CSC263H1 Assignment 1

Jiatao Xiang, Huakun Shen, Xu Wang

April 11, 2019

Q1 write together

Let $f(n) = n$, $T(n)$ is $\Theta(n)$
want to show that T(n) is $\mathcal{O}(n)$ and $\Omega(n)$

Proof. First, prove it is $\mathcal{O}(n)$
the outer loop will run at most n times and the inner loop will run at most n-1 times, which makes the runtime $\mathcal{O}(n^2)$. However, for the inner loop, notice that if $i + j > n - 1$, the loop will stop, which means if $i = 1$, the function will stop at j = n-1, thus the outer for loop only runs 1 time. which shows that the function will run at most $1 * (n - 1)$ times, which is $\mathcal{O}(n)$.

Second, want to show that it is $\Omega(n)$.
Then we need to find a input family that makes that function has time complexity of n.
Let $A = [-1, 2, -1, 2, -1, 2, \dots]$
for this input, the first "if" condition will be always satisfied. The inner loop will run $n - 1$ times and the outer loop only runs once, the total step taken is $n - 1$. Thus, it is $\Omega(n)$ □

Q2 write together

we are using max-heap in the algorithm.
Data structure: insert(A, input) max(A) extract_max(A), the algorithm of all the methods are provided in the slides of the first week, where insert and extract_max have the time complexity of $\mathcal{O}(log(n))$ and max has time complexity of $\mathcal{O}(1)$.

Sudo-Code for the Algorithm are written below:

```
global attributes: maxHeap
algorithm_A(m, input):
        if input == print:
                for number in max_heap:
                        print(number)
```

```
        else:
                if max_heap.size != m:
                        max_heap.insert(input)
                else:
                        if input < max_heap.max():
                                max_heap.extract_max()
                                max_heap.insert(input)
```

Algorithm: first, we need to tell whether an input is a number or a print. If it is print, then we just print all elements in the maxHeap(which is an array). If the input is a number, we need to discuss whether the maxHeap reach the limit, which is m. If it does not reach the limit, then we just simply insert into the heap by a simple algorithm to make sure it satisfies the property of the heap. If the maxHeap size equals to m, we need to pop the max number out and add the new number if the new number is less than the max number in the heap, otherwise, we just discard the new number and keep the maxHeap as what it is before.

Worst-case runtime: the first if condition is used to perform print operation, and it will print all elements in maxHeap, which is at most m times, thus is $\mathcal{O}(m)$. The second if condition is used to process each input key. the method like insert and eatract_max takes at most log(m) times and others just take constant time, which match the requirement.

proof correctness: using induction
First find loop invariant: At the end of the i^{th} iteration, maxHeap will contain the m smallest numbers among i items.

Proof. Let m be a random number greater than or equal to 1
Base Case1:the first input is a number. Since there is only one number, and this number will be inserted in maxHeap according our algorithm and of course maxHeap contains the m smallest numbers.
Induction Step: Assume loop invariant maintains, that is at the end of the i^{th} iteration, maxHeap will contain the m smallest number among i items. We want to show that at the $(i+1)^{th}$ iteration, maxHeap will also contain the m smallest numbers.
Case 1: the next input is print, then there is no change to maxHeap, only print command is executed, and the state of maxHeap after the $(i+1)^{th}$ iteration stays the same as the state of the previous iteration. By induction hypothesis, maxHeap only contains the m smallest numbers after the $(i+1)^{th}$ iteration.
Case 2: the next input is a number, this case can be divided into other 2 cases:
First: i is less than m, then the new number will be inserted in maxHeap directly, since the total number is less then or equal to m, maxHeap contains the m smallest numbers.
Second: i is greater than m, then by induction hypothesis, maxHeap contains m smallest numbers, and then for the next iteration, we have a new input. If the new input is less than the max number in the maxHeap, the max number will be extracted and the new input is inserted into the maxHeap and thus maxHeap still contains the m smallest numbers among i+1 items. If the new input is greater than the max number in max Heap, then we just discard this input and do nothing with the maxHeap and thus maxHeap still contains the m smallest numbers
We've proven that the loop invariant will be maintained.

□