CSC 212 Midterm 1 - Fall 2014

College of Computer and Information Sciences, King Saud University
Exam Duration: 2 Hours

18/11/2014

Question 1 [25 points]

Find the number of steps and the corresponding big-oh notation for the following two methods. **Your answer must be written on the answer sheet. Copy only the line numbers; Do not copy the code**:

1.

	Statement	S/E	Frequency	Total
1	int func(int n) {			
2	int sum=0;			
3	for(int i=0; i< n^2; i++) {			
4	for(int j=i; j< $i+3$; j++) {			
5	sum=i+j;			
6	System.out.println(sum);			
7	}			
8	}			
9	}			
Total operations				
Big-oh				

2.

	Statement	S/E	Frequency	Total
1	int func(int n) {			
2	int sum=0;			
3	for(int i=0; i< n; i++) {			
4	for(int j=i; j>=0; j- -) {			
5	sum=i+j;			
6	System.out.println(sum);			
7	}			
8	}			
9	}			
Total operations				
Big-oh				

Question 2 [25 points]

1. Complete the implementation of the class *DoubleLinkedList* below. Write down the methods: *public void findPrevious()*, *public boolean last()*, *public void update(T e)*, and *public void insert(T e)*.

2. Add to the class the method *updateLast*, that takes an element e, and replaces the last element with e. Assume it will be called on a non-empty double-linked list. **Do not use other class methods or auxiliary data structures**. The method signature is *public void updateLast(T e)*.

   Example 2.1. *If the double-linked list was:* $A \leftrightarrow B \leftrightarrow C \leftrightarrow D$. *After calling* updateLast("E"), *the double-linked list will be:* $A \leftrightarrow B \leftrightarrow C \leftrightarrow E$.

```
public class DoubleLinkedList<T> {
        private Node<T> head, current;
        public DoubleLinkedList() {
                head = current = null;
        }
}
```

Question 3 [25 points]

1. Write the method *reverse* (member of *LinkedQueue*), that reverses the queue elements' order. **Do not use other class methods or auxiliary data structures**. The method signature is *public void reverse()*.

   Example 3.1. *If the queue contains:* $A \rightarrow B \rightarrow C \rightarrow D$, *then call* reverse() *will result in the queue becoming:* $D \rightarrow C \rightarrow B \rightarrow A$.

2. Write the method *exchange* (member of *ArrayQueue*), that takes as parameters two integers i, j, and exchanges the elements at positions i and j. The first element of the queue has position 0. Assume that $0 \leq i < j < n$, where n is the length of the queue. **Do not use other class methods or auxiliary data structures**. The method signature is *public void exchange(int i, int j)*.

   Example 3.2. *If the queue contains:* $A \rightarrow B \rightarrow C \rightarrow D$, *then call* exchange(1, 3) *will result in the queue becoming:* $A \rightarrow D \rightarrow C \rightarrow B$.

Question 4 [25 points]

1. Write the static method *insertList* (user of List ADT), that takes two lists l_1, and l_2, and an index i as inputs. It should insert the elements of list l_2 after the element at position i in list l_1. Assume that i is within the range of list l_1, and that the first element has an index of 0. The list l_2 should not be changed after the method. The method signature is *public static* $<T>$*void insertList(List*$<T>l_1$*, List*$<T>l_2$*, int i)*.

   Example 4.1. *If* l_1 *contains:* $A \rightarrow B \rightarrow C \rightarrow D$, *and* l_2 *contains:* $M \rightarrow N \rightarrow O$,*then after calling* insertList(l_1, l_2, 1), l_1*'s content becomes* $A \rightarrow B \rightarrow M \rightarrow N \rightarrow O \rightarrow C \rightarrow D$.

2. Write the method *aggregate*, member of class *LinkedList*, which groups similar elements next to each other. The order of the groups must follow the order of their first elements in the original list. Assume that the list is not empty, and use the method *equals* to test for equality. **Do not use any auxiliary data structures and do not call any methods**. The method signature is *public void aggregate()*.

   Example 4.2. *If the list contains:* $B \to A \to A \to R \to C \to A \to C \to R$*, then after calling* aggregate*, its content becomes* $B \to A \to A \to A \to R \to R \to C \to C$.

.1 Specification of ADT List

- FindFirst ( ): **requires**: list L is not empty. **input**: none. **results**: first element set as the current element. **output**: none.
- FindNext ( ): **requires**: list L is not empty. Current is not last. **input**: none. **results**: element following the current element is made current. **output**: none.
- Retrieve (Type e): **requires**: list L is not empty. **input**: none. **results**: current element is copied into e. **output**: element e.
- Update (Type e): **requires**: list L is not empty. **input**: e. **results**: the element e is copied into the current node. **output**: none.
- Insert (Type e): **requires**: list L is not full. **input**: e. **results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.
- Remove ( ): **requires**: list L is not empty. **input**: none. **results**: the current element is removed. If L is empty, current will point to null. If the next element exists, it is made current, else the first element is made current. **output**: none.
- Full (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L has reached the maximum then flag is set to true otherwise false. **output**: flag.
- Empty (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **output**: flag.
- Last (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the last element is the current element then flag is set to true otherwise false. **output**: flag.

.2 Specification of ADT Double Linked List

In addition to the operations of the ADT List:

- FindPrevious ( ): **requires**: list L is not empty. Current is not first. **input**: none. **results**: element preceding the current element is made current. **output**: none.
- First (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the first element is the current element then flag is set to true otherwise false. **output**: flag.