CS180 – Vectors

Note Title 10/20/2011

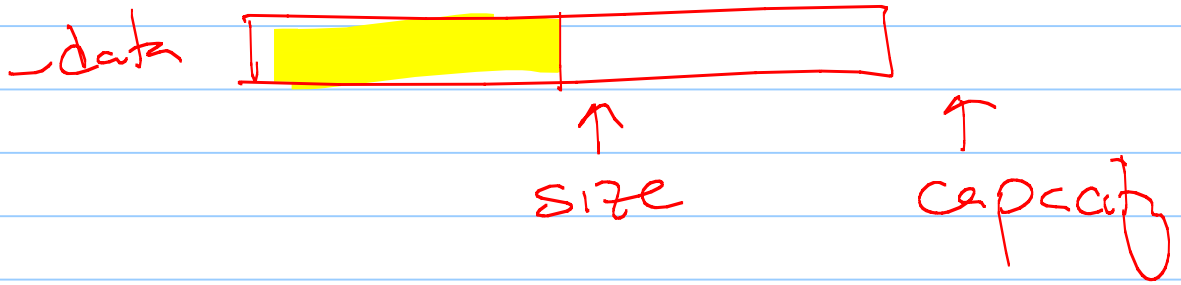
Announcements

- Vector HW – due next Monday
- Lab tomorrow
- Review next Thursday, test next Friday

Vector :

"nice" version of array

implementation:



Runtimes

insert: O(n)
erase:
push_back:

operator[]: O(1)

push_back
amortized runtime of O(1)

Analysis

Consider push_back in a vector

Running time? (worst case)

O(n)

making k push_backs

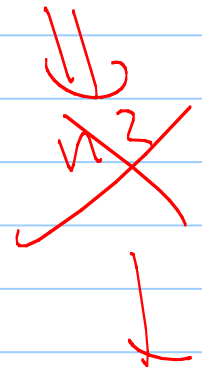
Amortization

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 operation

n calls to O(n) function



$O(n)$

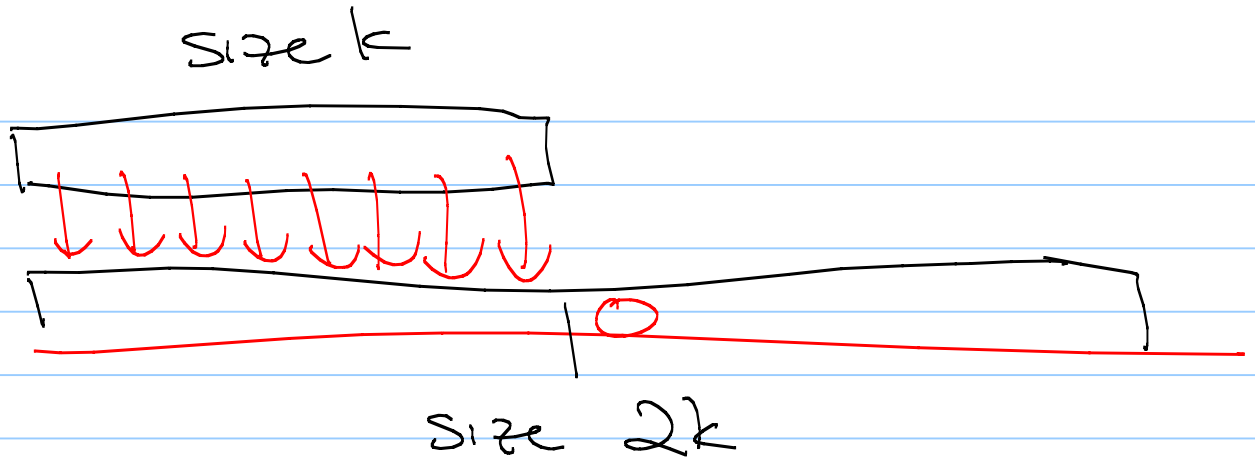
$\underbrace{O(1) \quad O(n) \quad O(1) \quad O(n)}_{n} = O(n^2)$

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.

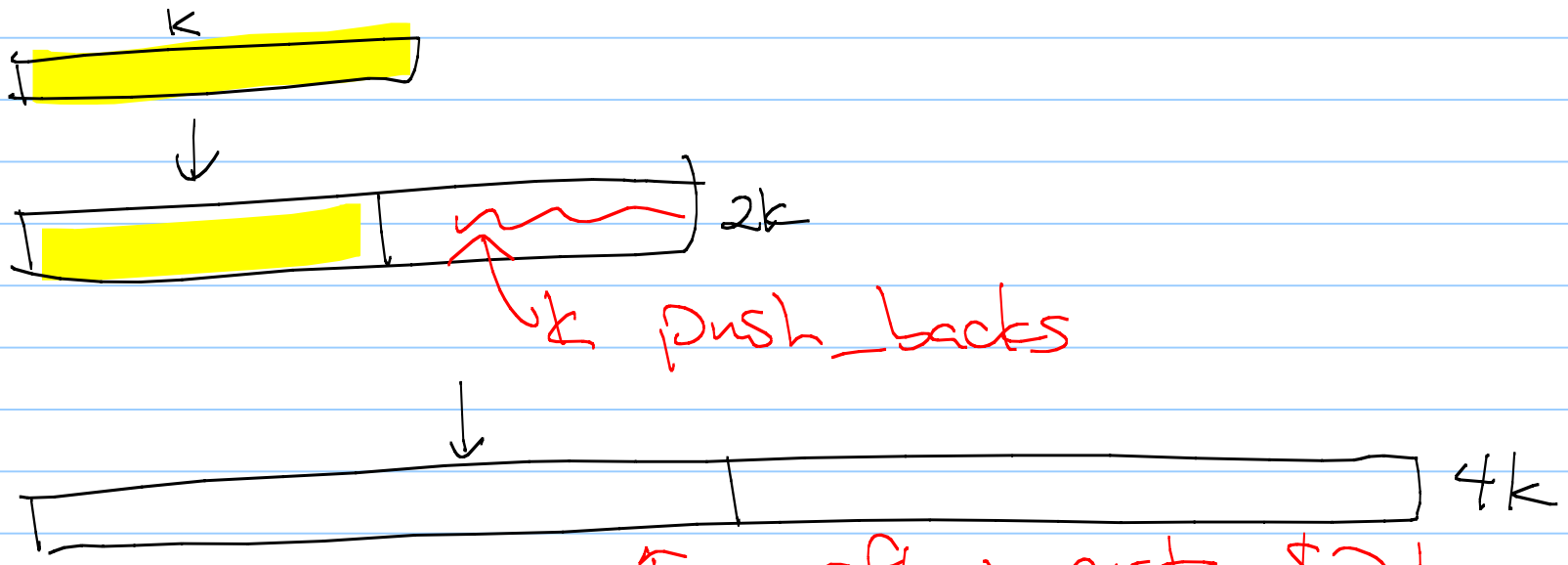
Overflow inserts? \$k, where k=size



$k+1$

10k?

Key idea: overcharge the non-overflow pushes



$3k - 1k = 2k$

Analysis: array has 2^i elements in it & 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.