By: Eric Koehli, Jacob Chmura, Conor Vedova

Question 1. [1 MARK]

Let $n \in \mathbb{N}$ be the length of the input array A and assume $n \geqslant 2$. Let $T(n)$ be the worst-case (WC) time complexity of calling `NOTHING(A)` on array A. We choose to define the function f as follows:

$$f(n) := n$$

With this choice of $f(n)$, we must now show that $T(n) \in \Theta(f(n))$.

Proof.
For a fixed iteration of the outer loop, we can see that the inner loop iterates at most $n-1$ times. The outer loop iterates at most *once* because on the first iteration we have $i = 1$, then at some later point, $j = n-1$ and the second if condition is met (since $i + j = 1 + (n-1) = n > (n-1)$) and the function returns.

Case 1: $T(n) \in \mathcal{O}(n) \iff \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geqslant n_0 \implies T(n) \leqslant cn$

If we want to argue that the WC runtime is $\mathcal{O}(n)$, we need to show that *for all* inputs x of length n, the runtime is *at most cn* for some constant c.

Let $n \in \mathbb{N}$ be the length of the input array A and assume $n \geqslant 2$. Also, let $c = 1$, $n_0 = 2$ and assume $n \geqslant n_0$. According to the logic above, we have

$$\begin{aligned} T(n) &= n - 1 \leqslant n && \text{(since } n \geqslant 2\text{)} \\ &= cn \end{aligned}$$

Case 2: $T(n) \in \Omega(n) \iff \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geqslant n_0 \implies T(n) \geqslant cn$

If we want to argue that the WC runtime is Big-$\Omega(n)$, then we need to show that *there exists* and input x of length n such that the runtime is *at least cn* for some constant c.

Let $n \in \mathbb{N}$ be the length of the input array A and assume $n \geqslant 2$. Also, let the input array A be defined as follows:

$$A[i] = \begin{cases} -1, & i \text{ is even} \\ 2, & i \text{ is odd} \end{cases}$$

for all $i \in \mathbb{N}$ such that $i \leqslant n$.

With this defined input, the first `if` clause is never met, as A[j] + A[j+1] = 1 for all $j \in \mathbb{N}$ such that $j \leqslant n-1$. However, the second condition still returns when $i = 1$ and $j = n-1$. Therefore the runtime is still $n-1$ steps. Let $c = \frac{1}{2}$, $n_0 = 2$ and assume $n \geqslant n_0$. Then we get,

$$\begin{aligned} T(n) &= n - 1 \geqslant \frac{1}{2}n && \text{(since } n \geqslant 2\text{)} \\ &= cn \end{aligned}$$

Since we have shown that $T(n)$ is bounded by Big-$\mathcal{O}$ and Big-Ω, we can conclude that $T(n) \in \Theta(n)$.

□

Answered by: Eric Koehli
Editied by: Conor Vedova, Jacob Chmura

Question 2. [1 MARK]

Listing 1: Algorithm

```
1  def alg(S: Sequence, m: int) -> None:
2      """
3      Pre: m is a natural number >= 1 and S is a possibly infinite sequence.
4      Post: Handle input keys and print operations correctly.
5      """
6      Q = MaxHeap()                          # create a new MaxHeap data structure
7
8      for key in S:                          # could be infinite
9          if key is type int:
10             # Case 1: Q.size <= m
11             if Q.size < m:                     # assume at least m keys before first print
12                 Q.insert(key)
13             # Case 2: Q.size == m
14             else:
15                 if key < Q.max():          # check if key is less than the max
16                     Q[1] = key             # set the max node to the new key value
17                     Q.drip(1)              # restore the max-heap property
18                 else:
19                     pass                   # don't store the key if it's greater than Q.max
20         else:                          # print the entire heap
21             for i = 1 to Q.size:       # i = 1, 2, ..., Q.size
22                 print(Q[i])
```

Data Structure

We are using a binary max-heap data structure that contains the input keys from the sequence provided.

Algorithm Explanation

Our algorithm creates a MaxHeap data structure Q to store the input keys from the sequence. The algorithm takes in a possibly infinite sequence S of integer keys and possibly a print statement, and a constant integer $m \geqslant 1$.

There are two general cases to consider in our algorithm. Our goal is to only store the smallest m items provided in the input sequence. To accomplish this task, we first fill our MaxHeap with the first m keys provided. After the MaxHeap has been filled, we know that $Q.size = m$, so now we need to just maintain the max-heap property and add new keys to the heap only if they are smaller than the current maximum (which is at the top of the heap).

When a `print` call is made, we simply print out all of the items stored in the MaxHeap. We know that this stores the smallest m items because $Q.size$ is *at most* m.

Note: The Sequence and for loop on line 8 can be moved outside of the function, but we chose to interpret the question in this manner.

Time Complexity Analysis

We know from class and the CLRS textbook that the WC time complexity of insert(Q, x) and $drip(Q, x)$ is $\lg(m)$, where $m = heapsize$. Thus, when a key from the sequence is an integer, this input is handled in $\mathcal{O}(\lg(m))$.

On the other hand, if we reach a `print` statement, the else block on line 20 executes. Then the worst case time complexity is linear because the for-loop operates m times. We will now prove the correctness of our algorithm by induction.

Proof. Let $m \in \mathbb{Z}^+$. Let n be the iteration count of the for-loop over the sequence on line 8. Also, n-1 represents how many keys or print statements have been processed so far. Let S be a possibly infinite sequence of distinct integer keys, and possibly print statements. Assume the Precondition holds. Define P(n): ”for the nth value of the sequence S, if the value is an integer, our algorithm inputs the value into the max-heap if it is among the m smallest keys processed so far, else, the value must be a print statement, and our algorithm prints the m smallest keys processed so far."

Base Case:
Let $n \in \{1, 2, ..., m-1\}$. The value from S must be a key as said in the question. Also, by the case, n ¡ m. Therefore the if statements on line 9 and line 11 evaluate to True. So, the key is inserted into the max-heap and P(n) is trivially satisfied.

Inductive Step:
Let $n \in \mathbb{N}$ st. $n \geq m$. Assume P(n-1) holds. WTS P(n).

Case 1: the nth value of the sequence is an integer
Since n $\geq m$, $Q.size \geq m$. By the case, the nth value is an integer. So, the if statement on line 9 is True, and the if statement on line 11 is False, so the else block on line 14 is executed. By the induction hypothesis, Q contains the m smallest keys processed after n-1 inputs. By max-heap property, the largest of these m keys is contained at the root. If the key is less then Q.max, then we put the key in the spot where the previous max of the heap was, then we dribble down the key to retain the max-heap property. As a result, Q still contains the m smallest values processed so far. If the key is not less than Q.max, then the m elements of Q are the smallest keys processed so far by P(n-1), and the above logic. In either case P(n) follows.

Case 2: the nth value of the sequence is a print statement
In this case the if statement on line 9 is False. So, line 20 executes. By P(n-1), Q contains the m smallest keys after n-1 iterations of the outer for loop. Since the nth value is a print statement, Q contains the m smallest keys after n iterations. The for loop on line 21 prints each of these keys. Therefore P(n).

By the principle of induction, P(n) holds for all n $\mathbb{Z}^+$. □

Algorithm by: Jacob Chmura, Eric Koehli, Conor Vedova
Proof by: Jacob Chmura, Conor Vedova
Edited by: Eric Koehli