Question 1:

1.1 A

The worst-case runtime of a single $MULTIPOP$ operation is $O(n)$ since in a single operation, the most you can pop is the size of the stack.

1.2 B

A sequence of n operations in any combination of $PUSH$, POP, and $MULTIPOP$ should take $O(n)$ time since for an element $PUSH$'ed to the stack it can be popped at most once. And so, the number of times that a POP operation can be executed is at most the number of times that an element has been $PUSH$'ed, which is at most n, so $O(n)$.

1.3 C

What virtual cost would you give to each operation?
$PUSH \to 2$
$POP \to 0$
$MULTIPOP \to 0$

What is the real cost of each operation?
$PUSH \to O(1)$
$POP \to O(1)$
$MULTIPOP \to O(min(k, n))$
k is the number of elements to be POP'ed
n is the number of elements in the stack.

1.4 D

When we push to the stack we pre-pay a cost of 1 coin, thus each element in the stack has been half paid for. So when we pop an element, we also remove the pre-paid cost and use it to pay the POP cost, so we charge the operation nothing but use the pre-paid cost for payment. This is the same for a $MULTIPOP$ operation, since that is just a series of POP operations. Therefore we always have enough stored up in pre-paid amounts to pay for the POP and $MULTIPOP$ operations. And since the stack is always nonnegative we also ensure that the amount stored up in credit is always nonnegative as well so we will never run out of coins.

1.5 E

Since we know:

- That for any sequence of n $PUSH$, POP, and $MULTIPOP$ operations, amortized cost upper bounds the total cost by definition.
- And the total credit stored in the stack is the difference between the amortized cost and the total cost.
- Therefore the total credit is always nonnegative.
- We also know that for any POP operation, there must have been a previous $PUSH$ operation.
- Therefore we know that since the amortized cost is $O(n)$ then the total cost must also be $O(n)$

1.6 F

Since a $MULTIPUSH$ operation is simply a series of $PUSH$ operations executed k times on an object. The amortized cost of the $MULTIPUSH$ operation would be $2k$. This is not different, however, from individually pushing an object k times and so the total cost would be dependent on the number of elements in the stack and not the method of pushing. So POP and $MULTIPOP$ have unchanged amortized costs and the total amortized cost would still be $O(n)$ and so too the total actual cost.

Question 2:

2.1 A

I would use an ArrayList.

2.2 B

To insert, I would simply insert at the end of the array if there is space left. Therefore insertion would take $O(1)$ time. If there is no space left, create a new array of size $2n$ and copy over the n elements, $O(n)$ time. So for the first n elements insertion is constant time.

2.3 C

To delete the largest $\lceil \frac{n}{2} \rceil$ elements, since to achieve constant insertion the array would need to be unsorted, I would use the median method to split the array into elements larger than the median and smaller than the median. That way we are left with a half of the array that is strictly larger than the other half. Deleting that half would be done in constant $O(1)$ time and partitioning by relation to the median would be done in $O(n)$ time.

2.4 D

Since we probably don't want to measure exact runtime, an individual operation can take at the worst case $O(n)$ time, I'm going to assess runtime in the average case where there are a sequence of operations being performed.

2.5 E

The potential function Φ would be defined as the number of elements present in the Array. The potential of the empty Array, D_0 would be set to 0. So $\Phi(D_0) = 0$.

2.6 F

From the TextBook we know $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

For the $INSERT$ operation, we know that for an Array containing n elements, after the operation it would contain $n+1$ elements. Therefore we know $\Phi(D_i) - \Phi(D_{i-1}) = (n+1) - n = 1$. So $\hat{c_i} = c_i + 1$. Since c_i is the actual cost of the insert operation. Assuming the insertion is in an array containing $n-1$ elements, $c_i = 1$. Therefore for the $INSERT$ operation we have $\hat{c_i} = 2$, so $\hat{c_i}$ for $INSERT$ is constant.

For the $DELETE{-}HALF$ operation, we know that for an Array containing n elements, after the operation it would contain $\lfloor \frac{n}{2} \rfloor$ elements. Therefore we know $\Phi(D_i) - \Phi(D_{i-1}) = \frac{n}{2} - n = \frac{-n}{2}$. So $\hat{c_i} = c_i - \frac{n}{2}$. Since c_i is the actual cost of the $DELETE - HALF$ operation, $c_i = \frac{n}{2}$ and so we have $\hat{c_i} = 0$.

2.7 G

- Since we know by our definition of Φ that $\Phi(D_i) \geq \Phi(D_0)$ (because the number of elements in the array will never be negative)
- This means that the total amortized cost is an upper bound on the total cost.
- So for a sequence of m operations, we have $O(m)$