Problem 4 (Amortized Analysis, 15 points)

Part (a) [3 Points]

Here, for every one expensive operation (of cost $\Theta(n)$), we get 9 cheap operations (of cost $\Theta(1)$). If we look at the average cost of an expensive operation and all preceding cheap operations, it will be $\Theta(\frac{n+9}{10}) = \Theta(n)$, so the worst-case amortized cost will be linear.

If we want to be a little more precise and mathematical about it, we can look at all the 'push' operations up to size n. For every (current) size k that is a multiple of 10, the cost will be $\Theta(k)$, while for all others, it will be $\Theta(1)$. Let's call this function $f(k)$ (that is, $f(k) = \Theta(k)$ for multiples of 10, and $f(k) = 1$ for other k). Then, the total cost will be:

$$\begin{aligned}
\sum_{k=1}^{n} f(k) &= \sum_{k=1:k \text{ multiple of } 10}^{n} f(k) + \sum_{k=1:k \text{ not multiple of } 10}^{n} f(k) \\
&= \sum_{k=1:k \text{ multiple of } 10}^{n} \Theta(k) + \sum_{k=1:k \text{ not multiple of } 10}^{n} \Theta(1) \\
&= \sum_{j=1}^{n/10} \Theta(10j) + (9n/10) \cdot \Theta(1) \\
&= \Theta(10 \sum_{j=1}^{n/10} j) + \Theta(n) \\
&= \Theta(n^2) + \Theta(n) \\
&= \Theta(n^2)
\end{aligned}$$

Here, we used the arithmetic series in the end. Because the total cost is quadratic, and we had n push operations, the amortized cost per operation is $\Theta(n)$, as we calculated earlier.

Part (b) [4 Points]

The simple argument is as follows. Following any expensive operation (with resizing, costing $\Theta(n)$, because the array size increased by $\sqrt{n}$, the next $\sqrt{n}$ operations will not involve resizing the array, so they cost $\Theta(1)$ each. The one expensive operation plus all subsequent cheap operations together thus cost $\Theta(n) + \sqrt{n} \cdot \Theta(1) = \Theta(n)$. Since this is amortized over $\sqrt{n}$ operations, the amortized cost per operation is $\Theta(\sqrt{n})$.

We can again use the other analysis. Here, things get a bit more tricky, because we can't easily figure out for which values of k the array gets resized. But we can get a good enough approximation. Think about some value of k between n/2 and n. Each such resize costs between $\Theta(n/2)$ and $\Theta(n)$, both of which are $\Theta(n)$. It increase the array size by something between $\sqrt{n/2}$ and $\sqrt{n}$, both of which are $\Theta(\sqrt{n})$. Because each increase is by $\Theta(\sqrt{n})$, the number of such increases to cross from $n/2$ to n is $\Theta(n/\Theta(\sqrt{n})) = \Theta(\sqrt{n})$. So going from $n/2$ to n, we have $\Theta(\sqrt{n})$ array copies, each costing $\Theta(n)$, and $n/2 - \Theta(\sqrt{n}) = \Theta(n)$ operations of cost $\Theta(1)$. Thus, the total cost going from $n/2$ to n is $\Theta(n^{3/2})$.

To get the total cost from 1 to n from this, we sum it over all powers of 2, giving us:

$$\begin{aligned}\sum_{j=0}^{\log_2 n} \Theta((2^j)^{3/2}) &= \Theta(\sum_{j=0}^{\log_2 n} (2^{3/2})^j) \\ &= \Theta(\frac{(2^{3/2})^{1+\log_2 n} - 1}{2^{3/2} - 1}) \\ &= \Theta((2^{\log_2 n})^{3/2}) = \Theta(n^{3/2})\end{aligned}$$

Since this total cost is accrued by n operations, their amortized worst-case cost is $\Theta(n^{3/2}/n) = \Theta(\sqrt{n})$.

Part (c) [4 Points]

Consider an array containing n elements, but with room for 2n. When we pop an element, we resize to n, with n-1 elements in it. Now, if we push two elements, we must expand the array size by doubling, bringing it back to 2n size, with n+1 elements. Now another deletion has us back to where we were before. Thus, out of any four operations, two involve a resizing of cost $\Theta(n)$; thus, the amortized worst-case time will be $\Theta(n)$. So not a good idea.

Part (d) [4 Points]

Here is the new reasoning. After each doubling operation, the resulting array is exactly half full (or half plus one more element). In order to force another doubling, we must insert at least n/2 elements, so there are n/2 operations of constant cost before an operation of cost $\Theta(n)$. In order to force a shrinking, we must delete at least n/4 elements, in order to get down to having only a quarter full array. Thus, again, before any expensive operation, we have at least $\Theta(n)$ cheap operations. In either case, the amortized worst-case cost of 'push' is $\Theta(1)$.

For an expensive 'pop' operation, there are two cases. To get another expensive 'pop' operation, we have to delete at least another $\Theta(n)$ elements, so the amortized worst-case cost per operation is constant. To get another expensive 'push' operation, just one immediate 'push' may be enough, so we may end up paying $\Theta(n) + \Theta(n) = \Theta(n)$ for those two operations combined. But after that, the previous case applies, and the next expensive operation will be far in the future. In each case, the amortized worst-case cost of 'pop' is $\Theta(1)$ per operation.