Com S 228
Spring 2015
Exam 1

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: ____

ISU NetID (username): ____

Recitation section **(please circle one):**

1. R 10:00 am (Monica, Chen-Yeou)
2. R 2:10 pm (Caleb B, Blake)
3. R 1:10 pm (Caleb V, Ryan)
4. R 4:10 pm (Yuxiang, Anthony)
5. R 3:10 pm (Jacob, Andrew)
6. T 9:00 am (Chen-Yeou, Brian)
7. T 2:10pm (Chris, Austin)

Closed book/notes, no electronic devices, no headphones. Time limit **60 minutes**. Partial credit may be given for partially correct solutions.

- Use correct Java syntax for writing code.
- You are not required to write comments for your code; however, brief comments may help make your intention clear in case your code is incorrect.

If you have questions, please ask!

Question	Points	Your Score
1	28	
2	22	
3	26	
4	24	
Total	100	

1. (28 pts) Refer to the class hierarchy on pages 13-15 to answer the questions below. (It helps to *peel off* pages 13-20 from your exam sheets for code lookup convenience and scratch purpose.) For each section of code, fill in the box stating one of the following:

- the output, if any, OR
- that there is a compile error (briefly explain the error), OR
- the type of exception that occurs at runtime

[Hint: It is helpful to sketch a UML diagram showing the class hierarchy.]

`Behavior b = new Locust(2, "Black");`
`b.move();`	

`Insect i = new Insect(3, "Green");`	

`Insect b = new Bee(1, "Golden-Black", "Lake");`
`System.out.println(b.getColor());`
`System.out.println(b.getSwarm());`	

`Mantis m = new Mantis(5, "Green");`
`m.move();`
`Insect i = m.preyOn();`
`System.out.println(i.getColor());`	

`Grasshopper g = new Locust(3, "Red");`
`Katydid k = (Katydid) g;`	

`Grasshopper g = new Katydid(2, "Green");`
`g.attack();`
`g = new Locust(3, "Black");`
`System.out.println(((Locust) g).antennae());`
`Behavior b = g;`	

`Insect k = new Katydid(2, "Green");`
`Grasshopper g = (Katydid) k;`
`Locust l = (Katydid) k;`	

`Insect i = new Mantis(4, "Yellow");`
`((Mantis) i).move();`
`((Mantis) i).preyOn().attack();`
`i = new Bee(1, "Golden-Black", "Hill");`
`((Bee) i).makeHoney();`	

2. (22 pts) Still refer to the same class hierarchy on pages 13-15.

a) (14 pts) For the Bee class, override the methods `equals()` and `clone()` from `java.lang.Object`. Please pay attention to the following requirements:

   i) Two Bee objects are equal (i.e., `equals()` will return `true` when called with them as arguments) if they have the **same** `color` and `size`, and are from the **same** `swarm`.

   ii) Since the superclass `Insect` of Bee is an abstract class, it does not support deep clone. You **cannot** call `super.clone()` when overriding `clone()` within Bee. Simply create a copy of the Bee object.

```
@Override
public boolean equals(Object o)    // 10 pts
{






















}
```

```
@Override
public Object clone()  // 4 pts
{















}
```

b) (8 pts) Write code for a class `InsectComparator` which compares by size two objects from the same or different subclasses of the `Insect` class. The class `InsectComparator` implements the Comparator interface, and determines the size of an `Insect` object by the value of its `size` field.

Start your implementation with filling in the blank that follows the class name below.

```
public class InsectComparator

        implements ______________________________________ // 2 pts
{
















}
```

3. (26 pts) Determine the worst-case execution time of each of the following methods as a function of the length of the input array(s). Express each of your answers as big-O of a simple function (which should be the simplest and slowest-growing function you can identify). For convenience, your analysis of each part has been broken down into multiple steps. For each step, you just need to fill in the blank a big-O function as the answer (in the **worst case** always).

