This exam is open book and open notes.

Please check that you have all pages, numbered 1 through 11. Write your name on each piece of paper, in case the pages become separated.

Write your answers concisely and legibly directly on this exam. Do not use extra sheets of paper.

Do your own work. No discussion or collaboration with other students is permitted.

If a question is unclear to you, please raise your hand and somebody will come to help you.

The exam is divided into parts as described below. By each section heading, an estimate of the time required to complete that section is provided to help you pace yourself. If you get stuck on a question, don’t spend too much time on it. Go on to the next question and the answer may occur to you later.

Partial credit will be given where appropriate. If you see how to approach a problem but don’t see the final answer, be sure to at least write down your approach.

<table>
<thead>
<tr>
<th>Section</th>
<th>Score</th>
<th>Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Definitions</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2. Program Behavior</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3. Algorithms</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4. ADTs</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>5. Algorithm Design</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. Definitions (10 minutes – 10 points)

**ACROSS**
3) Such methods and instance variables can be accessed only within the defining class.
7) Last name of the English mathematician who founded computer science.
8) A Java type that defines only the API and no instance variables
10) An ADT whose API includes methods named isEmpty, enQ, and deQ
12) The process of hiding detail and generalizing functionality.

**DOWN**
1) At least one method in this kind of class appears in API-form but with no method definition
2) Such methods and instance variables are available in the defining class’s subclasses but not its package
4) This sorting algorithm employs a pivot element and recursion
5) The English phrase for “extends”
6) Java uses this area to store local variables
9) The (design) process of breaking a large problem into easily attached subcomponents (not recursion)
11) This ADT was used in Lab 9 and can also be used to represent sets
12) When instantiated, its slots are of a fixed number but are all initially null.
10) An ADT whose API includes methods named push, pop, and isEmpty
2. Java Behavior (10 minutes – 10 points)

A series of multiple-choice questions is given below. Circle the response that best answers each question. These can be tricky! Be sure to read them over carefully.

(a) (5 points) A Java program stops due to a NullPointerException. The line where the error is first reported appears below.

\[ a(p) . b(q) \]

Consider each of the following statements.

I. The variable \( p \) is null.

II. The value returned by \( a(p) \) is null.

III. The variable \( q \) is null.

Which of the above statements best explains the behavior of the program?

A. I only
B. II only
C. III only
D. I and II only
E. I, II, and III

(b) (5 points) Consider the following class

```java
public class A {
    public void meth1() { }
    public void meth2() { }
    public void meth3() { }
}
```

Assuming that the methods shown above are the only methods that can be called on an instance of class \( A \), how many distinct\(^1\) interfaces could class \( A \) implement? Justify your answer.

---

\(^1\)Let’s say two interfaces are distinct if they do not offer exactly the same methods.
(c) (5 points) Consider the following Java code fragment, which uses \texttt{SetB}—the \texttt{Set}-implementing class based on ordered lists.

```
Set s = new SetB();
int i = 0;
while (i < 100) {
    s.insert(foo(i));
    i = i + 1;
}
int size = s.sizeOf();
```

It is observed that the running time of the program is more like $n^2$ or $(n/2)^2$ than like $n$ or $n/2$. Consider the following statements.

- **I.** For all $i$ and $i + 1$, $\text{foo}(i) < \text{foo}(i + 1)$
- **II.** At the end of all the insertions, $\text{size}$ has value 50
- **III.** For all $i$, $\text{foo}(i)$ always returns the same value

Which of the above statements could explain the observed running time?

A. I only  
B. II only  
C. III only  
D. I and II only  
E. I, II, and III

---

3. Algorithms (30 minutes – 30 points)

(a) (5 points) QuickSort

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>7</td>
<td>49</td>
<td>63</td>
<td>50</td>
<td>54</td>
<td>52</td>
</tr>
</tbody>
</table>

Above you see an array of integers $A$ which has 7 elements. As shown, the array represents the effects of QuickSort \textit{after just one partitioning operation}, when the entire array was considered and the array’s elements were rearranged with respect to the pivot.

Given the array as shown, what must the pivot have been for this first partitioning operation?

