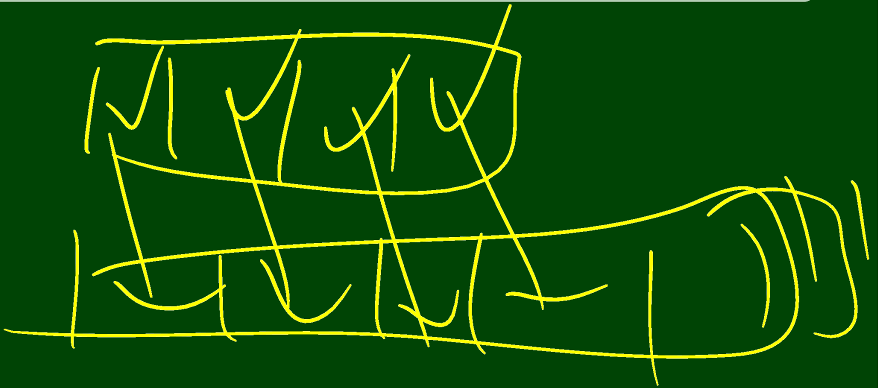
Analyzing `push` for an `ArrayStack`

Best Case

Worst Case



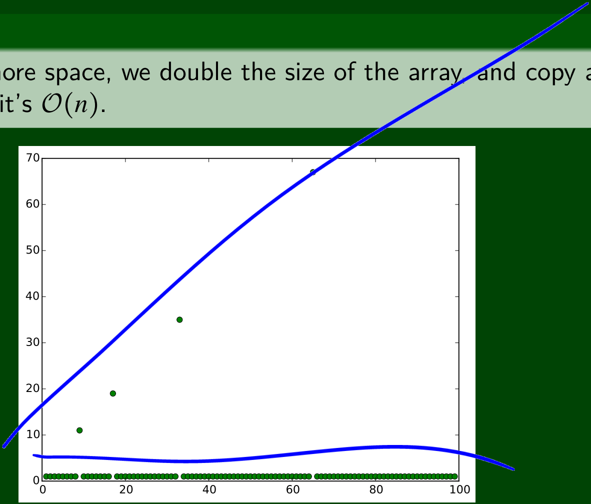
Analyzing `push` for an `ArrayStack`

Best Case

There's more space in the underlying array! Then, it's $\Omega(1)$.

Worst Case

If there's no more space, we double the size of the array, and copy all the elements. So, it's $\mathcal{O}(n)$.



Insight: Our analysis seems wrong. Saying linear time feels wrong.

Analyzing `push` for an `ArrayStack`

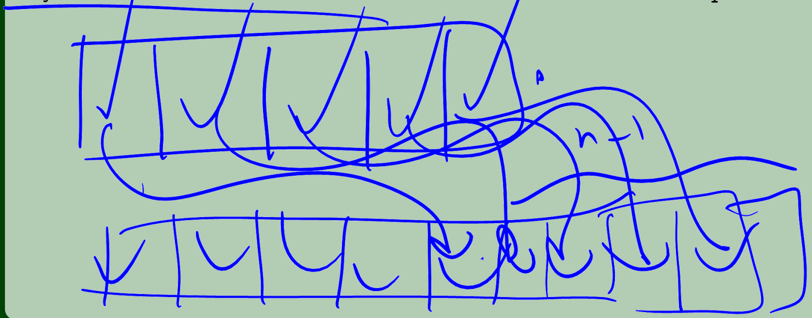
This is where “amortized analysis” comes in. Sometimes, we have a **very rare** expensive operation that we can “charge” to other operations.

Intuition: Rent, Tuition

You pay one big sum for a long period of time, but you can afford it because it happens very rarely.

Back to `ArrayStack`

Say we have a full `Stack` of size n. Then, consider the next n `pushes`:



Analyzing `push` for an `ArrayStack`

This is where "amortized analysis" comes in. Sometimes, we have a **very rare** expensive operation that we can "charge" to other operations.

Intuition: Rent, Tuition

You pay one big sum for a long period of time, but you can afford it because it happens very rarely.

Back to `ArrayStack`

Say we have a full `Stack` of size n. Then, consider the next n `push`es:

- The next `push` will take $\mathcal{O}(n)$ (to resize the array to size $2n$)
- The $n-1$ operations after that will all be $\mathcal{O}(1)$, because we know we have enough space

$$\frac{n + (n-1)\cdot 1}{n} = O(1)$$

What happens if we change our resize rule to each of the following:

- $n \to n+1$
- $n \to \frac{3n}{2}$
- $n \to 5n$