements in it and needs to be doubled

Last double had $2^{i-1}$ so a total of $2^{i-1}$ new things have been inserted since then.

Each gave $3$, & cost $1$.

Account: $3 \cdot 2^{i-1} - 2^{i-1} = 2 \cdot 2^{i-1} = 2^i$.

"paying" for overflow insert in this round.