

Exam II

Given: 5 May 2000

Due: end of exam period

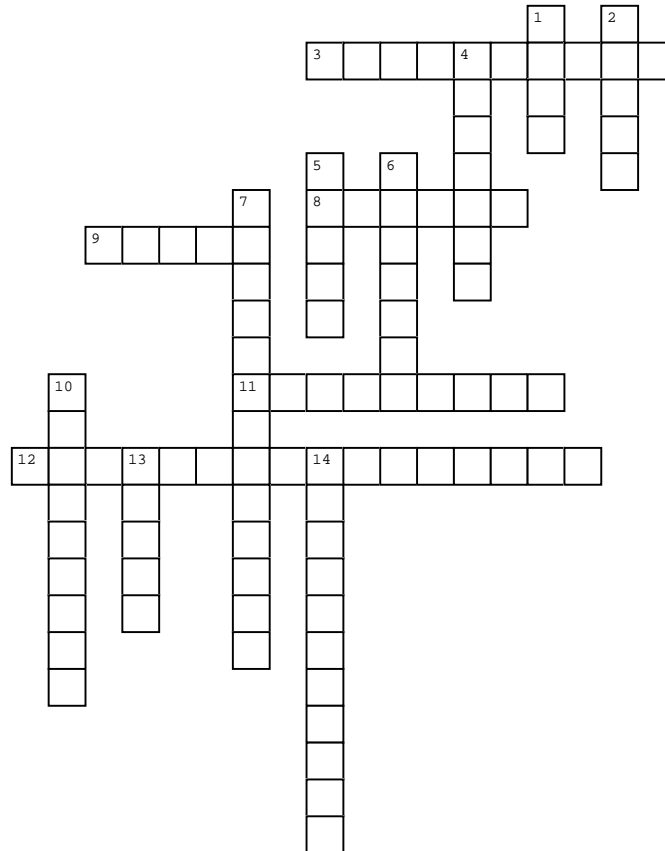
Name:

Lab Section:

- Bob will hold administrative office hours Sunday, May 7, in Lopata 408 (graders' office). Your exam will not be graded by then, but if you have any grading issues that need to be resolved, please stop by and see Bob.
- This exam is open book and open notes.
- Please check that you have all pages, numbered 1 through 15. 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.

Section	Score	Possible
1. Definitions		10
2. Program Behavior		15
3. Algorithms		30
4. ADTs		25
5. Design and APIs		20
TOTAL		100

1. Definitions (10 minutes – 10 points)

**ACROSS**

- 3) This Java keyword is used for "is-also-a".
- 8) Last name of the English mathematician who founded computer science.
- 9) An ADT whose first returned item is the first one inserted.
- 11) Such methods and instance variables are available in the defining class's subclasses but not its package
- 12) This data structure is useful for representing sets. Elements can be inserted or deleted in $\log(n)$ time; intersection takes $n \log(n)$ time.

DOWN

- 1) Java uses this area to store instantiated objects
- 2) An ADT whose first returned item is the one most recently inserted.
- 4) This Java keyword is used for "is-a".
- 5) Java uses this area to store local variables
- 6) Such methods and instance variables can be accessed only within the defining class
- 7) The (design) process of breaking a large problem into easily attacked subcomponents (not recursion)

2. Java Behavior (15 minutes – 15 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) When the following line of Java code is executed, the program stops because a `NullPointerException` is thrown.

```
p.foo(q)
```

Consider each of the following statements.

- I. The variable `p` is null.
- II. The variable `q` is null.
- III. There is no method `foo` that can be called on `p`.

Which of the above statements 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) When the following line of Java code is executed, the program stops because a `ClassCastException` is thrown.

```
A a = (A) (new B());
```

Consider each of the following statements.

- I. `A` is an interface and `B` implements `A`.
- II. `B` is a superclass of `A`.
- III. `A` is an abstract superclass of `B`.

Which of the above statements could be true given the program's behavior?

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

Continued on next page...

- (c) (5 points) Consider the following Java code fragment. (The relevant code from `SetA` is reproduced at the bottom of this page.)

```
Set s = new SetA();
s.insert(a);
s.insert(b);
Terminal.println("Size of the set is " + s.sizeOf());
```

Suppose the above code produces the following line of output:

```
Size of the set is 1
```

Consider each of the following statements.

- I. `a` and `b` are the same object.
- II. `a.elEquals(b)`
- III. `b.elEquals(a)`

If we assume that `a` and `b` implement `Element`, and *if we assume no particular implementation of their `elEquals`*, then which of the above statements can explain the behavior of the program?

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

```
// The list-based set, no ordering
public class SetA implements Set {
    private ListOfObjects ls;

    public SetA() { ls = new ListOfObjects(); }

    // positions the list at element x if it is there, else return null
    private ListOfObjects locate(Element x) {
        ls.reset();
        while (!ls.atEnd()) {
            Element curr = (Element) ls.getItem();
            if (curr.elEquals(x)) return ls;
            ls.next();
        }
        return null;
    }

    public void insert(Element x) {
        if (locate(x) == null) ls.append(x);
    }
}
```

3. Algorithms (30 minutes – 30 points)

(a) (18 points) QuickSort

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
23	14	49	47	52	54	53

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

- i. (6 points) Given the array as shown, what must the pivot have been for this first partitioning operation?

Circle that array element above.

- ii. (12 points) Now consider a second partitioning step, which will rearrange the elements to the *left* of the pivot element you chose above.

- A. (4 points) What pivot element do you choose for this next partitioning step?

- B. (8 points) Below, show the contents of the array after this next partitioning step, based on the element you have chosen for its pivot.

