Gray Houston

ghousto

Project 1 Analysis Questions

Part 1

1.)

a.) The worst case running time/computation for the enqueue operation when no resizing of the array occurs is **O(c)** where “c” is an arbitrary constant value. This operation occurs in constant time because the array is never traversed. The only operations that occur in this function (without resizing) are placing the patient into the array at a known index, modularly incrementing the tail index, and incrementing the counter for the patients. All of these operations run in O(c) time. Therefore, the enqueue operation runs a worst case of O(c) time.

b.) The worst case running time for the dequeue operation without resizing the array is **O(c)** where “c” is an arbitrary constant. Similar to the enqueue operation, dequeue runs in constant time because the array is never traversed and the internal computations/assignments occur in O(c) time (see above for examples of internal operations running in constant time).

c.) The worst case running time for the size() operation regardless of the array resizing or not is **O(c)** where “c” is an arbitrary constant. First, the array is never traversed. Second, this function only returns the value of a counter variable, which runs in O(c) time. Third, the increment/decrement of this counter runs in O(c) time and occurs during the enqueue/dequeue operation. Therefore, the worst case running time of the size operation is O(c).

Furthermore, because only a variable is returned during a call to the size() function and the counter variable is updated before any potential array resizing in the enqueue and dequeue operations (in fact, the counter variable is used as a loop bound in array resizing), the running time of the size operation/function is independent of array resizing. Therefore, the worst case running time in all situations for the size operation is O(c).

2.)

a.) The worst case running time for the enqueue operation with array resizing is **O(n)** where “n” is the number of elements in the queue. During array resizing, all of the elements in the old array must be copied to the new array, resulting in “n” number of assignment operations. Because the memory allocation and the computations in the array resizing run in O(c) and the other computations of the enqueue method run in O(c) time (as shown in part “a” above), the “n” assignments used to copy the elements becomes the determining factor. Because the worst case running time for copying the array runs in O(n), the overall worst case running time for the enqueue operation with array resizing is O(n).

b.) Similar to the enqueue operation, the dequeue operation with array resizing is **O(n)** where “n” is the number of elements in the queue. With the same reasoning as with enqueue, during array resizing, copying the elements into the new array will cost “n” assignments and therefore run in O(n) time while all other computations will cost O(c) time. Therefore, copying the array elements becomes the determining factor of running time, resulting in the dequeue operation with array resizing to run in O(n) time.

c.) As proven above in question 1.c, the size operation will run in O(c) time, where “c” is an arbitrary constant, in the worst case even with array resizing. Because the size function's only action is to return a counter variable and the counter variable is updated before array resizing occurs, the size operation is independent of array resizing and its O(n) running time. Therefore, the O(n) time of array resizing is not included when calculating running time for the size operation. For this reason, the answer is the same as in question 1.c: the worst case running time of the size operation is O(c), even with array resizing.

3.)

The length of the queue can mathematically be calculated from the array length and head & tail indices using modulus arithmetic to accounting for wrapping around the end of the array. The equation for the queue length using Java modulus logic is

qLength = ((tail – head) % arrayLength + arrayLength) % arrayLength

where “qLength” is the length of the queue, “tail” is the tail index (i.e. index to insert a new element during the enqueue operation), “head” is the head index (i.e. index of element at the front of the queue), and “arrayLength” is the length of the physical array.

In regards to running time, this calculation takes O(c) time. Because retrieving the length of an array occurs in O(1) or O(c) time due to the Java implementation and basic math operations run in O(c) time, the computation shown above will run in O(c) time. Because this calculation runs in constant time, this cost is negligible.

Part 2A