CS2852 Exam 2 Name:

No note-sheets, calculators, or other study aids on this exam. Write your initials at the top of each page except this one. Read through the whole exam before you get started.

Have fun!

1. (2 points) ***Select one:*** To ensure print jobs are handled in the order they are received, they should be stored in…
   a. A LinkedList
   b. A Queue
   c. A Stack
   d. An ArrayList
2. (2 points) ***Select one***: If a queue is implemented by wrapping Java's `LinkedList`, and the `poll` method is implemented with `list.remove(0)`, the `offer` method should be implemented with
   a. `list.get()`
   b. `list.get(list.length()-1)`
   c. `list.add(e)`
   d. `list.add(list.length()-1,e)`
3. (2 points) ***Select one***: A stack is:
   a. MIMO
   b. LIFO
   c. FIMO
   d. FIFO
4. (2 points) Suppose a queue is implemented by wrapping Java's `ArrayList`, with the front of the queue at index 0 and the back (rear) of the queue at index *n*-1. ***Write*** the Big-O runtime for the following methods:
   a. `queue.poll()`
   b. `queue.offer(E e)`
   c. `queue.peek()`
   d. `queue.isEmpty()`
5. (4 points) ***Describe*** an error that our `Stack` interface can help you catch at compile-time.
6. (5 points) ***Describe*** the difference between the `offer(e)` and `add(e)` methods of Java's `Queue` interface.
7. (3 points) ***Give an example*** of when you would want to use Java's add(e) instead of offer(e), and when you would want to use Java's offer(e) instead of add(e).
8. (5 points) The items 8 5 7 2 1 4 are put onto a queue in that order. ***Write*** the items in the order they will come off the queue.

9. Consider the circular queue we implemented in class:

```
public class CircularQueue<E> implements Queue<E> {
    private final int MAX_SIZE = 6; // max number of elements that can be held
    private int indFront = 0; // index of "front of the line", or 0 if empty
    private int indRear = -1; // index of "back of the line", or 0 if empty
    private E[] array = (E[]) new Object[MAX_SIZE];
//… methods will go here…
}
```

a. (5 points) Implement an isEmpty method that returns true if the queue is currently empty

b. (5 points) Write a helper method willBeEmpty that is `true` if (and only if) removing another item from the queue will make the queue empty.

```
private boolean willBeEmpty() {

}
```

c. (10 points) ***Draw*** the contents of `array` after offering 1 5 3 6 2 to a `CircularQueue<Integer>` in that order, calling poll twice, then offering 8 9 4. Your diagram should make clear what the index and contents of each element of the queue are. ***Show*** your work.

d. (5 points) ***Write*** whether the CircularQueue behaves any differently from a LinkedQueue from the client code's perspective. ***Explain*** your answer.

10. (10 points) Consider the expression represented by the recursive sequence
    $a_n = a_{n-1} + 5 \quad n > 1$
    $a_1 = -4$
    Write a recursive method to compute a(n)

11. a. (5 points) ***Determine*** how many times the recursive method below will be called while evaluating `m(2)`, include the first call. (Note: This is not the most efficient recursive implementation.) ***Show*** your work.

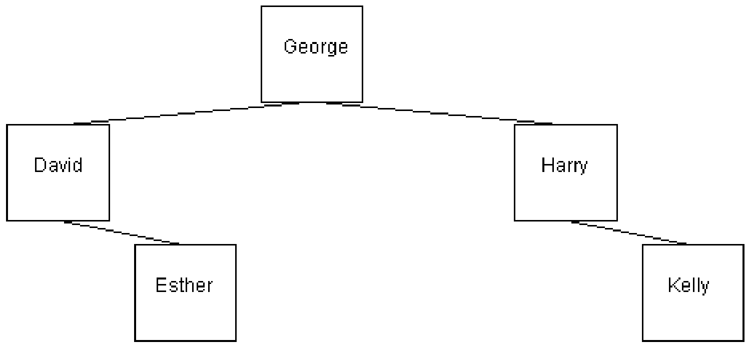
```
public int m(int x) {
  if (x <= 1) {
     return 1;
  } else {
     return m(x-1)-m(x-2);
  }
}
```

    b. (5 points) ***Determine*** what the return value of `m(2)` will be. ***Show*** your work.

12. (5 points) Consider a binary search of an array. With *n* comparisons (for some *n* that you don't need to know), you can search 16,383 elements. ***Determine*** how many elements you can search with *n*+1 comparisons. ***Show*** your work.

13. (5 points) A Binary Search Tree and a Linked List have a similar structure. ***Explain*** the advantage of a Binary Search Tree's structure.

14. (10 points) ***Insert*** Alice, Fred, Joe, and Mike into this Binary Search Tree without changing any non-null references.



15. (10 points) ***Implement*** the `size()` method of a Binary Search Tree, like the one we defined in class. If you call any other methods of this class, write the method as part of your solution.

```
public class BinarySearchTree<E extends Comparable<E>> {
    private Node root;
    private class Node {
        private Node left;
        private Node right;
        private E value;
        private Node(Node left, Node right, E value) {/* … */}
    }
















}
```