[**Closed book and notes.**] Show all of your work clearly in the space provided or on the additional page at the end of the exam. If the additional page is used, clearly identify to which exam question it is related. Be sure to **read each problem carefully**. Note that the exam is double sided.

1. (a) (10 points) Consider the following implementation of the clone() method for the Roster class (implemented in lab 3). Identify any errors (explaining what is wrong) and fix them.

```java
/**
 * Creates a deep copy of the Roster object.
 *
 * @return Reference to a new Roster object which is an identical (deep)
 * copy of this object, or null if CloneNotSupportedException was detected
 */
public Roster clone() {
  Roster roster = new Roster();
  roster.size = size;
  roster.capacity = capacity;
  roster.students = new Student[capacity];
  for(int i = 0; i < capacity; i++){
    roster.students[i] = students[i].clone();
  }
  return roster;
}
```

Answer:

```java
public Roster clone() {
  Roster roster = null;                // -4 make null here, then use super.clone()
  try {                                // -2 use try/catch
    roster = (Roster) super.clone();
    roster.size = size;
    roster.capacity = capacity;
    roster.students = new Student[capacity];
    for(int i=0; i<size; i++)          // -1 use SIZE not CAPACITY
    {
      roster.students[i] = students[i].clone();
    }
  } catch (CloneNotSupportedException e) {
    roster = null;
  }
  return roster;
}
```

(b) (5 points) Using big-oh notation, describe the overall worst case time complexity for the clone() method **before** your corrections. Suppose that the size of the Roster object is determined by the number of Student objects contained within it, identified as n. Be sure to explain your reasoning and state any additional assumptions that you make.

Answer:

Creating new array takes $O(n)$ time, we need to clone n students and it takes a constant amount of time for each clone. Since these two things happen in sequence, the whole method is $O(n)$.

(c) (5 points) Using big-oh notation, describe the overall worst case time complexity for the clone() method **after** your corrections. Suppose that the size of the Roster object is determined by the number of Student objects contained within it, identified as n. Be sure to explain your reasoning and state any additional assumptions that you make.

Answer:

$O(n)$ — for the same reasons as part b.

2. (10 points) Give an example of a method that $O(n)$ for the ArrayList and $O(1)$ for the LinkedList. Explain why (discuss the internal structure of each container that causes the specific time complexity).

Answer:

list.add(0, value); is $O(n)$ for an ArrayList since each element in the underlying array must be shifted to the right by one.

list.add(0, value); is $O(1)$ for a LinkedList since adding an element to the front of the list involves the same amount of work regardless of how many elements are contained in the list: allocate space for new entry, connect current head (if one exists) to the new entry, connect head (and potentially tail) to point to the newly created entry.

3. (10 points) Give an example of a method that $O(1)$ for the ArrayList and $O(n)$ for the LinkedList. Explain why.

Answer:

list.get(list.size()/2); is $O(1)$ for an ArrayList since getting to the middle element just involves adding an offset to the memory address denoting the beginning of the underlying array.

list.get(list.size()/2); is $O(n)$ for a LinkedList since we need to walk through the first $n/2$ elements to get to the middle one.

4. Consider the following partial implementation of the SinglyLinkedIterator that could be used with the singly linked list we developed in lecture.

```
private class SinglyLinkedIterator implements Iterator<Integer> {
  private int index;

  private LinkedRosterIterator() {
    index = -1;
  }

  public boolean hasNext() {
    boolean hasNext = true;
    try {                   // Try to get the next element
      get(index + 1);
    } catch(IndexOutOfBoundsException e) {
      hasNext = false;  // Return false if unable to get the next element
    }
    return hasNext;
  }

  public Integer next() {
    Integer value = null;
    if(hasNext()){
      ++index;
      value = get(index);
    }else{
      throw new NoSuchElementException("Iteration has no more elements");
    }
    return value;
  }
}
```

(a) (10 points) Using big-oh notation, describe the overall worst case time complexity for the hasNext() method. Be sure to explain your reasoning and state any additional assumptions that you make.

Answer:

$O(n)$ — hasNext() calls get(index+1) which must walk through index+1 elements.

(b) (10 points) Using big-oh notation, describe the overall worst case time complexity for the next() method. Be sure to explain your reasoning and state any additional assumptions that you make.

Answer:

$O(n)$ — Since next() calls hasNext(), it's at least $O(n)$, it then calls get(index). Since these happen one after the other, the total time is $O(n)$.

5. (10 points) In order to make use of the enhanced for loop (for-each loop), we needed to make the Roster class support iterators. List all of the steps required in order to allow the Roster class to support the enhanced for loop.

Answer:

- Have LinkedRoster implement Iterable
- Add iterator() method to LinkedRoster
- Add an inner class that implements the Iterator interface with: hasNext, next, and remove.

6. (10 points) Recall that the Java Stack class uses the Vector class (a class very similar to the ArrayList class) to do the real work associated with managing the data on the stack. Draw your best guess of how the data stored in memory would look after the following operations had been performed:

```
Stack<String> wordStack = new Stack<String>();
wordStack.push("Where");
wordStack.pop();
wordStack.push("there");
wordStack.push("is");
wordStack.peek();
wordStack.push("smoke");
wordStack.push("there's");
wordStack.push("a");
wordStack.push("fire");
wordStack.pop();
wordStack.pop();
wordStack.push("polution");
```

Answer:

7. In lecture, we implemented a portion a CircularQueue<E> class (see below).

```
public class CircularQueue<E> implements PureQueue<E> {

  private static final int CAPACITY = 32;
  private E[] queue;
  private int front;
  private int back;
  private boolean isFull;

  public CircularQueue() {
    queue = (E[])new Object[CAPACITY];
    front = 0;
    back = 0;
    isFull = false;
  }

}
```

(a) (10 points) Implement the isEmpty() method.

Answer:

```
public boolean isEmpty() {
  return front==back && !isFull;
}
```

(b) (10 points) Implement the enqueue() method.

Answer:

```
public boolean enqueue(E val) {    // 2 pts
  boolean added = false;
  if (!isFull) {                   // 2 pts
    queue[back] = val;             // 1 pt
    added = true;
    back = (back + 1) % CAPACITY;  // 2 pts
    if (back == front) {           // 2 pts
      isFull = true;
    }
  }
  return added;
}
```