✖

Dynamic Arrays and Amortized Analysis

✓ **4/4** points earned (100%)

Quiz passed!

Continue Course

Back to Week 1

✓   1 / 1
        points

1.
Let's imagine we add support to our dynamic array for a new operation PopBack (which removes the last element), and that PopBack never reallocates the associated dynamically-allocated array. Calling PopBack on an empty dynamic array is an error.

If we have a sequence of 48 operations on an empty dynamic array: 24 PushBack and 24 PopBack (not necessarily in that order), we clearly end with a size of 0.

What are the minimum and maximum possible final capacities given such a sequence of 48 operations on an empty dynamic array? Assume that PushBack doubles the capacity, if necessary, as in lecture.

◉ minimum: 1, maximum: 32

Correct Response

The minimum is achieved when we alternate with one PushBack followed by one PopBack. The size of the array never exceeds 1, so the capacity also never exceeds 1.

The maximum is achieved when we have 24 PushBacks followed by 24 PopBacks. The maximum size is 24, so the corresponding capacity is 32 (next highest power-of-two).

○   minimum: 1, maximum: 1

○   minimum: 32, maximum: 32

○   minimum: 1, maximum: 24

○   minimum: 24, maximum: 24

✔   1 / 1
     points

2.

Let's imagine we add support to our dynamic array for a new operation PopBack (which removes the last element). PopBack will reallocate the dynamically-allocated array if the size is $\leq$ the capacity / 2 to a new array of half the capacity. So, for example, if, before a PopBack the size were 5 and the capacity were 8, then after the PopBack, the size would be 4 and the capacity would be 4.

Give an example of n operations that require $O(n^2)$ copies.

◉   Let n be a power of 2. Add $n/2$ elements, then alternate $n/4$ times between doing a PushBack of an element and a PopBack.

Correct Response
Once we have added $n/2$ elements, the dynamically-allocated array is full (size=$n/2$, capacity=$n/2$). When we add one element, we resize, and copy $n/2$ elements (now: size = $n/2 + 1$, capacity= n) When we then remove an element (with PopBack), we reallocate the dynamically allocated array and copy $n/2$ elements. So, each of the final $n/2$ operations costs $n/2$ copies, for a total of $n^2/4$ moves, or $O(n^2)$

○   PushBack $n/2$ elements, and then PopBack $n/2$ elements.

○   PushBack 2 elements, and then alternate $n/2 - 1$ PushBack
      and PopBack operations.

✔   1 / 1
            points

3.
Let's imagine we add support to our dynamic array for a new operation
PopBack (which removes the last element). Calling PopBack on an empty
dynamic array is an error.

PopBack reallocates the dynamically-allocated array to a new array of half
the capacity if the size is ≤ the capacity / 4 . So, for example, if, before a
PopBack the size were 5 and the capacity were 8, then after the PopBack,
the size would be 4 and the capacity would be 8. Only after two more
PopBack when the size went down to 2 would the capacity go down to 4.

We want to consider the worst-case sequence of any n PushBack and
PopBack operations, starting with an empty dynamic array.

What potential function would work best to show an amortized $O(1)$ cost
per operation?

○ $\Phi(h) = 2$

○ $\Phi(h) = 2 \times size - capacity$

◉ $\Phi(h) = max(2 \times size - capacity, capacity/2 - size)$

Correct Response
This is a valid potential function since:

- When we start, size=capacity=0, so $\Phi(h_0) = 0$

- $\Phi(h_t) \geq 0$ since, if size > capacity/2, the first term of the max is
  non-negative, and if size ≤ capacity/2, the second term of the
  max is non-negative.

The analysis of PushBack remains just as in lecture. The question is what happens when we do a PopBack. Amortized cost = true cost + $\Phi(h_t) - \Phi(h_{t-1})$

- no resize needed: true cost = 1. If size > capacity/2, the change in $\Phi = \Phi(h_t) - \Phi(h_{t-1}) = 2$. If size $\leq$ capacity/2, the change in $\Phi = 1$. Max total amortized cost is 3.

- Resize needed: true cost = capacity/4 + 1. $\Phi(h_t) = 0, \Phi(h_{t-1})$ = capacity/2 - capacity/4 = capacity/4. Total amortized cost = capacity/4 + 1 - capacity/4 = 1.

○ $\Phi(h) = max(0, 2 \times size - capacity)$

✔   1 / 1
points

4.

Imagine a stack with a new operation: PopMany which takes a parameter, i, that specifies how many elements to pop from the stack. The cost of this operation is i, the number of elements that need to be popped.

Without this new operation, the amortized cost of any operation in a sequence of stack operations (Push, Pop, Top) is $O(1)$ since the true cost of each operation is $O(1)$.

What is the amortized cost of any operation in a sequence of n stack operations (starting with an empty stack) that includes PopMany (choose the best answers)?

☑ $O(1)$ because we can define $\Phi(h) = size$.

Correct Response
Correct.

Push operations will have an amortized cost of 2: 1 for the push, and 1 for the change in Φ.

Pop operation will have an amortized cost of 0: 1 for the pop, and -1 for the change in Φ.

PopMany operations will have an amortized cost of 0: i for the pop, and -i for the change in Φ.

Thus, the worst-case amortized cost is 2, which is $O(1)$.

☐ $O(1)$ because we can place one token on each item in the stack when it is pushed. That token will pay for popping it off with a PopMany.

▲

Correct Response

Correct. Add a token to each element on the stack as it is pushed. Then, on a PopMany, use those tokens to pay for the popping cost of each. Thus, the amortized cost is 2 which is $O(1)$.

☐ $O(1)$ because the sum of the costs of all PopMany operations in a total of n operations is $O(n)$.

▲

Correct Response

Correct. Since over n operations starting with an empty stack there can be at most n items in the stack, the sum of the costs of the PopMany operations can be at most n. Thus, the total actual costs of n operations is at most $O(n)$, so the amortized cost is $O(n)/n = O(1)$.

☐ $O(1)$ because there wouldn't be that many PopMany operations.

▲

Correct Response

This is not a convincing analysis.

☐ $O(n)$ because we could push $n - 1$ items and then do one big PopMany($n - 1$) that would take $O(n)$ time.

▲

Correct Response

When looking at amortized costs, one can't look at a single operation in isolation. How many prior stack operations would be needed in order to do a PopMany(n)?