Com S 228
Fall 2015
Exam 1

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: _____

ISU NetID (username): _____

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

1. M 11:00 am (Alex, Kevin)
2. M 10:00 am (Caleb, Hyeyoung)
3. M 4:10 pm (Monica, John)
4. T 11:00 am (Ryan Y, Logan)
5. T 10:00 am (Blake, Kelsey)
6. T 4:10 pm (Andrew, Ryan K)

Closed book/notes, no electronic devices, no headphones. Time limit **75 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	22	
4	18	
5	10	
Total	100	

1. (28 pts) Refer to the class hierarchy on pages 15-16 to answer the questions below. (It helps to *peel off* pages 15-20 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.]

`Point p = new Point(0, 0);` `Shape s = new Shape(p);`	

`Point p = new Point(0, 0);`
`Region r = new Rectangle(3, 4, p);`
`System.out.println(r.perimeter());`	

`Solid c = new Cube(3, new Point(-1, -1));` `System.out.println(c.location());`	

`Point p = new Point(0, 0);`
`Shape s = new Cube(2, p);`
`System.out.println(s.area());`
`s = new Rectangle(3, 5, p) ;`
`System.out.println(s.area());`	

`Region r = new Square(2, new Point(3, 2));`
`Rectangle rt = (Rectangle) r;`
`System.out.println(rt.location());`	

`Square s = new Square(4, new Point(0,0));`
`Cube c = (Cube) s;`
`System.out.println(c.location());`	

`Shape s = new Rectangle(5, 2,`
`new Point(1, 1));`
`Square sq = (Square) s;`	

2. (22 pts) Below are the definitions of two classes named `Course` and `Instruction`.

a) (15 pts) For the `Course` class, override the methods `clone()`and `equals()` from `java.lang.Object`. Please pay attention to the following requirements:

i) Two `Course` objects are equal (i.e., `equals()` will return `true` when called with them as arguments) if they have **semantically equivalent** objects referenced by each of `lecture` and `recitation`.

ii) When overriding `clone()`, make a deep copy.

b) (7 pts) For the `Instruction` class, override the method `equals()` only. Two Instruction objects are equal if they have **semantically equivalent values** for each of `teacher`, `time`, and `room`.

```
public class Course implements Cloneable
{
      private Instruction  lecture;     // refers to a mutable object.
      private Instruction  recitation;  // refers to a mutable object.

      public Course(Instruction lec, Instruction rec)
      {
          if ( lec == null || rec == null )
            throw new IllegalArgumentException("null reference");
          lecture = lec;
          recitation = rec;
      }

      @Override
      public Course clone()  // makes a deep copy.
      {
           try
           {
               // TODO
```

```
            }

            catch (CloneNotSupportedException e)
            {
                // TODO









            }
        }

       @Override
       public boolean equals(Object obj)
       {
           // TODO









       }
}



public class Instruction implements Cloneable
{
       private String teacher;   // name of a person
       private int    time;      // meeting time for the class
       private String room;      // meeting place for the class
```

```
    public Instruction(String person, int mtime, String place)
    {
        if ( person == null || place == null )
           throw new IllegalArgumentException("null reference");
        teacher = person;
        time = mtime;
        room = place;
    }

    public Instruction clone()
    {
        try
        {
            Instruction cloned = (Instruction) super.clone();
            return cloned;
        }

        catch (CloneNotSupportedException e)
        {
            return null;
        }
    }

    @Override
    public boolean equals(Object obj)
    {
        // TODO









    }
}
```

3. (22 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 every answer as big-O of a simple function (which should be the **simplest** and **slowest-growing** function you can identify). For convenience, your analysis of every method 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 sum = 0;
    for (int i = 0; i < n; ++i)
    {
        int j = n - 1;
        while (j > i)
        {
            sum = sum + arr[j];
            j = j - 2;
        }
    }
}
```

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

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

iii) Worst-case execution time: _____

b) (4 pts)

```
public static int methodB(int[] arr, int i)
{
    if (i <= 0)
        return arr[0];
    return arr[i] + methodB(arr, i-3);
}
```

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

i) Number of recursive calls to `methodB`: _____

ii) Worst-case execution time: _____

c) (6 pts) Assume that the method `foo()` takes $O(n^3)$ time.

```
public static void methodC (int[] arr)
{
    int n = arr.length;
    while (n > 0)
    {
        if (n % 2 == 0)
        {
           n = 2 * (n / 4) + 1;
        }
        else
        {
           foo(arr);
           n = n – 1;
        }
    }
 }
```

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

ii) Time per iteration: _____

iii) Worst-case execution time for `methodC`: _____

d) (6 pts) The method `AdaptedMergeSort(A)` sorts an input array of size up to `1000000` using the insertion sort algorithm. If the array exceeds that size, the method then splits the array into two halves, recursively sorts each half, and finally merges the two sorted halves into one. Below is the pseudocode for this method.

