Try again once you are ready
TO PASS 75% or higher

Try again

GRADE
45%

Dynamic Arrays and Amortized Analysis

LATEST SUBMISSION GRADE
45%

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: 24
   - ○ minimum: 32, maximum: 32
   - ◉ minimum: 1, maximum: 32
   - ○ minimum: 24, maximum: 24

   ✓ **Correct**
   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).

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

   0 / 1 point

   - ○ 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.

   ✗ **Incorrect**
   Once we've added 2 elements, each PushBack will take 3 copies, and each PopBack will take 2 copies, so, the total cost will be linear in the number of operations.

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 $\leq$ 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?

   0 / 1 point

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

   ✗ **Incorrect**
   Although this is a valid potential function (since it is 0 at time 0, and never negative), it doesn't lead to an $O(1)$ amortized cost.

   Let n be a power of 2 and do $n/2 + 1$ PushBack operations. At this point, the size will be $n/2 + 1$, and the capacity will be n, so $Phi(h) = 2$. Now, do $n/4$ PopBacks. The size will be $n/4 + 1$ and the capacity will be n, so $\Phi(h) = 0$. Do one more PopBack which will require allocating a new dynamically-allocated array of size

so $\Phi(n) = 0$. Do one more PopBack which will require allocating a new dynamically-allocated array of size $n/2$, and will require $n/4$ copies of the existing elements. The amortized cost of this last PushBack will be the true cost $n/2 + 1$, plus the new potential (0) - the old potential (0) for a total of $n/2 + 1$. So, the amortized cost of this operation is $O(n)$.

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. **0.8 / 1 point**

   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)?

   - [ ] $O(n)$ because we could push $n - 1$ items and then do one big PopMany($n - 1$) that would take $O(n)$ time.
   - [ ] $O(1)$ because there wouldn't be that many PopMany operations.
   - [ ] $O(1)$ because we can define $\Phi(h) = size$.
   - [x] $O(1)$ because the sum of the costs of all PopMany operations in a total of n operations is $O(n)$.

   ✓ **Correct**
   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 $O(n)/n = O(1)$.

   - [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 $O(1)$.

   You didn't select all the correct answers