Stacks & Queues Efficiency & Extension Problems

- Efficiency/time complexities for the topic are considered. Note that these discussions are not exhaustive or representative of the types of questions that may be found on the exams.
- Practice problems are given to test your understanding of the topic. These problems are not for a grade, but promote a deeper level of understanding of the topic. Unless otherwise specified, assume you are using *n* data/objects/elements in the data structure or algorithm.
- We look at the worst/average/best case of Big O; however, do not expect all three cases to be explored extensively.
- Feel free to discuss these problems in recitation or during office hours at the TA Help Desk.

Efficiency Stacks

Stacks are your first ADT and can be implemented using a singly linked list or an array. Stacks are a linear data structure with specific operations of *push*, *pop*, *top*, *isEmpty*, and *size*. Stacks allow one to view/access the top of the stack quickly at all times. Stacks are not designed to be searched or to randomly access data further down in the stack.

Backed by a Linked List

1) What is the time complexity of peeking at the top of the stack? __________

2) What is the time complexity of pushing data onto the stack? __________

3) What is the time complexity of popping data off the stack? __________

4) What is the time complexity of determining if the stack is empty? __________

5) What is the time complexity of checking the size of the stack? __________

Backed by an array

1) What is the time complexity of peeking at the top of the stack? __________

2) What is the time complexity of pushing data onto a stack that is not full? __________

3) What is the time complexity of pushing data onto a full stack? __________

4) What is the time complexity of popping data off the stack? __________

5) What is the time complexity of determining if the stack is empty? __________

6) What is the time complexity of checking the size of the stack? __________

Efficiency Queues

Queues are your second ADT and can be implemented using a singly linked list or an array. Queues are a linear data structure with specific operations of *enqueue, dequeue, front, back, isEmpty*, and *size*. Queues are not designed to be searched or to randomly access data further down in the queue.

Backed by a Linked List

1) What is the time complexity of looking at the front of the queue? __________
2) What is the time complexity to enqueue data into the queue? __________
3) What is the time complexity to dequeue data out of the queue? __________
4) What is the time complexity of determining if the queue is empty? __________
5) What is the time complexity of checking the size of the queue? __________

Backed by an array

1) What is the time complexity of looking at the front of the queue? __________
2) What is the time complexity to enqueue data into a queue that is not full? __________
3) What is the time complexity to enqueue data into a full queue? __________
4) What is the time complexity to dequeue data out of the queue? __________
5) What is the time complexity of determining if the queue is empty? __________
6) What is the time complexity of checking the size of the queue? __________

Extension

Coding

For these questions, you must use the Stacks and Queues from Java's API. Do not use your homework files.

In these problems, the leftmost number is the next to be popped or removed. Thus, a stack of [1, 2, 3] written in a problem would have 1 removed next, and a queue of [1, 2, 3] would also have 1 removed next. The only functions you may call are enqueue, dequeue, push, pop, isEmpty, and size. You may not create any additional storage structures other than what the problem specifies. You may create as many primitive variables as you would like. As with any work from this course, efficiency matters a lot! Always try to come up with a more efficient solution! Do NOT use recursion.

1) Write a method called isNumPalindrome() that accepts a list of integers as a parameter and checks if the numbers in the list are the same in reverse order. For example, if the list [5, 1, 2, 9] is passed in, then the method would return false. If the list [5, 1, 1, 5] is passed in, then the method would return true. An empty stack is considered to be a palindrome. Make sure that the Stack is not modified by the end of the method; the contents of the stack before running

the method should be same after running the method. You may use one stack as a helper to complete this method.

2) Write a method called doubleNums() that takes in a list of integers as a parameter and adds an extra occurrence of each integer immediately after the integer. For example, if the list [5, 1, 2, 9] is passed in, then the method should change it to [5, 5, 1, 1, 2, 2, 9, 9]. You may use a queue as a helper to complete this method.

3) Write a method called sort() that accepts a list of integers as a parameter that is already sorted by magnitude and not sign. For example, [-2, 5, 8, -13, 15] is an acceptable parameter because the list would be considered sorted if you took the absolute value of number. Your method would take this list, and continue to sort the list by sign. Thus, for the list provided above, your method returns [-13, -2, 5, 8, 15]. You may use one stack or queue as a helper to complete this method.