# CS180 - Vectors

Note Title 10/20/2011

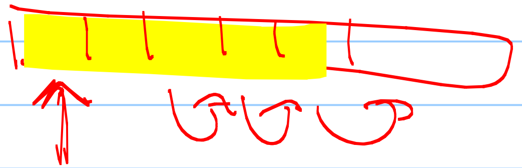
## Announcements

- HW due tomorrow
- Review in class Wed.
- Test Thursday.

## Runtimes (worst case)

Insert: O(n)

Erase: O(n)



⚠ Operator []: O(1)

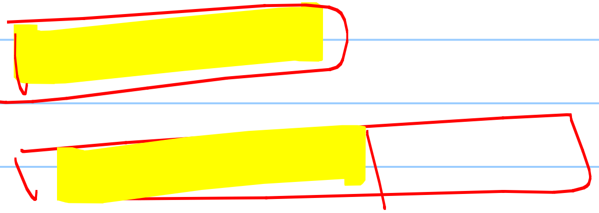
# Analyis

Consider push_back in a vector

Running time? (worst case) → 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 operation

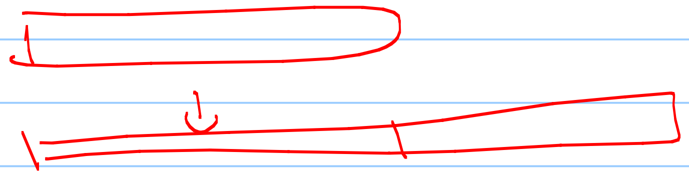
Claim: The total time to perform a series of n push_back operations into an initially empty vector is $O(n)$.

