Jiawei Wang

Question 1:

- **1.1**

  Worst case runtime of Multipop is $O(n)$

- **1.2**

  Worst case runtime is impossible to reach because for a stack we need to do push (which takes $O(1)$) before we can do multipop

- **1.3**

  For virtual cost, push costs 2, pop and multipop cost 0; For actual cost, push and pop cost 1 and multipop costs n for n elements

- **1.4**

  for every element we push, we charge 2 coins but actually spend 1 coin so we save 1. So after n pushes, we can save n coins, that means coins are the same as elements. Because we can only pop one element once and one pop costs 1 coin, so we always have enough coins to pop all the elements.

- **1.5**

  For one operation, we can get at most 2 coins, so in total n operations can give at most 2n coins. operations consume coins and $1coin = O(1)runtime$. Because 1.4 is correct (we always have enough coins), then we spend at most 2n coins, which is $O(2n) = O(n)$

- **1.6**

  This will increase the amortized cost. For multipop and multipush, each time we take out / put in n elements and these operations take $O(n)$ runtime for each one. So n total operations take $O(n^2)$ runtime at most.

Question 2:

- **2.1**

  Use a dynamic array

- **2.2**

  If the array is not full, simply insert the element into it; if the array is full, expand it and then insert element. Amortize cost is $O(1)$ for one operation.

- **2.3**

  For deletion, first we have to find the median, then go through the array again to delete all elements that are bigger or equal to median, and this is $O(n)$

- **2.4**

  We will measure runtime of each operation by the number of elements we insert or delete.

- **2.5**

  $\Phi(emptydatastructure) = 0;$
  $\Phi(non - emptydatastructure) \geq 0;$
  $\Phi(datastructure) = number - of - elements$

- **2.6**

- We already know $Insertion = O(1)$ so we claim: if i-th operation is delete-half, then $\hat{c}_i = 0$

- Prove: we assume the data structure has n elements, then

$$\Phi(before) = n \tag{1}$$

$$\Phi(after) = \left\lfloor \frac{n}{2} \right\rfloor \tag{2}$$

$$c_i = \left\lceil \frac{n}{2} \right\rceil \tag{3}$$

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(after) - \Phi(before) && (4) \\ &= \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor - n && (5) \\ &= j - j && (6) \\ &= 0 && (7) \end{aligned}$$

- So the amortized cost of any operation $\hat{c_i}$ is constant.

- **2.7**

- For the potential: $\Phi(DS_0) = 0$ and $\Phi(DS_i) \geq 0$
- $\Phi(DS_i)$ is the potential of data structure after i-th operation;
  c_i is the actual cost for i-th operation;
  Then $\hat{c_i} = c_i + \Phi(DS_i) - \Phi(DS_{i-1})$
- $Total - time = \sum_{i-1}^{n}(c_i) \leq \sum_{i-1}^{n}(\hat{c_i})$
- Because $\hat{c_i} = O(1)$
- $Total - time = \sum_{i-1}^{n}(c_i) \leq \sum_{i-1}^{n}(\hat{c_i}) \leq \sum_{i-1}^{n}(O(1)) = O(n)$