SECTION 7
Advanced data structures
SECTION 7.1
Aggregate analysis
You’ve heard the moral: “slow and steady wins the race.”
My version: “whoever finishes the race fastest wins the race.”
Running time of each operation, in a run of Dijkstra’s algorithm

Don’t worry about the worst-case cost of each individual operation.

Worry about the worst-case aggregate cost of a \textit{sequence} of operations.

\[ \text{total time} = O(V) \times c_{\text{popmin}} + O(E) \times c_{\text{push/dec.key}} \]
Advanced data structures involve a clever design tradeoff, to make sequences of operations cheaper:

❖ individual operations are usually cheap, but occasionally expensive

❖ the worst-case aggregate cost of a sequence of $m$ operations is cheaper than $m$ times the worst-case of a single operation

**DOUBLY-LINKED LIST**

- `x = DoublyLinkedList(...)` $n$ items
- `x[i]` $\Theta(n)$
- `x.append(·)` $\Theta(1)$

**PYTHON LIST**

- `x = [...]` $n$ items
- `x[i]` $\Theta(1)$
- `x.append(·)` $\Theta(1)$ usually, sometimes $O(n)$

\[\text{but } m \times \text{append(·)} \text{ is ALWAYS } m \times \Theta(1)\]
To design advanced data structures, we need to be able to reason about aggregate costs. How?

❖ Just be clever and work hard
❖ Use an accounting trick called *amortized costs*

\[
x.\text{append(·)} \quad \mathcal{O}(1) \text{ usually, sometimes } \mathcal{O}(n)
\]

but \( m \times \text{append(·)} \) is ALWAYS \( m \times \mathcal{O}(1) \)
SECTION 7.2, 7.3
Amortized costs
class MinList<T>:
    def append(T value):
        # append a new value
    def flush():
        # empty the list
    def foreach(f):
        # do f(x) for each item
    def T min():
        # return the smallest
        # (without removing it)

Stage 0
- Use a linked list
- \texttt{min} iterates over the entire list

Stage 1
- Use a linked list
- \texttt{min} caches its result, so that next time it only needs to iterate over newer values

Stage 2
- Use a linked list
- Store the current minimum, and update it on every append

Stage 3
- \texttt{min} caches its result, the same as Stage 1
- ... but we argue it’s just as good as Stage 2