```
AdaptedMergeSort(A)
    n = A.length
    if n <= 1000000
        perform insertion sort on A
    else
        m = n/2
        Aleft = (A[0] ,..., A[m-1])
        Aright = (A[m] ,..., A[n-1])
        AdaptedMergeSort(Aleft)
        AdaptedMergeSorg(Aright)
        merge Aleft and Aright into A so it becomes sorted.
```

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

4. (18 pts) Every table below lists the input array (first line), output array (last line), and internal array states in sequential order for one 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. It splits an array in **exactly the same way** as described in the pseudocode for `AdaptedMergeSort` in Problem 3d). QUICKSORT prints after each call to PARTITION.

Decide the sorting algorithm used on each problem. There is **exactly one** right answer, which you are asked to circle. For **partial credit**, please explain your reasoning in the space below.

a) (3 pts)

9	8	6	7	5	3	2	4	1	0
8	9	6	7	5	3	2	4	1	0
8	9	6	5	7	3	2	4	1	0
8	9	5	6	7	3	2	4	1	0
5	6	7	8	9	3	2	4	1	0
5	6	7	8	9	2	3	4	1	0
5	6	7	8	9	2	3	4	0	1
5	6	7	8	9	2	3	0	1	4
5	6	7	8	9	0	1	2	3	4
0	1	2	3	4	5	6	7	8	9

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

Reasoning:

b) (3 pts)

9	8	6	7	5	3	2	4	1	0
0	8	6	7	5	3	2	4	1	9
0	8	6	7	5	3	2	4	1	9
0	1	6	7	5	3	2	4	8	9
0	1	6	7	5	3	2	4	8	9
0	1	3	2	4	6	7	5	8	9
0	1	2	3	4	6	7	5	8	9
0	1	2	3	4	5	7	6	8	9
0	1	2	3	4	5	6	7	8	9

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

Reasoning:

c) (3 pts)

9	8	6	7	5	3	2	4	1	0
8	9	6	7	5	3	2	4	1	0
6	8	9	7	5	3	2	4	1	0
6	7	8	9	5	3	2	4	1	0
5	6	7	8	9	3	2	4	1	0
3	5	6	7	8	9	2	4	1	0
2	3	5	6	7	8	9	4	1	0
2	3	4	5	6	7	8	9	1	0
1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9

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

Reasoning:

d) (3 pts)

5	6	7	9	8	3	2	1	0	4
3	2	1	0	4	5	6	7	9	8
0	2	1	3	4	5	6	7	9	8
0	2	1	3	4	5	6	7	9	8
0	1	2	3	4	5	6	7	9	8
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

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

Reasoning:

e) (3 pts)

9	8	6	7	5	3	2	4	1	0
0	8	6	7	5	3	2	4	1	9
0	1	6	7	5	3	2	4	8	9
0	1	2	7	5	3	6	4	8	9
0	1	2	3	5	7	6	4	8	9
0	1	2	3	4	7	6	5	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

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

Reasoning

f) (3 pts)

5	6	7	9	8	3	2	1	0	4
5	6	7	9	8	3	2	1	0	4
5	6	7	9	8	3	2	1	0	4
5	6	7	9	8	3	2	1	0	4
5	6	7	8	9	3	2	1	0	4
3	5	6	7	8	9	2	1	0	4
2	3	5	6	7	8	9	1	0	4
1	2	3	5	6	7	8	9	0	4
0	1	2	3	5	6	7	8	9	4
0	1	2	3	4	5	6	7	8	9

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

Reasoning

5. (10 pts) Bubble sort is performed in multiple passes. In every pass, it steps through the entire array, comparing each pair of adjacent numbers and swapping them if the preceding number in the pair is **greater than** the succeeding one. If no swaps have been performed during a pass, the array is sorted and the algorithm terminates. Intuitively, the algorithm works in a way that smaller elements "bubble" to the front of the array.

Below is an example showing how bubble sort orders an input array of size 16 in 9 passes. (Some intermediate passes are omitted.) Each pass guarantees to put the next biggest element in place.

503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703 (input)
087 503 061 512 170 897 275 653 426 154 509 612 677 765 703 908 (pass 1)
087 061 503 170 512 275 653 426 154 509 612 677 765 703 897 908 (pass 2)
...

061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908 (pass 9)

a) (6 pts) Write out the content of the array at the end of pass 3.

b) (4 pts) Is bubble sort a stable sorting algorithm?

Sample code for problem 1

```
public class Point
{
	private int x, y;  // point (x, y) in the plane
	                   // or point (x, y, 0) in the space

	public Point(int a, int b)
	{
		x = a;
		y = b;
	}

	public String toString()
	{
		return "(" + x + ", " + y + ")";
	}
}

public interface Region
{
	int perimeter();
}

public interface Solid
{
	int volume();
}

public abstract class Shape
{
	private Point center;  // geometric center

	public Shape (Point c)
	{
		center = c;
	}

	Point location()
	{
		return center;
	}

	public abstract int area();
}

public class Rectangle extends Shape implements Region
{
	private int length;
	private int width;

	public Rectangle (int l, int w, Point c)
	{
		super (c);
		length = l;
```

```
            width  = w;
        }

        public int area()
        {
            System.out.print("Rectangle area: ");
            return length * width;
        }

        public int perimeter()
        {
            System.out.print("Rectangle perimeter: ");
            return 2 * (length + width);
        }
}

public class Square extends Rectangle
{
        private int sideLength;

        public Square (int s, Point c)
        {
            super(s, s, c);
            sideLength = s;
        }

        public int area()
        {
            System.out.print("Square area: ");
            return sideLength * sideLength;
        }
}

public class Cube extends Shape implements Solid
{
        private int sideLength;

        public Cube(int s, Point c)
        {
            super(c);
            sideLength = s;
        }

        public int volume()
        {
            System.out.print("Cube volume: ");
            return sideLength * sideLength * sideLength;
        }

        public int area()
        {
            System.out.print("Cube area: ");
            return 6 * sideLength * sideLength;
        }
}
```

(Scratch only)

(Scratch only)

(Scratch only)

(Scratch only)