Maxwell Schaefer

I

A)

A doubly linked list would not be beneifical here since you would only need to traverse the list once, forward so there are no beneifts to a doubly linked list.

B)

A doubly linked list still wouldn't provide any advantages here since it should take approximately the same amount of time traversing to "Thomas" then halway back or traversing to "Thomas" then starting back at the head and traversing half the length.

II

```
public class LList{
    LListNode header;

    public class LListNode {
        private String studentName;
        public LListNode nextNode;
        public LListNode lastNode;
    }

    public boolean swapUp(LListNode P) {
        P.lastNode = P.lastNode.lastNode;
        P.lastNode.nextNode = P.nextNode;
        P.nextNode = P.lastNode;
        P.nextNode.lastNode = P;
    }
}
```

III

A)

When adding an element to the queue, first move all elements already in the first stack into the second stack, then put the new element in the first stack. After that move all elements back from the second stack to the first stack.

```
public class StackQueue {
    Stack stack1;
    Stack stack2;

    public void add(node) {
        while(!stack1.isEmpty()) {
            stack2.push(stack1.pop());
        }
```

```
            stack1.push(node);
            while(!stack2.isEmpty()) {
                stack1.push(stack2.pop());
            }
        }

        public node remove() {
            return stack1.pop();
        }
    }
```

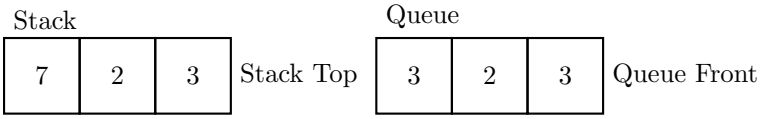
B)

To add an element to the top of the stack, move all elements in main queue to the secondary queue, then enqueue the element being pushed onto the stack in the first queue, then enqueue all the elements from the secondary queue back to the primary queue.

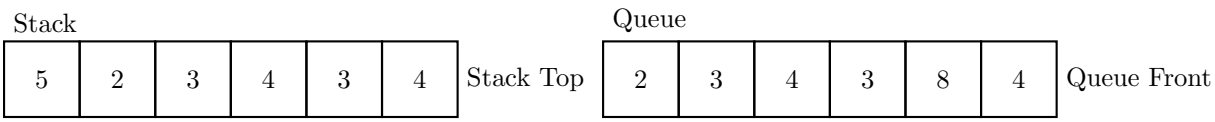
```
public class QueueStack {
    Queue queue1;
    Queue queue2;

    public void add(node) {
        while(!queue1.isEmpty()) {
            queue2.add(queue2.remove());
        }
        queue1.add(node);
        while(!queue2.isEmpty()) {
            queue1.add(queue2.remove());
        }
    }

    public node remove() {
        return queue1.remove();
    }
}
```

IV

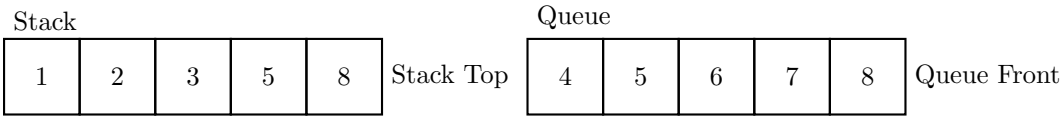
A)



B)



C)



V

A)

$O(a + b + c)$

Always returns 0.

B)

$O(1)$

Calculates n factorial.

C)

$O(1)$

Will always return n.

VI

$T1(n) \leq cf(n)$ and $T2(n) \leq cg(n)$, therefore $T1(n)+T2(n) \leq cf(n)+cg(n) \Rightarrow T1(n)+T2(n) \leq c(f(n)+g(n))$ so by the definition of Big O, $T1(n) + T2(n) = O(f(n) + g(n))$