CSC 212 Midterm 1 - Spring 2014

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

12/03/2014

Question 1 [25 points]

Find the running time and the corresponding big-oh notation for the following two methods (write your answer on the exam answer sheet. Do not copy the code, but indicate the line number clearly):

1.

	Statement	S/E	Frequency	Total
1	void func(int n) {			
2	for(int i=n-1; i >= 0; i=i-1) {			
3	int sum= 0;			
4	for(int j=0; j <i; j++) {			
5	sum+= j;			
6	System.out.println(sum);			
7	}			
8	}			
10	}			
	Total operations			
	Big-oh			

2.

	Statement	S/E	Frequency	Total
1	void func(int n) {			
2	for(int i=0; i<n*n; i++) {			
3	int j=1;			
4	while(j <= n) {			
5	j++ ;			
6	System.out.println(i+j);			
7	}			
8	}			
10	}			
	Total operations			
	Big-oh			

Question 2 [25 points]

1. Complete the implementation of the Double-Linked List ADT below. Write down the methods: *public boolean full(), public void findFirst(), public boolean last(), public void insert(T e).*

2. Add to the class the method *public void removeLast()*, that removes the last element. If current points to the removed element, the first element is made current, else current is not modified. Assume it will be called on a non-empty Double-Linked List, and do not reuse other class methods.

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

Question 3 [25 points]

1. Write a static method (User of ADT) named *replace* that accepts a queue q and two integers i and j, and replaces the element at position i with the one at position j (the first element in the queue has position 0). Element j is not modified. The queue order should not change otherwise. Assume that $0 \leq i < n$ and $0 \leq j < n$, where n is the length of the queue (notice that we may have $i \geq j$ as well as $i \leq j$). The method signature is *static <T >void replace(Queue <T >q, int i, int j).*

   Example 3.1. *If q contains:* $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$*, then after the call* replace(q,1,4)*, q becomes:* $A \rightarrow E \rightarrow C \rightarrow D \rightarrow E$.

2. Rewrite the previous method as a member of the class *LinkedQueue.*

3. Give the big-oh notations for the running time of the previous two methods and compare them.

Question 4 [25 points]

1. Write the static method *insertIth* (user of List ADT), that takes a list l, an index i, and an element e as inputs. It should insert the element e after the element at position i in the list l. The new element is made current. You can assume i is within the range of the list, and that the first element has an index of 0. The method signature is: *public static< T >void insertIth(List <T >l, int i, T e).*

   Example 4.1. *If* $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$*, and we call* insertIth(l, 2, "X")*, then l will become:* $A \rightarrow B \rightarrow C \rightarrow X \rightarrow D \rightarrow E$.

2. Write the method reverse (member of LinkedList ADT), that reverses the list nodes order. Do not use other class methods or auxiliary data structures. The method signature is *public reverse().*

   Example 4.2. *If the list contains:* $A \rightarrow B \rightarrow C \rightarrow D$*, then calling* reverse() *will result in the list becoming:* $D \rightarrow C \rightarrow B \rightarrow A$.

.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 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.

.3 Specification of ADT Queue

- Enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- Serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- Length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- Full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.