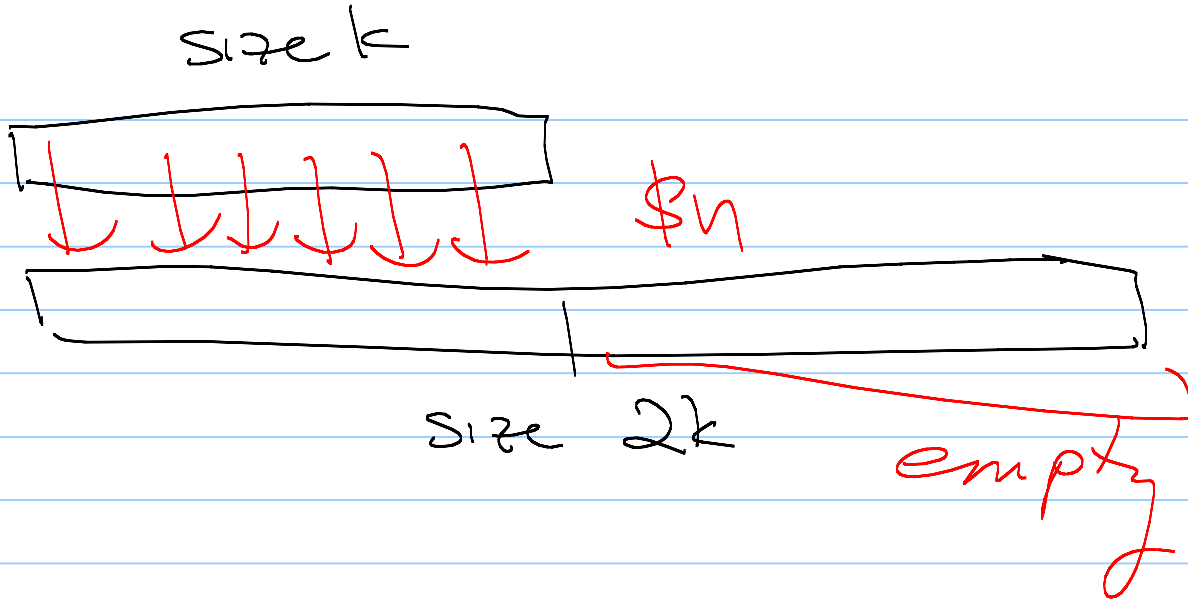
not $O(n^2)$

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

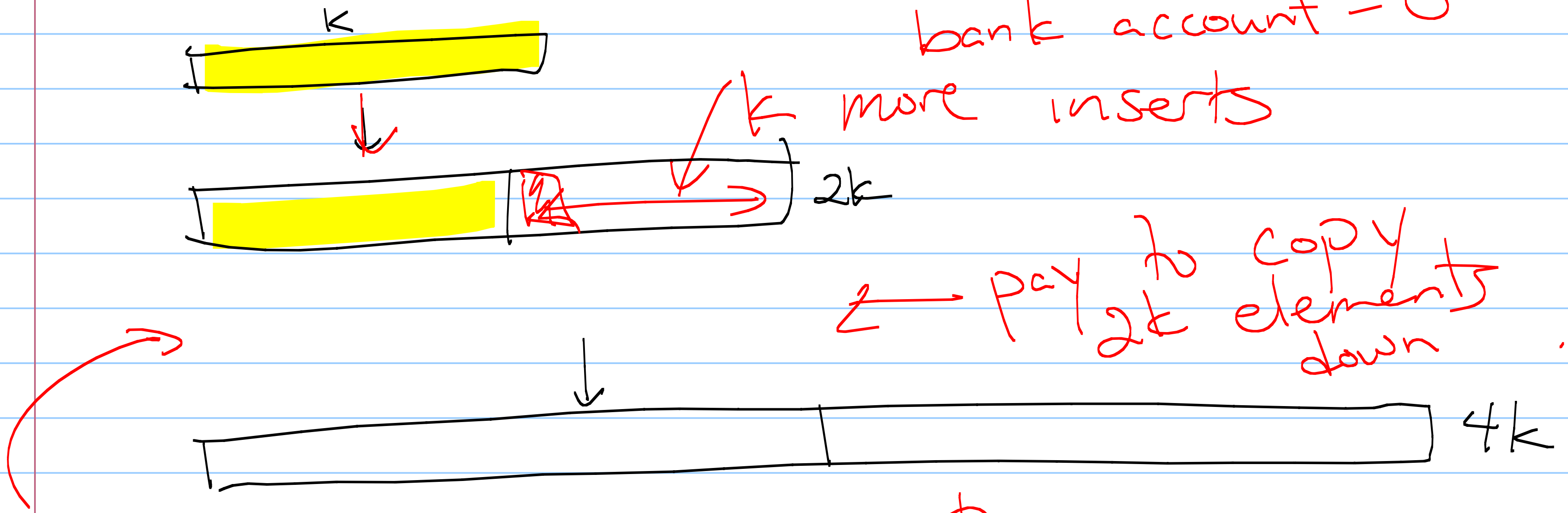
So each non-overflow push costs \$1.

Overflow inserts? \$n to copy




size k
$n
size 2k
empty

Key idea: overcharge the non-overflow pushes



bank account = 0

k more inserts

← pay to copy 2k elements down

bank account: $\underbrace{\$1 + \cdots + \$1}_{k} + \$2k = \$3k$

Analysis: array has $2^i$ elements in it & needs to be doubled

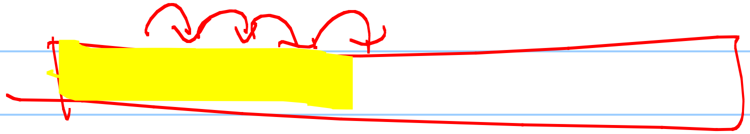
Last double had $2^{i-1}$, so a total of $2^{i-1}$ 'new' things have been inserted since then

Each gave \$3.

$$\underbrace{3 \cdot 2^{i-1}}_{\text{bank}} - \underset{\substack{\uparrow \\ \text{paid for} \\ \text{single push\_back}}}{2^{i-1}} = 2 \cdot 2^{i-1} = \underset{\substack{\uparrow \\ \text{exactly the} \\ \text{number of} \\ \text{elements in overflow} \\ \text{push\_back}}}{2^i}$$

What about n inserts?

Insert at beginning: O(n)



## ~~Other functions.~~

~~insert:~~

~~erase:~~

~~push_back~~

See earlier

# Iterator