CSCI 151
Exam 1 Solutions

The exam has 6 numbered questions that are worth 16 points each. You get 4 points for free, for a total of 100 points.

This is a closed-book, closed-notes, closed-Internet exam.

Please write only on the front side of each page; I won't see what you write on the backsides. The last page is blank and you can use that if you need more space.

After you have finished the exam, please indicate whether you followed the Honor Code on the exam.

I ☐ did ☐ did not

adhere to the Honor Code while taking this exam.

_____

Signature

1.

A. Suppose S is a Stack<Integer> that starts off empty and we do the following sequence of operations:

```
S.push(1);
S.push(2);
S.push(3);
S.pop();
S.push(4);
S.pop();
S.push(5);
```

With the same stack we do the following loop:

```
while (!S.isEmpty()) {
        System.out.print(S.pop() );
}
```

What will this loop print?

~~3~~ ~~4~~ 5
2
1

Push 1, then push 2 on top of 1
Then push 3 on top of 2.
The first pop pops 3 of the stack

Then push 4 on top of 2
The second pops removes the 4, then we push 5.

The stack has 5 on top of 2 on top of 1

Answer: 5 2 1

B. Next, we do this with a queue. Suppose Q is an empty queue and we do

```
Q.enqueue(1);
Q.enqueue(2);
Q.enqueue(3);
Q.dequeue();
Q.enqueue(4);
Q.dequeue();
Q.enqueue(5);
```

What will the following loop print?

```
while (!Q.isEmpty()) {
        System.out.print( Q.dequeue() );
}
```

~~1~~ ~~2~~ 3 4 5

Start by enqueing 1, then 2, then 3

The first dequeue removes 1; now 2 is at the front.

We enqueue 4, so the queue is 2 3 4.

The second dequeue removes 2. The queue is now 3 4.

Finally we enqueue 5.

Answer: 3 4 5

2. Here are 3 blocks of code. **Give a Big-Oh analysis of each block** (i.e, is this O(n)? $O(n^2)$? O(log(n))?? etc). In each case A and B are arrays of integers that have at least n elements; there are no index-out-of-bounds errors here. Don't worry about what the code does, just count the number of steps each block takes.

A.
```
for (int i = 0; i < n; i++){
      int min = A[i]
      for (int j =i-1; j >= 0; j--){
            if (A[j] < min)
                  min=A[j];
      }
      B[i] = min;
      System.out.println(B[i]);
}
```

The loop on i goes around n times.

For each value of i the j loop goes around i times.

So this does 1+2+3+…+n steps.

1+2+3+…+n is $O(n^2)$

B.
```
for (int i = 1; i < n-1; i++)  {
      int sum = 0;
      for(int j = -1; j <= 1; j++) {
            sum += A[i+j];
      }
      B[i] = sum/3;
      System.out.println(B[i]);
}
```

The loop on i goes around n-2 times.

For each value of i the loop on j does 3 steps: j gets the values -1, 0, and 1

Altogether this does O(n*3) = O(n) steps.

C.
```
for (int i = 0; i < n; i++) {
      B[i] = A[i]*A[i];
}
int sum = 0;
for (int j = 0; j < n; j++){
      sum = sum+B[j];
}
System.out.println(sum);
```

The loops here are sequential rather than nested. First the loop on i does n steps. After this is finished the loop on j does another n steps. So this is O(n+n) = O(n).

3. In Lab 3 we implemented MyQueue<E> as a linked structure, and we implemented MyStack<E> with an underlying ArrayList. We could have implemented MyQueue<E> with an ArrayList, where we enqueue an item by adding it at the end of the list and we dequeue by removing and returning the index 0 item. **Explain why this ArrayList implementation of queues is not good for applications where we will have a large number of elements in the queue.**

When you remove the index 0 element from an ArrayList you have to shift all of the remaining elements down one index. This makes dequeue a O(n) operation, where n is the number of elements in the queue. With linked structures we can make all of the queue operations O(1).

Some people pointed at the resizing that happens when you add data to an ArrayList. If you just make the original capacity of the ArrayList (the array size) large enough you can avoid the resizing, but you can't get away from the shifting; that is built into the idea of an Arraylist.

