Q1. What on earth does this maths mean? –
\[ O(L) - L = O(1) \]
Q2. How do we come up with potential functions?
EXAMPLE: DYNAMIC ARRAY

A Python list is implemented as a dynamically-sized arrays. It starts with capacity 1, and doubles its capacity whenever it becomes full.

Suppose the cost of writing an item is \( O(1) \), and the cost of doubling capacity from \( m \) to \( 2m \) (and copying across the existing items) is \( O(m) \).

Show that the amortized cost of append is \( O(1) \).

\[
\begin{align*}
\Delta \Phi &= \Phi \text{after} - \Phi \text{before} \\
&= (1 - \frac{\epsilon}{2}) \epsilon
\end{align*}
\]

am. cost = cost + \( \Delta \Phi \)
\[
\leq k_1 n + k_2 + (1 - \frac{\epsilon}{2}) \epsilon
\]
\[
= n \left( k_1 - \frac{\epsilon}{2} \right) + k_2 + \epsilon.
\]

Let's set \( \epsilon = 2k_1 \). Then

am. cost \( \leq k_2 + 2k_1 \cdot = O(1) \).
We should design our potential function to pay for “unbounded” costs.

**FibHeap decreasekey**

The true cost is $O(L)$. Can we make decreasekey be $O(1)$? We’d need $\Delta \Phi = -L$.

IDEA: put credit on each loser, and release the credit when loser is moved to root.

So: $\Phi = \text{const} \times \#\text{losers}.$

**FibHeap popmin / cleanup**

The true cost is $O(M + \log N)$, $N = \#\text{items in heap}$.

Can we make popmin be $O(\log N)$? Can we get $\Delta \Phi = -M$?

THINK: put credit on each tree/root. Release the credit when the tree is made a child.

So: $\Phi = \text{const} \times \#\text{trees}.$

Finally: look meticulously at all the operations to see if there’s a tradeoff between the two constants.