1. Simulating Queues with Stacks

A queue is a first-in-first-out data structure. It supports two operations push and pop. Push adds a new item to the back of the queue, while pop removes the first item from the front of the queue. A stack is a last-in-first-out data structure. It also supports push and pop. As with a queue, push adds a new item to the back of the queue. However, pop removes the last item from the back of the queue (the one most recently added).

Show how you can simulate a queue by using two stacks. Any sequence of pushes and pops should run in amortized constant time.

2. Multistacks

A multistack consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the $i$th stack $S_i$ can hold up to $3^i$ elements. Whenever a user attempts to push an element onto any full stack $S_i$, we first move all the elements in $S_i$ to stack $S_{i+1}$ to make room. But if $S_{i+1}$ is already full, we first move all its members to $S_{i+2}$, and so on. To clarify, a user can only push elements onto $S_0$. All other pushes and pops happen in order to make space to push onto $S_0$. Moving a single element from one stack to the next takes $O(1)$ time.

![Figure 1. Making room for one new element in a multistack.](image)

(a) In the worst case, how long does it take to push one more element onto a multistack containing $n$ elements?

(b) Prove that the amortized cost of a push operation is $O(\log n)$, where $n$ is the maximum number of elements in the multistack.

3. Powerhungry function costs

A sequence of $n$ operations is performed on a data structure. The $i$th operation costs $i$ if $i$ is an exact power of 2, and 1 otherwise. Determine the amortized cost of the operation.