Kevin Andrews – netID: kandre28 – UIN: 655237017

Project 2

ServiceQueueSlow2.h Runtime Analysis

give_buzzer() – Runtime: constant amortized

There are no loops dependent on any type of size “n” in the give_buzzer function. Every vector function that is used in the give_buzzer function, including size(), back(), and pop_back(), are of constant complexity. The only exception is the push_back() function, which is called once. The reason this is an exception is because there is one case that push_back will take linear time, due to having to expand its size, copying over all its values to a larger vector. However, because this has little to no impact on the overall runtime of the function due to its uncommon nature, it is considered only an amortized case. Therefore, the overall runtime of the give_buzzer function is constant amortized.

seat() – Runtime: linear

There are no loops dependent on any type of size “n” in the immediate seat function. There are really only 3 functions that could increase the seat function beyond linear runtime; The vector function, size, has constant complexity. Slide_left has linear complexity dependent on n, where n is the size of the queue. The vector function push_back has, as discussed in the give_buzzer runtime analysis, constant amortized complexity. The linear complexity of slide_left dominates the other two functions. Therefore, the overall runtime for the function seat is linear.

take_bribe() – Runtime: linear

There are no loops dependent on any type of size “n” in the immediate take_bribe function. There really only 3 functions that could increase the seat function beyond linear time; The functions of find, slide_left, and push_front. The find function has linear complexity dependent on n, where n is the size of the queue. The slide_left function also has linear complexity dependent on n, where n is the size of the queue. The push_front function also has linear complexity dependent on n, where n is the size of the queue. Since all 3 functions called inside take_bribe are linear, the overall runtime of the function is linear.

kick_out() – Runtime: linear

The kick_out function acts similarly to the take_bribe function. There are no loops dependent on any type of size “n” in the immediate take_bribe function. There really only 3 functions that could increase the seat function beyond linear time; The functions of find, slide_left, and the vector function push_back. The find function has linear complexity dependent on n, where n is the size of the queue. The slide_left function also has linear complexity dependent on n, where n is the size of the queue. The push_back function, as discussed previously, has constant amortized complexity. The linear nature of the first two functions are dominant over the third functions. Therefore, the overall runtime of the kick_out function is linear.

snapshot() – Runtime: linear

The snapshot function does not contain any loops or conditionals. It only contains a single call to the vector function clear and an assignment of one vector to another. The vector function clear is of linear complexity dependent on n, where n is the size of the vector. The assignment of one vector to another actually executes an element by element copy of the right side vector into the left side vector. This copy has linear complexity dependent on n, where n is the size of the right side vector. Since both operations take linear time, the overall runtime of the snapshot function is linear.

length() – Runtime: constant

The length function only contains one operation. It returns the return value of the vector function size. Size is of constant complexity, Therefore, the length function overall has a runtime of constant complexity.