Amortized Analysis

We'll start with an example:
expansion of dynamic arrays.

Dynamic array has
- size - allocated capacity
- num - number of elements currently in the array

What is an optimal strategy for expanding a dynamic array?

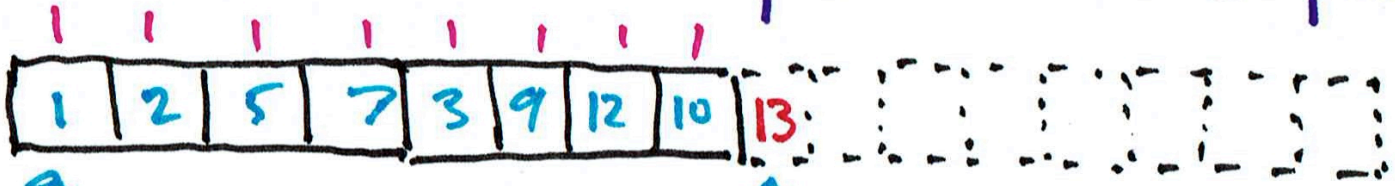
When an insertion would overflow the allocated array, we should

1. Allocate an array of size 2·size.
2. Copy the elements to the new array.
3. Insert the element into the new array.

Why is this optimal?

First way to think about it...

With every insertion, "bank" some time credits to spend on expansion.

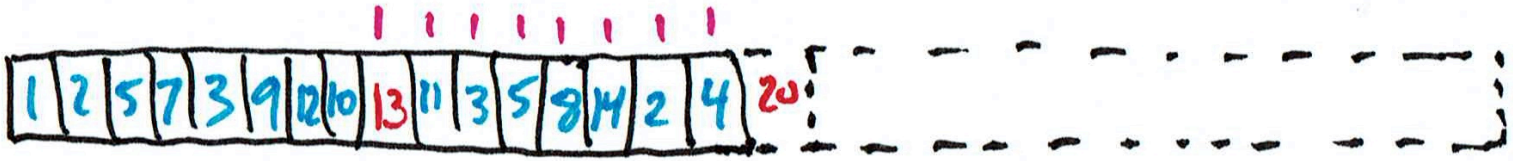


insertion costs 1 time unit; bank 1 additional charge 2 per insertion!

spend my credits to copy original data to expanded array

So far, so good. I can cover the cost of expansion with stored credits.

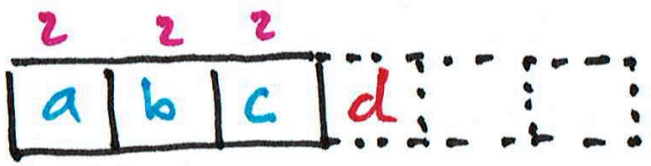
Keep going...



Problem! I only have half the credits I need.

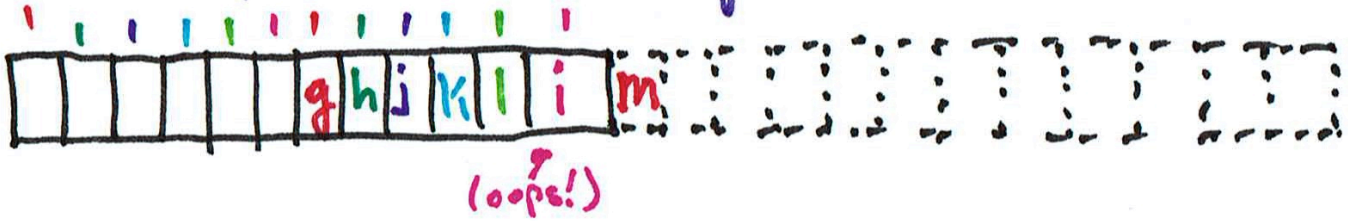
We should have charged 3 time units per insertion and banked 2.

For the first expansion we will have overcharged, but we'll be good after that.



Six credits. More than what we need. Discard extra.

Six credits. Enough to cover copy.
Spend all my credit.



So what?

If we think of insertion as costing three time units, we can cover all future expansion costs with credits. Expansion costs nothing!

Since "3" is a constant, insertion is still O(1) irrespective of how many expansions are required.

Another way to say this ...

Let n be size. The time to perform an expansion is $O(n)$.

However, I must have done n insertions if an expansion is necessary.

Divide the cost of expansion by the number of insertions:

$$O(n)/n = O(1)$$