CS380 - Vectors

Announcements

- HW due tomorrow

- Review in class Wed.

- Test Thursday.
Runtimes (Worst case)

- Insert: $O(n)$
- Erase: $O(n)$

Operator []: $O(1)$
Analysis

Consider push-back in a vector

Running time? (worst case)

$\Rightarrow O(n)$

Is it really that bad?

How long would $n$ push-backs take?

$n \cdot O(n) = O(n^2)$
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.
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? $\Omega(n)$ to copy
Key idea: overcharge the non-overflow pushes

\[ \text{bank account} = 0 \]

\[ k \text{ more inserts} \]

\[ 2k \]

\[ \rightarrow \text{ pay to copy } 2k \text{ elements down} \]

\[ 4k \]

\[ \text{bank account: } \sum_{i=1}^{k} 1 + \sum_{i=1}^{2k} 1 = 3k \]
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 $\$3$.

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

Bank paid for single push-back exactly the number of elements to overflow push-back.
What about n inserts?

Insert at beginning: 0(n)
Other functions:

insert:   See earlier
erase:    
push-back
Iterator