CSCI 104 Homework 3

(Problem 2 and Problem 3)

Yang Li (yli546@usc.edu)

February 13, 2018

1 Problem 2 (Runtime Analysis)

1. The first line of code shows that i iterates over n times, so that the time complexity would be $\sum_{i=0}^{n-1} \Theta(1)$. The second line of code shows that j iterates over $(n-i)$ times, so that the time complexity would be $\sum_{j=i}^{n-1} \Theta(1)$, and then $s.push(j)$ takes $\Theta(1)$.

The next line of for loop iterates from n to i, so that the running time is $\sum_{k=i+1}^{n} \Theta(1)$, so that these two nested for loops take a combined running time of $\sum_{j=i}^{n-1} \Theta(1) + \sum_{k=i+1}^{n} \Theta(1)$, which is equal to $2 * \sum_{k=i}^{n-1} \Theta(1)$. Therefore, these two lines combined would have a running time of $2 \sum_{i=0}^{n-1} \sum_{k=i}^{n-1} 1 = \Theta(n^2)$.

Since stack s would be empty be the end of for loops. It would only cost a constant operation to check that the condition in while statement is false. Therefore, $\Theta(n^2) + \Theta(1) = \Theta(n^2)$.

2. The demonstrated example is a recursion function and the main calls $func(0, n)$, in which n is a parameter of curr in the function definition. It would execute the second line of the function, so that the function would call $n-1$ times. And it would continue until the first parameter becomes zero. Therefore, the running time can be represented by $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \Theta(1) = \sum_{i=0}^{n-1} \Theta(n) = \Theta(n^2)$, so that the time complexity would be $\Theta(n^2)$.

3. As every operation of Queue ADT takes constant time, the first for loop would have the running time of $\sum_{i=0}^{n-1} \Theta(1) = \Theta(n)$. In the while loop, it would take approximately $(n-1)$ times after till it hits the 1 at the front and the nested for loop inside the while loop would only happen once. Therefore, the queue would eventually have $2n$ number of elements. As the else statement does not reduce the number of element and if statement would reduce one element for every two loop, it would take quadrupled number of elements to reach the end of the code. Overall, the time complexity is $\Theta(n)$.

4. As i increments and curr→value $\mod i$, it actually represent a harmonic series $\sum_{i=1}^{n/i} \Theta(i)$, which has a time complexity of $\Theta(\log n)$. combined with two outer while and for loop, the total time complexity could be represented by $\sum_{i=0}^{n-1} \sum_{j=1}^{n-1} \Theta(\log n) = \Theta(n^2 \log n)$.

2 Problem 3 (Amortized Analysis)

Part (a)
The worse case runtime would happen when the $n = max$, so that somefunc() would call bar(). In this way, it would have a time complexity of $O(n^2)$.

Part (b)
Total runtime $= \sum_{i=1}^{\log n} \Theta((2^i)^2) + (n - \log n) * \Theta(\log n) = \Theta(n^2) + (n - \log n) * \Theta(\log n) = \Theta(n^2) + \Theta(n \log n) = \Theta(n^2)$. Therefore, amortized runtime $= \Theta(n^2)/n = \Theta(n)$.

Part (c)
Total runtime $= \sum_{i=1}^{\log n} \Theta((2^i)^2) + (n - \log n) * \Theta(n \log n) = \Theta(n^2) + (n - \log n) * \Theta(n \log n) = \Theta(n^2) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$. Therefore, amortized runtime $= \Theta(n^2 \log n)/n = \Theta(n \log n)$.

Part (d)
The worst sequence of calls would happen when somefunc calls twice and then anotherfunc calls twice and then they continue to alternate. In this way, the amortized runtime per call would be $\Theta(n^2)$ because somefunc() and anotherfunc() would always call bar(), which takes $\Theta(n^2)$ time.