Competitive Algorithms: Searching for Keys

You live at position 0 of the PA turnpike. You’ve dropped your keys somewhere (at an integer location) and need to find them. The keys are small, so you cannot see them from afar, you will only see them when you are at the same location as them. If the keys are at location $X \in \mathbb{R}$, then the optimal solution is just to go to that location, and travel $|X|$. If you knew the sign of $X$, you could do that. But you don’t.

What should you do to minimize the distance traveled? Your algorithm should have constant competitive ratio, ideally 9.

Competitive Algorithms: Examples for Non-MTF Algorithms

During lecture, we covered several solutions to the List Update problem with competitive ratio $\Omega(n)$

- **Do Nothing**: Don’t reorder the list.
- **Single Exchange**: After accessing $x$, if $x$ is not at the front of the list, swap it with its neighbor toward the front.

Now we will be covering an alternative algorithm and proving the same competitive ratio

- **Frequency Count**: Maintain a frequency of access for each item. Keep the list ordered by non-increasing frequency from front to back.

Give request sequences for the List Update problem where the following algorithms have competitive ratio $\Omega(n)$. 
Counting with suffix trees

Given a string $S$ of length $n$, for each $1 \leq i \leq n$, we want to find the number of strings that occur as substrings of $S$ exactly $i$ times. How fast can you do this? (Hint: suffix trees)