Circle that array element above.
(b) (20 points) The code for Binary Search is shown below.

```c
int bsearch(int[] nums, targ) {
    int lo = 0;
    int hi = nums.length;
    while (lo < hi) {
        int mid = (lo + hi)/2;
        if (targ < nums[mid]) hi = mid - 1;
        if (targ > nums[mid]) lo = mid + 1;
        if (targ == nums[mid]) lo = hi = mid;
    }
    if (nums[lo] == targ) return(lo);
    else return(-1);
}
```

In this problem, we generalize binary search as follows

```c
int gBsearch(int[] nums, minOfRange, maxOfRange) {
    // return any index i, 0 <= i < nums.length, such that
    // minOfRange <= nums[i] <= maxOfRange
    // If nothing in nums fits that description,
    // return -1
}
```

For example, suppose we have an array of numbers as follows.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>107</td>
<td>125</td>
<td>337</td>
<td>451</td>
<td>601</td>
<td>700</td>
<td>725</td>
<td>800</td>
</tr>
</tbody>
</table>

- If we invoke `gBsearch(nums, 115, 450)` then the method could return any one of the following:
  - the method could return 2, because $115 \leq nums[2] \leq 450$
  - the method could return 3, because $115 \leq nums[2] \leq 450$

It doesn’t matter which index of the two is returned, so long as the value in nums at that index falls inclusively in the specified range.

- If we invoke `gBsearch(nums, 24, 84)`, -1 would be returned.

- If we invoke `gBsearch(nums, 105, 124)`, the method is obligated to return 1, since $105 \leq nums[1] \leq 124$ and no other `nums[i]` satisfies that.
i. (5 points) Using the principle of *reduction*, fill in the method below for regular binary search so that it calls `gBsearch`.

```
int bsearch(int[] nums, int targ) {
    return
}
```

ii. (15 points) Below, fill in the method. To receive full credit, your algorithm should on-average behave like regular binary search, taking about $\log n$ steps to return an answer on an array of length $n$.

```
int gBsearch(int[] nums, int minOfRange, int maxOfRange) {
    return
}
```
4. ADTs (30 minutes – 35 points)

A **MultiSet** is like a **Set**, except that elements can be repeated. Consider the following multiset.

\[ \{ x, y, x, z, x, z \} \]

Unlike a set, the elements \( x \) and \( z \) are repeated. We say that \( x \) occurs 3 times in the multiset; \( z \) occurs twice; \( y \) occurs once; and \( w \) occurs 0 times (is not in the set).

Consider the following API and **interface** for **MultiSet** given below. The API is thinner than what we’ve used in class for **Set**—do not worry about methods not mentioned in the API below.

```java
public interface MultiSet {
    public void insert(Element x);
    public void delete(Element x);
    public int numOccurencesOf(Element x);
    public boolean hasElement(Element x);
}
```

(a) **(5 points)** Using the principle of **reduction**, complete the abstract class definition below.

```java
abstract public AnyMultiSet implements MultiSet {
    abstract void insert(Element x);
    abstract void delete(Element x);
    abstract int numOccurencesOf(Element x);

    public boolean hasElement(Element x) { // fill in below
}
}
```
(b) (10 points) Complete the class `MultiSetA` that implements `MultiSet` using an *unordered* list.

To receive full credit:
- Your implementation of `insert` should take constant time.
- Your implementation of `numOccurrencesOf` should take time proportional to the size of the set.

```java
public class MultiSetA extends AnyMultiSet {

    private List<Object> ls;
    public MultiSetA() {
        ls = new List<Objects>();
    }

    public void insert(Element x) { // fill this in

    }

    public void delete(Element x) {
        ls.reset();
        while (!ls.atEnd()) {
            Element e = (Element) ls.getItem();
            if (e.equals(x)) ls.delete();
            else ls.next();
        }
    }

    public int numOccurrencesOf(Element x) { // fill in

    }

```
(c) (10 points) Complete the class `MultiSetB` that implements `MultiSet` using an ordered list.

```
public class MultiSetB extends AnyMultiSet {

    private List0fObjects ls;
    public MultiSetB() {
        ls = new List0fObjects();
    }

    public void insert(Element x) { // fill this in

    }

    public void delete(Element x) { // fill this in

    }

    public int numOccurrencesOf(Element x) { // fill in

    }
}
```

(d) (10 points) Answer the following questions about cost.

i. What is the cost on average of your `insert` for `MultiSetA`?

ii. What is the cost on average of your `insert` for `MultiSetB`?
5. Algorithm Design (20 minutes – 20 points)

(a) (8 points) You are given a ListOfObjects that contains Elements, and there may be duplicates in the list. In fact, such a list could be the ls instance variable of the MultiSetA class in problem 4b. Thus, the duplicates do not occur in any particular order in the list.

You are charged with processing such a list into an array, such that no element occurs twice in the resulting array. Your solution will be graded on its elegance (the shorter, the better) and your judicious use of ADTs covered in class.

```java
public Element[] noDups(ListOfObjects ls) { // fill this in
}
```

(b) (12 points) Provide pseudocode or Java code for a Queue ADT implemented in terms of a Relation.
Due by end of exam period