COMP2123 Assignment 1

Ruoyang Lu 490443653

- Question 1
    - The upper bound running time of this algorithms by using big O-notation is O(n)
- Question 2
    - A)
        - The essence of queue is FIFO, which stands for first in , first out.[1] Every activities done by queue is O(1) in term of time complexity. Therefore, a two variables named sum and count. The purpose of having a sum variable is when doing enqueue process, sum starts from 0 and plus every element just parsed in. For instance, the first two elements of an array is 10 and 11. Sum variable will plus 10 from 0, then plus 11. Once index pointing the end, sum will have value of 21. Same to the count variable, when an element parsed in, count variable will do plus 1 till the end of the

[1] https://www.geeksforgeeks.org/queue-data-structure/

process. Now when we call getAverage(), just simply return the value of sum divided by count, based on the parse process. The time complexity of this operation is O(1)

- B)
    - We can assume that the getAverage() function always return the true value and its time complexity is O(1). First assume that the array is an empty one, by now, we only call this array once, plus 0 time enqueue and 0 time dequeue. Those 2 variables of sum and length are both 0, which getAverage() does not have any mathematical meanings. Second, assume that the array has only one element, once getAverage() function get called, only one time of enqueue been called, the value of sum is that element's value, the count value is 1, the return value is element's value/1 and it equals to that element's value which means is correct. Third, assume that array has N elements, the count of this array is N, temporarily defined sum as M. Now enqueue

process been called for M times, dequeue process been called for M-N times. The getAverage() will return $\frac{T}{K}$. Based of those previous three assumes, Add another element of K into queue, enqueue process will added count value from N to N+T, sum value will add to M+K, now the average value should be $\frac{N+T}{M+1}$, Therefore we can get $(k+1)\text{getAverage}_{M+1} = M*\text{getAverage}_{M} + T$[2]

- C)
    - Whether sum and count all done by once. Call getAverage() by returns sum/count. Therefore, the running time/ time complexity for this algorithms is O(1)

- Question 3
    - A)
        - In this case, Two Pointer algorithms can be used. Setting up this algorithms with two pointer and final value K were needed. First, pointer i starts from index 0, then set up

[2] https://stackabuse.com/mathematical-proof-of-algorithm-correctness-and-efficiency/

another pointer j but starts from the array length – 1. Set up a count variable to count the value. Using a while loop when the condition (I < J) always true. If array[i] + array[j] = K, then return true. Else if array[i] + array[j] < k, index of i will plus 1 to the next index. If the sum of elements at current pointers is more, lower down the value by subtracting 1 on index J, finally return count[3].

- B)
    - To prove the correctness of this algorithm. First let's assume that array is empty, based on the algorithm, i = 0 and j = -1 this time. While loop will not be executing because of the condition does not met. The condition here requires I < j and 0 < -1 is obviously somehow impossible to execute. Therefore the algorithm return 0 by this time, so whether what value of K is, it always true. Second, when I < j, can ensure that array[i] < array[j] because the array is sorted, if I < j

[3] https://www.geeksforgeeks.org/two-pointers-technique/

then the value of array times those 2 index will always true. However, we have 2 different possible here. One possible is when array[i] times array[j] <= K, and the other one is when array[i] times array[j] > K. In the first possible, the array was sorted from small to large, therefore any value from array[i] to array[j] times array[i] is less than K. Therefore array[i] times array[p] is less or equal to K (i<p<j). Thus, counter need to add the value where j-i. In the second possible, there's no need to consider any value larger than array[j] in order to times array[i]. Therefore j -= 1 here.

    - Overall, every single time when executing the while loop statement, the value of counter will be added properly and no values to be left. So the correctness of this algorithm can be prove.

- C)
    - Even though we have 2 pointers here, but within while loop, index I and j only add or subtract once, therefore the loop only

executes for array length times. Every execution under while loop considered as O(1), Therefore the entire time complexity of this algorithm is O(N)