Karim Sammouri
Dr. Philippe Giabbanelli
CSE 374 – A
4 March 2021

Assignment 3 Part V

1) $\Theta(\log(\log(n)))$: An example of an algorithm that will have a time complexity in the order of $\Theta(\log(\log(n)))$ is as follows: The function will take in as an argument an array list containing consecutive integers from one to n inclusive as per the documentation. The function, through a for-loop, will go through the array list in a logarithmic fashion by starting with an index at the end of the array list and dividing the index by two with every iteration. Inside the loop, the element at the specified index of the list given as an argument will be added to **another pre-initiated local array list**. After this first loop has finished going over the argument-given list and filled the local array list. Another loop outside the aforementioned loop will go through the local array list also in a logarithmic fashion by starting with an index at the end of the array list and dividing the index by two with every iteration. Inside this loop, we will simply print out the list element at the specified index. Thus, this is an algorithm in the order of $\Theta(\log(\log(n)))$ because we're taking the output (the logarithmically filled local list) of an $\Theta(\log(n))$ operation (the first loop) and using it as input to another $\Theta(\log(n))$ operation (the second loop).

2) $\Theta(\log^2(n))$: An example of an algorithm that will have a time complexity in the order of $\Theta(\log^2(n))$ is as follows: The function will take in as arguments two arrays of the same length containing consecutive integers from one to n inclusive as per the documentation. The function, through a for-loop, will go through the first array in a logarithmic fashion by starting with an index at the end of the array and dividing the index by two with every iteration. Inside this loop, another inner loop will go through the second array in also a logarithmic fashion by starting with an index at the end of the array and dividing the index by two with every iteration. Inside this inner loop, we will check whether the element at the index specified by the outer loop in the first array is equal to the element at the index specified by the inner loop in the second array. If so, we'll simply increment a pre-initiated counter

variable by one. In the end, we’ll return this variable. Thus, this is an algorithm in the order of $\Theta(\log^2(n))$ because we’re nesting a loop with a logarithmic time complexity within a loop also with a logarithmic time complexity: $\log(n) * \log(n) = \log^2(n)$.