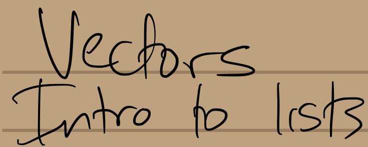
CS2100



Recap

- MT back next week
- HW 1-3 are graded
  Blackboard set up
- HW 5 is posted
  due next Friday
  over Vectors

Last time

Vector running times

- size + empty: $O(1)$
- all others: $O(n)$

except operator [ ], $O(1)$

But: Is it really that bad?

When overflow, we double size

→ bunch of empty spots

Consider a sequence of push_back operations. n of them

Runtime:

- do n operations
- push_back in worst case is $O(n)$

$\Rightarrow O(n^2)$

But:

When do we actually double?

only when we double

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 is $O(n)$.

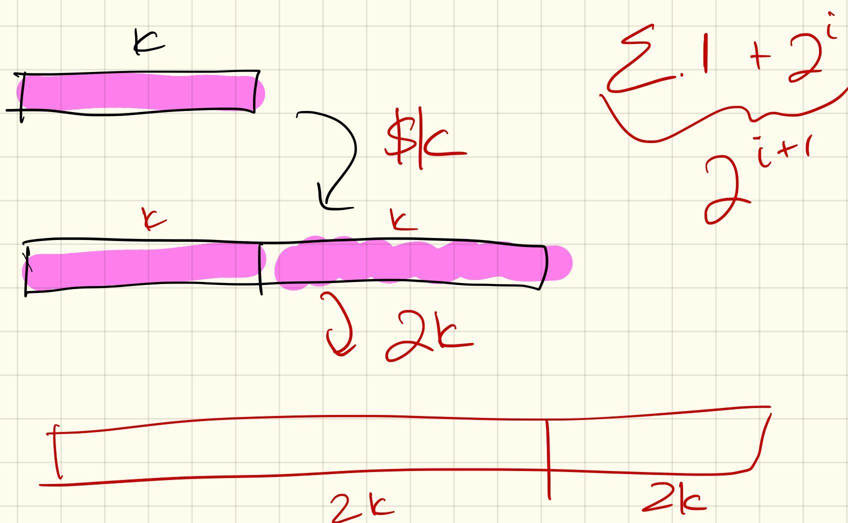
pf: bank account analogy:

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

So each non-~~push overflow~~ overflow push_back costs \$1.

Overflow ones? \$n

So: overcharge the non-overflow ones:



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

Last double: 2^{i-1}

Charge each push_back: \$3

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

So: $O(n)$ total, since $O(1)$ operations = slow one

Vectors: testing

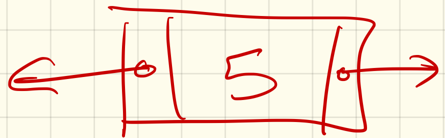
Lists: Motivation

Insert in vectors is slow!
If I'm changing 1 thing, want O(1).

Doubly linked List struct:

```
struct Node {
  T _data;
  Node* _prev;
  Node* _next;

}
private:
  Node* head;
```

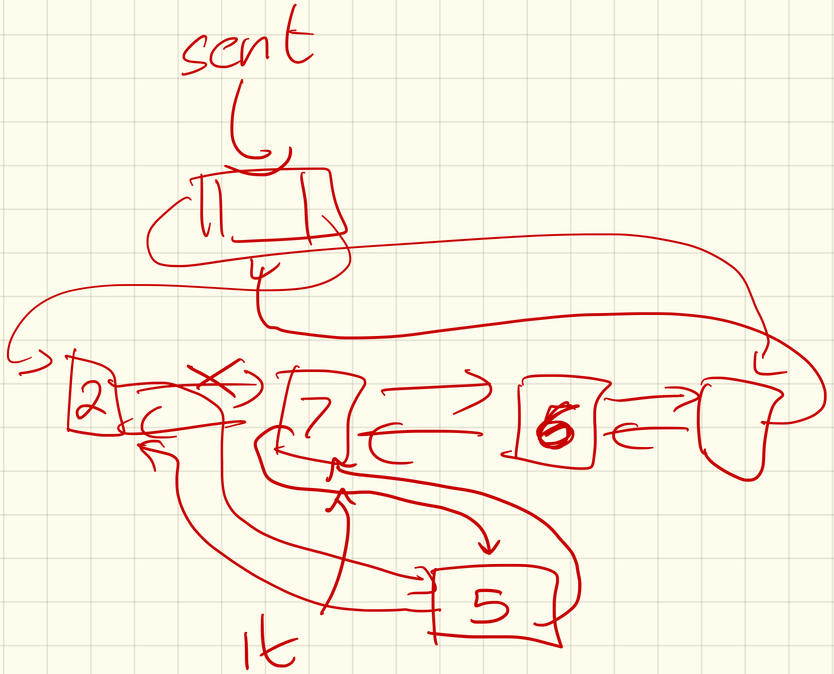


Circularly linked lists:

private data:

Node* _sent

_size



insert(5, it)

↳ allocate node

4 ptr updates

Iterator: inside List,

```
class Iterator {
    Node* _current;

T& operator*() {
    return _current->_data;
}
```

in main:

```
Iterator it;
*it
```