4. I have a list L with String data; each item in the list is a person's name. The methods of my List class are size( ), add(String x), add(int index, String x), remove(int index), set(int index, String x) and get(int index). These all work just the way you implemented them for the ArrayList class in Lab 2. **Give an algorithm in English** (sure, you can give code if you prefer) **for reversing the list**, so if we start with the list {"John", "Paul", "George", "Ringo"}, after we call reverse(L) L will be the list {"Ringo", "George", "Paul", "John"}

One way to do this is to put index low at 0 and index high at index size()-1. Then while (low < high) flip the data at index low and high, increment low and decrement high. Note that if you only use index low (so high is size-1-low) you have to stop when you get to the middle of the list or you will flip every element twice and have the original order back.

Another way to do this is to make a second list temp. Walk backwards through L (from index L.size()-1 to index 0, adding each element onto temp. Or you could walk forwards through L adding each element to temp at index 0. Either way, list temp is now the reversal of L. So copy temp back over L with

```
for(int i=0; i < L.size(); i++)
    L.set(i, temp.get(i));
```

Yet another way (and what I expected most people to do; it is what you did in Lab 3) is to make a stack of strings S. Walk through L pushing each element onto S. Then walk through L again setting each element to the value popped from the stack. That last loop is

```
for(int i=0; i < L.size(); i++)
    L.set(i, S.pop());
```

5. Remember the ArrayList<E> class that you implemented in Lab 2. This was based on an array E[ ] data and an integer size. **Write a new method**

boolean addIfNew(E x)

that adds item x to the list only if the list does not already contain x. If x is added addIfNew(x) should return true; if x is already in the list addIfNew(x) should return false. So if the list L is {5, 10, 15} L.addIfNew(10) returns false and the list remains {5, 10, 15}. On the other hand, L.addIfNew(20) returns true and the list becomes {5, 10, 15, 20}. Note that you can assume all of the other ArrayList methods from Lab 2 are implemented: if you determine that the list does not already contain x you can add it by just calling add(x).

Note that this will be a method of a list class. You don't have a specific list L to work with. You can get around that by using *this*. I didn't take off points for it, but you really should use the .equals( ) method all objects have rather than == to test if two objects are the same.

```
boolean  addIfNew(E x) {
     for (int i = 0;  i < this.size( ); i++ ) {
               if (x.equals( this.get(i) ))
                         return false;
     }
     this.add(x);
     return true;
}
```

The most common mistake was to use this for the loop body:

```
if (x.equals( this.get(i) ))
          return false;
else {
     this.add(x);
     return true;
}
```

That returns true or false based only on the first element of the list; you don't really know if you want to add x and return true until you have gone through the whole list without returning.

6. Here is an abstract class that holds two data values of type E. I call these values *first* and *second*:

```
abstract class Pair<E>{
        abstract void setFirst(E x);
        abstract void setSecond(E y);
        abstract E getFirst( );
        abstract E getSecond( );

        public Pair(E x, Ey) {
             setFirst(x);
             setSecond(y);
        }

        public int size( ) {
             return 2;
        }
}
```

Give a concrete subclass of Pair<E>:

`class myPair<E> extends Pair<E>`

Note that there are many ways to do this; you just need to find one way that works.

```
class myPair<E> extends Pair<E> {
      E first, second;

      myPair(E x, Ey) {
          super(x, y);
      }

      void setFirst(E x){
          first = x;
      }

      void setSecond(E y){
          second = y;
      }

      E getFirst( ) {
           return first;
      }

      E getSecond( ) {
          return second;
      }
}
```

There are two issues for this question. One is that you have to make some place to hold two data values of type E. I think using two variables is the easiest way, but you could also use one array of length 2. You can use an ArrayList but that is trickier; you would need to add two placeholder values into the ArrayList when you construct it, then use set(0, x) or set(1, y) for setFirst and setSecond. The ArrayList's add( ) method will get you into trouble if you try to use it for setFirst and setSecond.

The other issue is that you must give a constructor for class myPair. Since there is a constructor for the abstract class you can't make any kind of Pair without calling the constructor, and that can only happen if myPair has a constructor.