a) (6 pts)

```
void methodA (int[] arr)
{
    int n = arr.length;
    int p = 0;
    for (int i = 0; i < n; ++i)
    {
        int j = n - 1;
        while (j >= i)
        {
            p = (p + arr[j]) % 128;
            --j;
        }
    }
 }
```

i) Number of iterations of the outer **for** loop: ______________________________

ii) Number of iterations of the inner **while** loop: ______________________________

iii) Worst-case execution time: ____

b) (8 pts) Assume that the method `foo()` takes $O(n)$ time and method `bar()` takes $O(n^2)$ time.

```
public static void methodB (int[] arr)
{
    int n = arr.length;
    foo(arr);
    while (n > 0)
    {
        bar(arr);
        foo(arr);
        n = n/2;
    }
 }
```

i) Number of iterations of the **`while`** loop: ______________________________

ii) Time per iteration: ____

iii) Total time for the **`while`** loop: ___________________________________

iv) Total worst-case execution time for `methodB`: ______________________

c) (6 pts)

```
public static int methodC(int[] arr, int i)
{
    if (i == 0) return arr[i];
    return arr[i] + methodC (arr, i-1);
}
```

Suppose `arr` has length `n`, where `n` is at least 1. Assume that we call `methodC(arr,n-1).`

i) Number of recursive calls to `methodC`: ______________________________

ii) Worst-case execution time: ______________________________________

d) (6 pts) Suppose `BinarySearch(x,D)` is a method that uses binary search to determine if `x` is contained in a sorted array D. `BinarySearch(x,D)` returns `true` if `x` is in D, and `false` otherwise. Consider the following algorithm, which takes two arrays A and B of length n and returns an array containing the common elements of A and B:

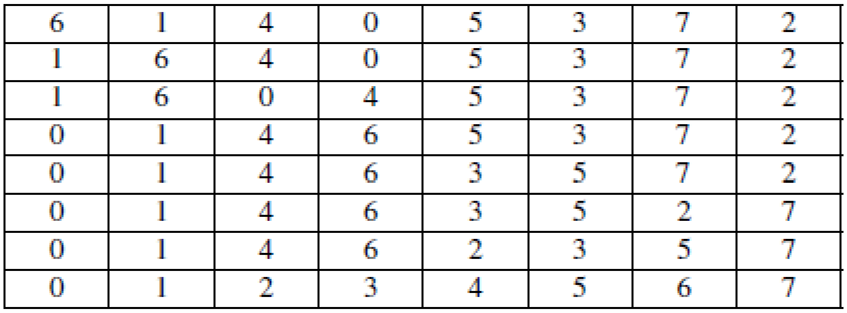
```
Intersection(A,B):
    sort A using mergesort
    sort B using mergesort
    for each i from 0 to n - 1
        found = BinarySearch(A[i], B)
        if found
            add A[i] to C
    return C
```

What is the big-O time complexity of `Intersection`? (For **partial credit**, to the right of each step of the algorithm write down the big-O time that it takes.)

4. (24 pts) The following tables list the input array (first line), output array (last line), and internal array state in sequential order for each of the sorts that we have studied in class (SELECTIONSORT, INSERTIONSORT, MERGESORT, and QUICKSORT). The two $O(n^2)$ sorts print internal results as the last operation of their outer loops. MERGESORT prints the output array after each call to MERGE. QUICKSORT prints after each call to PARTITION. Array contents occupy rows in the following tables, with the top rows containing the input and proceeding down through time to the output on the bottom.

There is **exactly one** right answer to each problem. For **partial credit**, please explain your reasoning in the space below.

a) (4 pts)



6	1	4	0	5	3	7	2
1	6	4	0	5	3	7	2
1	6	0	4	5	3	7	2
0	1	4	6	5	3	7	2
0	1	4	6	3	5	7	2
0	1	4	6	3	5	2	7
0	1	4	6	2	3	5	7
0	1	2	3	4	5	6	7

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning:

b) (4 pts)

6	1	4	0	5	3	7	2
1	6	4	0	5	3	7	2
1	4	6	0	5	3	7	2
0	1	4	6	5	3	7	2
0	1	4	5	6	3	7	2
0	1	3	4	5	6	7	2
0	1	3	4	5	6	7	2
0	1	2	3	4	5	6	7

A) SelectionSort B) InsertionSort C) MergeSort D) QuickSort

Reasoning:

c) (4 pts)

6	1	4	0	5	3	7	2
0	1	4	6	5	3	7	2
0	1	4	6	5	3	7	2
0	1	2	6	5	3	7	4
0	1	2	3	5	6	7	4
0	1	2	3	4	6	7	5
0	1	2	3	4	5	7	6
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

