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.

   1 / 1 point

   - ( ) minimum: 1, maximum: 1
   - (•) minimum: 1, maximum: 32
   - ( ) minimum: 24, maximum: 24
   - ( ) minimum: 32, maximum: 32
   - ( ) minimum: 1, maximum: 24

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.**

1 / 1 point

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

- (selected) 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.
- PushBack 2 elements, and then alternate $n/2 - 1$ PushBack and PopBack operations.
- PushBack $n/2$ elements, and then PopBack $n/2$ elements.

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.**

1 / 1 point

PopBack reallocates the dynamically-allocated array to a new array of half the capacity if the size is $\leq$ 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?

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

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)?

1 / 1 point

- [x] $O(1)$ because we can define $\Phi(h) = size$.

Correct

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 there wouldn't be that many PopMany operations.
- [ ] $O(n)$ because we could push $n-1$ items and then do one big PopMany($n-1$) that would take $O(n)$ time.
- [x] $O(1)$ 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

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 $O(1)$.

- [x] $O(1)$ because the sum of the costs of all PopMany operations in a total of n operations is $O(n)$.