# CS2100 – Vectors

Note Title 2/15/2016

## Announcements

- HW due Wednesday (in pairs, to me)
- Lab tomorrow – due Friday (no prelab)
- Midterm 1 in 1 week
  - review on Friday
    → bring questions!

## HW 3 Recap:

- be careful w/ pointers & templates
- don't alter your private data!

```
Object minimum() const {
    SNode* temp = _head;
    Object minsofar;
    while (temp != NULL) {
        if (temp->_data < minsofar)
            minsofar = temp->_data;
        temp = temp->_next;
    }
} return minsofar;
```

Can you return const Object& ?

Can't return variable created in function!

<u>But:</u>

```
                              {
SNode* temp = _head;
SNode* minsofar = _head;
while (not Null) {
   if (minsofar->_data > temp->_data)
      minsofar = temp;
   temp = temp->_next;
 }
return minsofar->_data;
}
```

Last time:

Vector running times:

Linear time: $O(n)$

(except for size, empty + other $O(1)$ - time fcns)

**But:** Is it really that bad?

Consider a sequence of push_back operations.

Runtime: n push_backs, each takes $O(n)$ time

$\Longrightarrow O(n^2)$

Really: $\sum_{i=1}^{n} i = O(n^2)$

**But:**

When do we actually double?

Only take linear time when doubling, & after doubling, half empty!

## 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 $O(n)$.

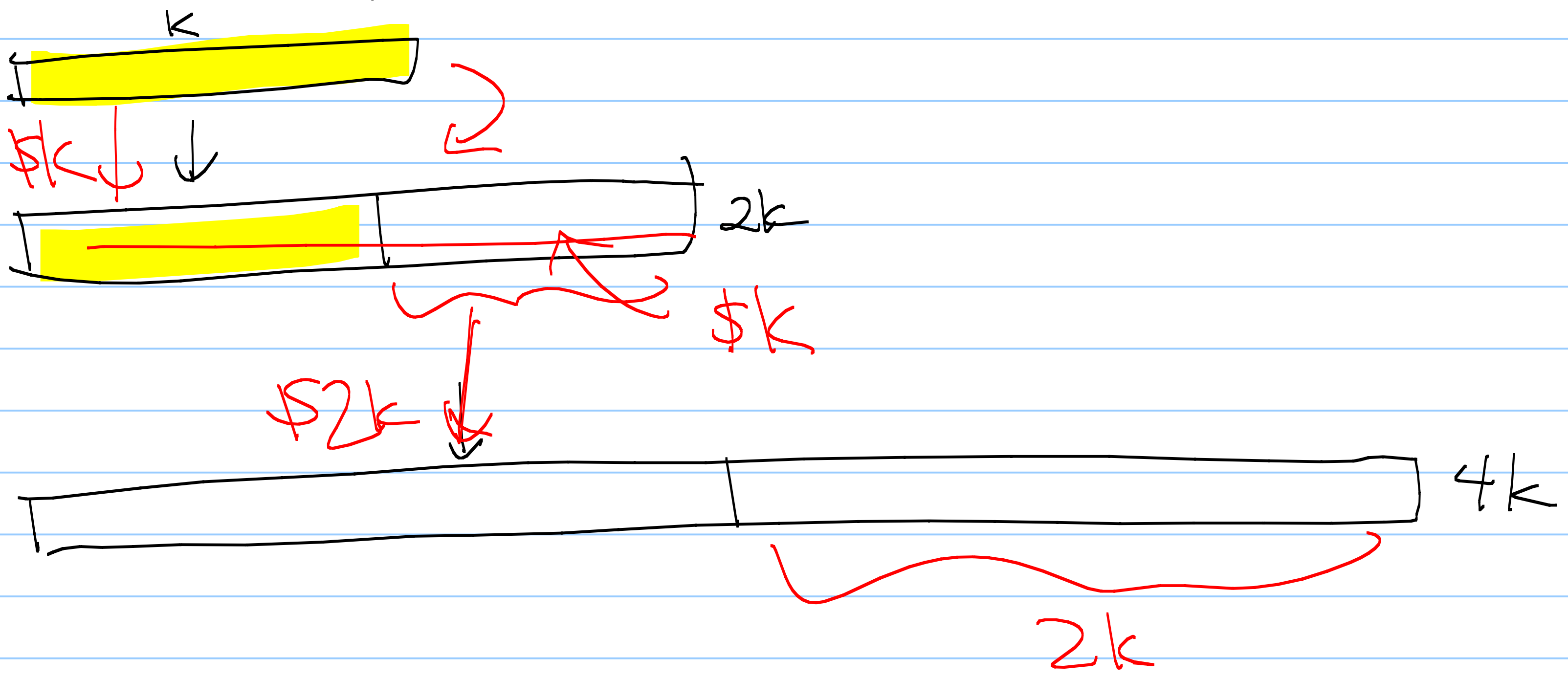
proof: Think of a bank account. Each constant time operation costs \$1 to run.

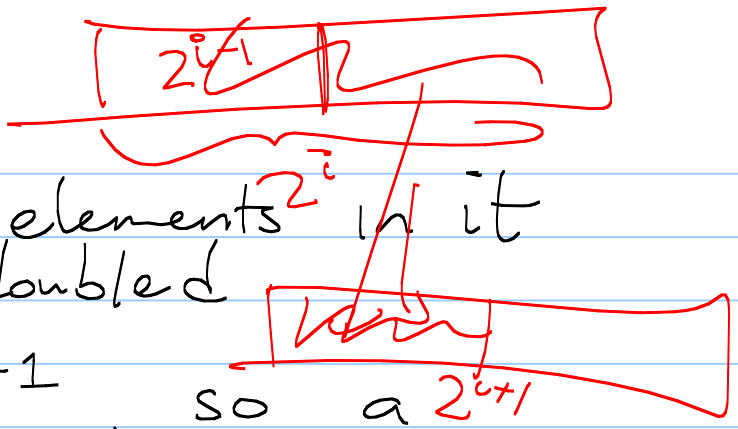
So each non-overflow push costs \$1.

Overflow inserts?

$1 + \$n$

Key idea: overcharge the non-overflow pushes





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 $\$1$

$\$\,3 \cdot 2^{i-1}$ in bank

$-2^{i-1}$

$= 2 \cdot 2^{i-1}$ left in bank

$= 2^i$

"enough" in bank to cover doubling.

## Vector class :

What's left : Housekeeping

~20 other functions

$\hookrightarrow$ HW next week