Recap:

- Lab tomorrow, due Sunday
- Reading due Fri. by 2pm
- HW due Saturday
- Review session Monday
- Exam on Wednesday

Also: Blackboard + gradebook

go check!

#2, pt 1
+ first 5 Zybook labs

Last time

Vector running times:

Accessing: operator[] : $O(1)$

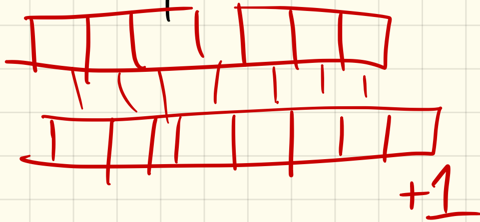
myvec[i]

Insert/remove: $O(n)$

But: Is it really that bad?

Related: Why did I double the size of the array?

if full



Consider a sequence of push_back operations.

Runtime: worst case:

n push_back into initially empty vector

$O(n) \times n = O(n^2)$

But:

When do we actually double?

only when full.

"~~average~~ amortized" analysis

Amortization:

Every time we rebuild the array, we have free space.

Formalize: find average running time per operation over a long sequence of operations

Claim: Total time to perform n push_backs to an initially empty vector ~~$O(n)$~~ $\Rightarrow$ time per push is $O(1)$

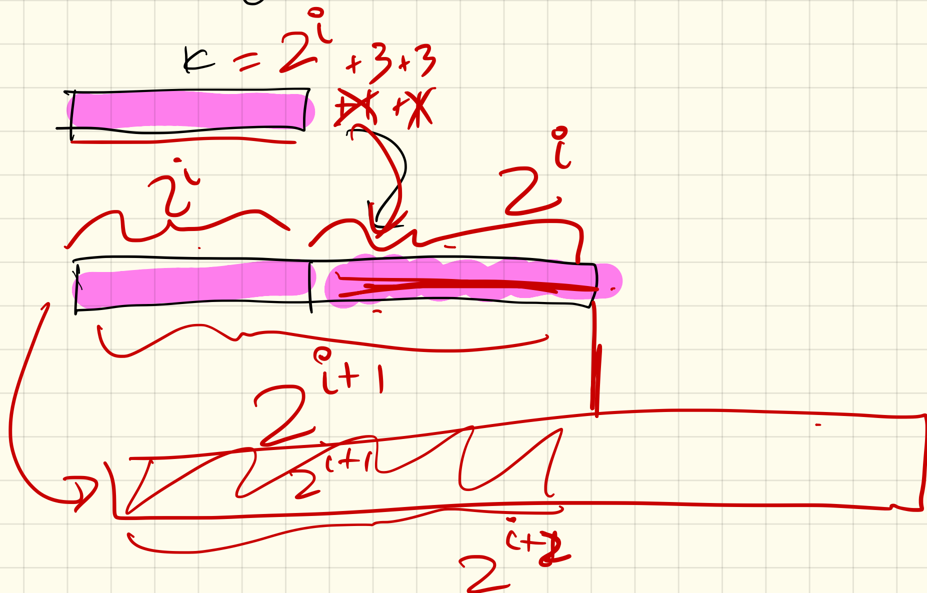
pf: bank account analogy:

Each $O(1)$ operation costs \$1.

Slightly overcharge the fast push operations.

↳ charge \$3

So: overcharge the non-overflow ones:



Analysis: array has 2^i elements & gets doubled.

Last double: takes 2^i copy operations

Charge each push_back:

$$2^i \cdot (3-1) = 2^i \cdot 2$$

precisely the cost to copy to 2^{i+2} array

Takeaway:

n push backs:

split into powers of 2

$2^0 + 2^1 \ldots$

$| 2^2 \ldots | 2^3$

$2^i \ldots 2^{i+1}$

Last parts:

- House keeping ✓
- push_back vs. insert:

```
void push_back(const Object& element){
    if(_size < _capacity){
        _data[size] = element;
        _size++;
    }
    else {
        // copy doubling code
    }
}
```

Next HW:

write 4-5 vector functions

add to Zybooks

due 1 week after midterm