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>	<input style="width: 40px; height: 30px;" type="text"/>

(b) (12 points) Binary Search

Consider the extension of binary search to trinary search. The idea is that, instead of dividing the space in half each time, we'll divide it into thirds.

In the code shown on the next page, complete the partial assignments so that the code works correctly. In some cases, a change to a given variable may not be necessary. For example, if no change to `lo` is required, then just assign `lo = lo` so it's clear you didn't just leave the assignment blank.

```
public int find(int n) {

    int lo = 0;
    int hi = nums.length-1;

    while (lo < hi) {
        int oneThird = lo + (hi - lo) * 1/3;
        int twoThirds = lo + (hi - lo) * 2/3;

        if (n < nums[oneThird]) {

            lo =

            hi =

        }
        else if (n == nums[oneThird])
            lo = hi = oneThird;
        else if (n < num[twoThirds]) {

            lo =

            hi =

        }
        else if (n == nums[twoThirds])
            lo = hi = twoThirds;
        else {

            lo =

            hi =

        }
    }

    if (lo == hi && nums[lo] == n) return(lo);
    else return(-1);
}
```

4. ADTs (30 minutes – 25 points)

You may want to rip this page out—go for it.

- (a) (10 points) A `LimitedSet` is like a `Set`, except that it has an upper limit on how many elements can ever be in the set at one time. Consider the following API for `LimitedSet` and `Element`.

```
public interface LimitedSet {
    public int maxSizeOf();
    public void insert(Element x);
    public void delete(Element x);
    public boolean hasElement(Element x);
    public int sizeOf();
    public boolean isEmpty();
}
```

```
public interface Element {
    public boolean elEquals(Element other);
}
```

A partial implementation is given below. All methods are implemented except `insert` and `delete`. Fill those in correctly.

```
public class ArraySet implements LimitedSet {

    private int maxsize, size;
    private Element[] contents;

    public ArraySet(int maxsize) {
        this.maxsize = maxsize;
        contents = new Element[maxsize];
        size = 0;
    }

    private int locOf(Element x) {
        for (int i=0; i < size; ++i)
            if (contents[i].elEquals(x)) return i;

        return -1; // if not found
    }

    public boolean hasElement(Element x) { return locOf(x) >= 0; }
    public boolean isEmpty()           { return sizeOf() == 0; }
    public int     maxSizeOf()         { return maxsize;      }
}
```

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

```
public void insert(Element x) {
```

```
}
```

```
public void delete(Element x) {
```

```
}
```

```
}
```

(b) (15 points) Binary Search Trees

When the data given in Problem 3a is finally sorted, the following array is obtained:

$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$
14	23	47	49	52	53	54

Suppose we insert these elements into a Binary Search Tree, in sorted order as shown above, from 14 through 54.

i. (2 points) Show the binary search tree that results from such insertion.

ii. (3 points) When a binary search tree is created in this manner, for an array of n elements, how much on average would it take to find an element in the tree, assuming that it *is* there? Express your answer in terms of n , the number of elements in the array.

iii. (3 points) How much time would it take to find an element in the tree, assuming the element is *not* there?

- iv. (2 points) The solution you show for Problem 4(b)i is probably not very balanced. In what order would you insert the elements of A to obtain the most balanced tree?

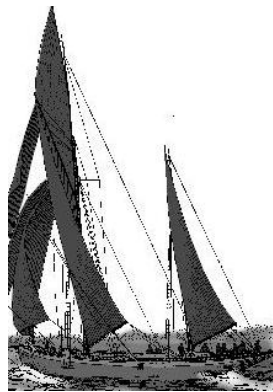
- v. (5 points) Devise a recursive algorithm that, when given a sorted array A , will insert its elements into a binary tree so as to obtain the bushiest (i.e., most balanced) tree possible. *Hint:* Think recursively, and use your solution from Problem 4(b)iv to guide your thinking. If you can't come up with the exact algorithm, state your approach for partial credit.

5. Design and API (30 minutes – 20 points)

You may want to rip this page out—go ahead, it will help on the following pages; besides, it will make you feel good.

Below you will find a text description of a class hierarchy.

- A **Sailboat** is a **Boat**.
- A **Motorboat** is a **Boat**.
- Every **Boat** has a
 - maximum speed;
 - length.
- Every **Sailboat** has some number of hulls.
- Every **Sailboat** has some number of sails.
- A **Sunfish** is a **Sailboat** with one hull and one sail.
- A **Catamaran** is a **Sailboat** with two hulls and two sails.
- A **Ketch** (shown below) is a **Sailboat** with one hull and three sails.¹



- Every **Motorboat** has a certain fuel efficiency, expressed as knots per gallon.
- A **Boat** is also **Taxable**.
- If something is **Taxable**, then we can find out its
 - purchase price;
 - year of purchase.

¹I know it looks like this one has four sails, but one of the sails shown is probably a spinnaker. In general, a ketch or yawl can have two or more jibs in addition to the mainsail; thus, in general, a ketch or yawl will have two or more sails. Let's just say it has three to simplify things, OK?

Below, and on the ensuing pages, you are to write a set of classes based on the above description. The phrases “has-a”, “is-a”, and “is-also-a” offer clues as to how this should be done. Also, the classes you are need are shown in a distinguished, sans-serif font, such as **Boat**.

- Show the API by writing class and interface declarations in Java.
- As always, every public method should be included—along with the method’s parameters and return type.
- You need not provide any code for the body of the methods, but indicate that code is necessary by writing “{...}”.
- Declare and describe any instance variables you need for the methods you provide.
- Be sure to fill in what a class extends, if not **Object**, and which interfaces, if any, a class implements.

Continued on next page...

Continued on next page...

Continued on next page...

Continued on next page...