CS180 - Vectors

Note Title 10/20/2011

Announcements

- Boeing scholarships (due Oct. 31)
- Program due tonight
- Next program posted today
- Program 1 grades emailed yesterday

Analysis

Consider push_back in a vector

Running time? (worst case)

- O(n) — have to double the array
- (if not full, O(1))

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

$\overbrace{cn + cn + cn + \cdots + cn}^{n}$

if worst case is $O(n)$, & we do n operations

$\Rightarrow$ total $O(n \cdot n) = O(n^2)$

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

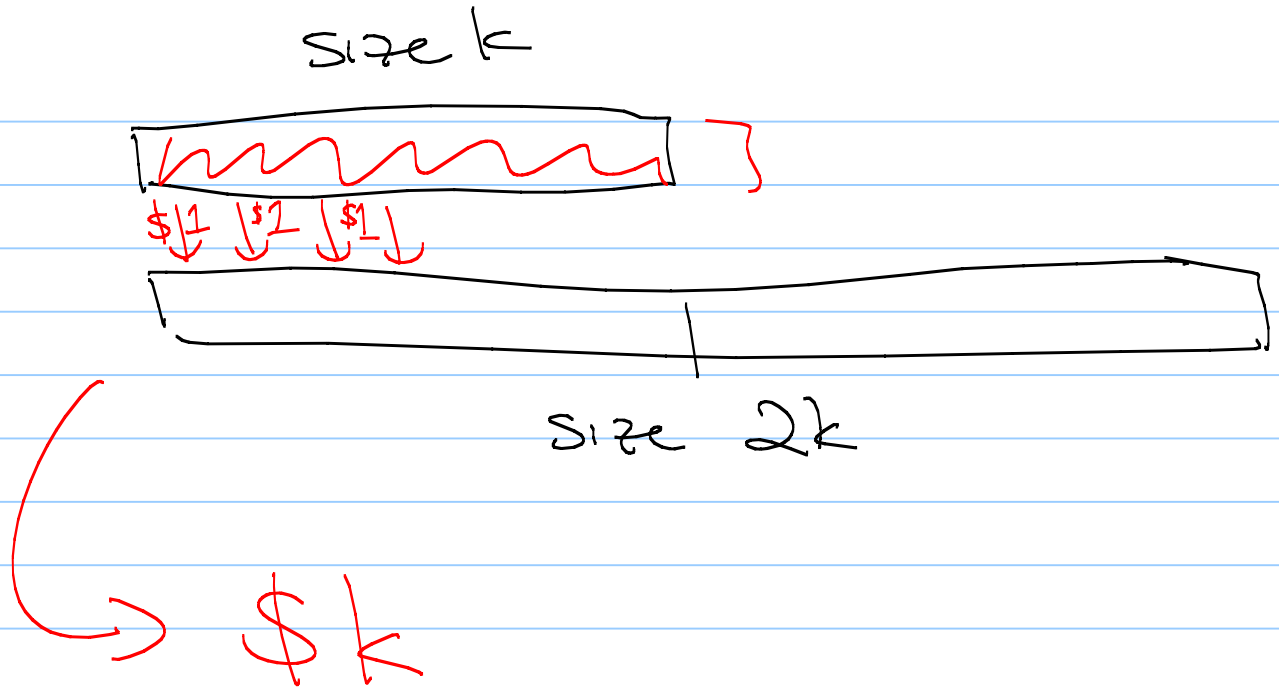
not $O(n^2)$

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

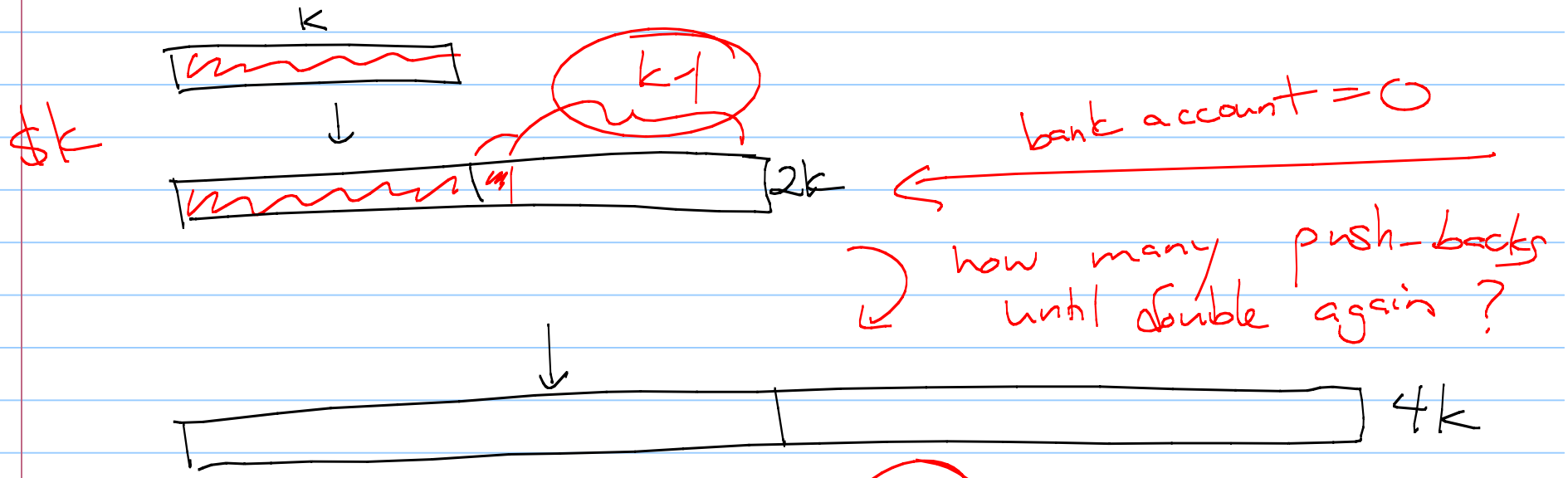
So each non-overflow push costs \$1.

Overflow inserts?

cost \$n


size k
$1 $1 $1
size 2k
$k

Key idea: overcharge the non-overflow pushes



k operations, each costs $1
at end of these k, need $2k in bank
⇒ $3 per non-overflow

$2^1 \cdot 2^i = 2^{i+1}$

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

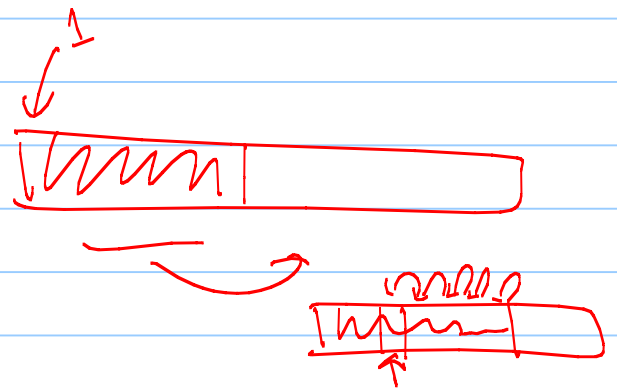
Each gave \$3.

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

↑ to pay for itself

We do k $O(1)$ time operations, then 1 $O(k)$ time.

Other functions:



- insert: O(n)
- erase: O(n)
- push_back: O(n) worst case
  - amortized: O(1)
- operator[]: O(1)