A) SelectionSort B) InsertionSort C) MergeSort D) QuickSort

Reasoning:

d) (4 pts)

6	1	4	0	5	3	7	2
1	0	2	6	5	3	7	4
0	1	2	6	5	3	7	4
0	1	2	3	4	6	7	5
0	1	2	3	4	5	7	6
0	1	2	3	4	5	6	7

A) SelectionSort B) InsertionSort C) MergeSort D) QuickSort

Reasoning:

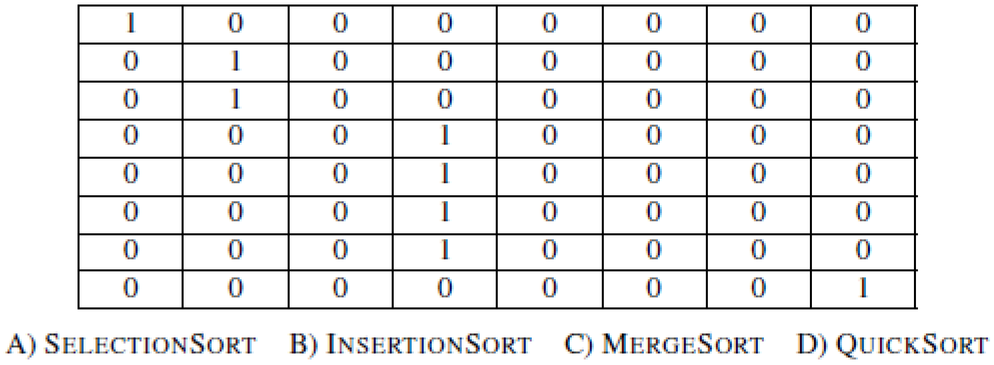
e) (4 pts)

9	3	3	1	8	4	8	5
3	3	1	4	5	9	8	8
3	3	1	4	5	9	8	8
1	3	3	4	5	9	8	8
1	3	3	4	5	9	8	8
1	3	3	4	5	8	8	9

A) SelectionSort B) InsertionSort C) MergeSort D) QuickSort

Reasoning

f) (4 pts)



1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1

A) SELECTIONSORT B) INSERTIONSORT C) MERGESORT D) QUICKSORT

Reasoning

Sample code for problems 1 and 2

```
public interface Behavior
{
      void move();
}


public abstract class Insect
{
      protected int size;                 // inches
      protected String color;

      public Insect(int size, String color)
      {
            this.size = size;
            this.color = color;
      }

      public int getSize()
      {
            return size;
      }

      public String getColor()
      {
            return color;
      }

      public abstract void attack();
}


public class Bee extends Insect implements Behavior
{
      private String swarm;

      public Bee(int size, String color, String swarm)
      {
            super(size, color);
            this.swarm = swarm;
      }

      public String getSwarm()
      {
            return swarm;
      }

      public void move()
      {
            System.out.println("fly");
        }

        @Override
        public void attack()
        {
            System.out.println("sting");
        }

        public void makeHoney()
        {
            System.out.println("Orange Blossom");
        }
    }


    public class Mantis extends Insect implements Behavior
    {
        public Mantis(int size, String color)
        {
            super(size, color);
        }

        public void move()
        {
            System.out.println("crawl");
        }

        @Override
        public void attack()
        {
            System.out.println("strike");
        }

        public Grasshopper preyOn()
        {
            return new Locust(3, "Brown");
        }
    }


    public abstract class Grasshopper extends Insect implements Behavior
    {
        public Grasshopper(int size, String color)
        {
            super(size, color);
        }

        public void move()
        {
            System.out.println("hop");
        }

        @Override
        public void attack()
        {
            System.out.println("bite");
        }

        public abstract String antennae();
    }


    public class Locust extends Grasshopper
    {
        public Locust(int size, String color)
        {
            super(size, color);
        }

        public String antennae()
        {
            return "Short";
        }
    }


    public class Katydid extends Grasshopper
    {
        public Katydid(int size, String color)
        {
            super(size, color);
        }

        public String antennae()
        {
            return "Long";
        }
    }
```

(Scratch only)

(Scratch only)

(Scratch only)

(Scratch only)

(Scratch only)