CSC263

Assignment 1

Question 1

Wrote by: Xinyi Liu Checked by: Kewei Qiu

f(n) = n

Define the input size of counter as n.

Assume each line takes constant time 1.

In worst-case we need the for loops runs as more as possible so assume the first "return" in line 5 will never be executed.

①First we prove $T(n) \in O(n)$

The inner for loop will runs at most n - 1 times since in worst-case line 5 will never be executed.

With each iteration taking constant time, the inner for loop cost at most n - 1 steps.

At the last iteration of the inner loop, j is n - 1.

At this time, since i starts from 1, i + j must be larger than n – 1 so line 6 is satisfied and the program returns.

This happens at the first iteration of the outer loop.

Therefore, the two for loops cost at most n – 1 steps.

Ignore the first assignment step, the total cost of NOTHING is at most n - 1 steps.

Therefore, for all inputs, NOTHING takes at most n-1 steps i.e. T(n) is O(n).

②Then we prove $T(n) \in \Omega(n)$

Consider the input family A = [0, 1, 0, 1, …] with size n.

In this case, the inner for loop iterates n - 1 steps since the first if statement will never be satisfied.

Then the program returns at the last iteration of the inner loop which is also at the first iteration of the outer loop.

So, the two loops iterate n – 1 steps at total.

Ignore the first assignment step, the total cost of NOTHING is at most n - 1 steps.

Therefore, there exists an input such that NOTHING takes at least n-1 steps, i.e. T(n) is Ω(n).

Combining ①② we can conclude that $T(n) \in \Theta(n)$.

Question 2

Wrote by: Xinyi Liu Checked by: Kewei Qiu

(a) Data structure

The algorithm uses a data structure which is binary max-heap

For this algorithm, the min-heap has size m and it contains the m smallest keys among all the keys that have been processed where bigger key has higher priority.

(b) Algorithm

This algorithm takes a number $m \geq 1$ and a sequence of distinct integer keys.

Then it processes each key one at a time.

For the first m keys, the algorithm will store them using max-heap data structure.

Then for each next key, the algorithm will check if it is smaller than the largest element of the max-heap (i.e. the root).

If so, the root will be extracted, and this key will be inserted into the max-heap. If not, process next key.

This can be implemented by a while loop.

And every time print operation occurs, the algorithm will loop the entire max-heap and print each element.

(c) Worst-case running time

①For the worst-case running time of processing each input key, we can assume that the key should be inserted into the max-heap.

Let n be a natural number represents the number of key which is currently being inserted into the max-heap

Case $n \leq m$: Since for the first m input keys, the algorithm will store them using max-heap data structure. Since there are $n - 1$ keys stored in the min-heap, as we discussed in lecture, the running time complexity is $\log n$.

Case $n > m$: Since we know the worst-case complexity of ExtractMax() and Insert() operation of a max-heap are both $O(\log m)$ for input of size m, The running time complexity is $\log m$

Combine these two cases we can conclude that the worst-case time complexity is $O(\log m)$ to process each input key.

②For the worst-case running time of performing each print operation, the algorithm should loop each element of max-heap to print it, so it takes m steps since there are m elements. Also each print operation takes a constant running time.

Therefore, the worst-case time complexity is $O(m)$ to perform each print operation.

(d) proof of correctness

To prove the algorithm is correct, we want to prove that for all $n \geq m$, the algorithm will print the m smallest keys among all the n keys that have been processed when the print operation occurs after the n^{th} key.

Proof by induction:

Let P be the predicate defined as P(n) : "The algorithm will print the m smallest keys among all the n keys that have been processed when the print operation occurs after the n^{th} key".

Claim: P(n) holds for all natural number $n \geq m$.

Base case: $n = m$

> At this time, only m keys have been processed and according to the algorithm, all of them are stored in the max-heap. So, when the print operation occurs, all m keys will be printed.
>
> Since these m keys are (vacuously) m smallest keys among all the n keys that have been processed when the print operation occurs after the n^{th} key
>
> P(n) holds at this time.

Induction steps:

> Let k in an arbitrary natural number, $k \geq m$.
>
> Assume that P(k) holds (Induction hypothesis).
>
> In what follows we prove that P(k+1) also holds.
>
> When the $k+1^{th}$ key is being processed, there are two cases:

Case 1: the $k+1^{th}$ key is smaller than the root of the max-heap

In this case, according to the algorithm, the $k+1^{th}$ key will be inserted into the max-heap.

By the induction hypothesis, we know that the max-heap before the insert operation contains the m smallest keys among all the k keys.

Then, the root of the max-heap will be extracted and the $k+1^{th}$ key will be inserted into the max-heap.

Since the insert operation of max-heap is correct, now the max-heap still contains the m smallest keys among all the k+1 keys where bigger key has higher priority.

Then the print operation will print all elements in the max-heap so P(k+1) holds in this case.

Case 2: the $k+1^{th}$ key is larger than any of the leaf of the min-heap

In this case, the max-heap won't change and by the induction hypothesis, the max-heap contains the m smallest keys among all the k keys.

Since the $k+1^{th}$ key is larger than the root of the max-heap, it is also larger than any of the key of the max-heap according to the property of max-heap.

Therefore, the max-heap contains the m smallest keys among all the k+1 keys.

The print operation will print all elements in the max-heap so P(k+1) holds in this case.

Combining these two cases, I can conclude that P(k+1) holds when P(k) holds.

In conclusion, by the Principle of Induction, P(n) holds for all natural numbers $n \geq m$.

Hence, I have proved that my algorithm is correct. ■