[**Closed book and notes.**] Show all of your work clearly in the space provided. Be sure to **read each problem carefully**. Note that the exam is double sided.

1. (10 points) One can look at the element about to come out of a Queue using either the element() or the peek() method. How do these two methods differ?

2. (10 points) Explain why an ArrayList is not an appropriate choice when implementing a pure queue interface.

3. (15 points) Suppose that the CircularQueue class has been implemented and the the integer passed to the constructor sets the capacity for the circular queue. Illustrate what the data stored in memory would look like after the following operations had been performed. Be sure that your illustration makes clear which element is at the front of the queue and which element is at the back.

```
CircularQueue<Character> waitingCharacters = new CircularQueue<>(10);
waitingCharacters.offer('A');
waitingCharacters.offer('B');
waitingCharacters.offer('C');
waitingCharacters.offer('F');
waitingCharacters.offer('C');
waitingCharacters.offer('Q');
waitingCharacters.remove();
waitingCharacters.remove();
```

4. (15 points) Recall that the Java LinkedList class implements the Queue interface. Illustrate what the data stored in memory would look like after the following operations had been performed:

```
Queue<Integer> waitingIntegers = new LinkedList<>();
waitingIntegers.offer(1);
waitingIntegers.remove();
waitingIntegers.offer(2);
waitingIntegers.offer(3);
waitingIntegers.peek();
waitingIntegers.offer(4);
waitingIntegers.offer(5);
waitingIntegers.remove();
waitingIntegers.remove();
waitingIntegers.offer(6);
```

5. Recall the recursiveSearch() method from the lab 6 assignment.
Assume that the recursive component was implemented as:

```
recursiveSearch(row+1, col, false);
recursiveSearch(row,   col+1, false);
recursiveSearch(row-1, col, false);
recursiveSearch(row,   col-1, false);
```

Assume the following grid of letters as input:

A	B	C
E	F	G
H	I	J

For partial credit, be sure to explain your answer.

(a) (5 points) Suppose that the method starts at **A**, i.e., recursiveSearch(0, 0, false); What will currWord contain the first time it is four characters long?

(b) (5 points) Suppose that the method starts at **F**, i.e., recursiveSearch(1, 1, false); What will currWord contain when it has maximum length?

(c) (5 points) Now suppose that the recursive component is changed to:

```
recursiveSearch(row,   col-1, false);
recursiveSearch(row-1, col,   false);
recursiveSearch(row,   col+1, false);
recursiveSearch(row+1, col,   false);
```

Use same grid:

A	B	C
E	F	G
H	I	J

and that the method starts at **F**, i.e., recursiveSearch(1, 1, false); What will currWord contain the first time it is four characters long?

(d) (5 points) Now suppose that the recursive component is changed to:

```
recursiveSearch(row-1, col-1, false);
recursiveSearch(row-1, col+1, false);
recursiveSearch(row+1, col-1, false);
recursiveSearch(row+1, col+1, false);
```

and that the method starts at **F**, i.e., recursiveSearch(1, 1, false); What will currWord contain when it has maximum length?

6. (15 points) Implement a recursive version of the contains method for a binary search tree. Assume that an empty tree has a root==null and that the recursive version is called as follows:

```
public boolean contains(E target) {
  return contains(root, target);
}
```

7. (15 points) Create a class that implements the following PureStack interface but does not add any additional `public` methods. You may use any classes from the Java Collections Framework that you would like.

```
public interface PureStack<E> {
  public E peek();
  public E pop();
  public void push(E item);
  public int size();
  public boolean isEmpty();
}
```

Additional work area for any problem. Clearly identify which problem is associated with the work on this page.