Com S 228
Fall 2016
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 (Anthony, Chandler)
2. M 10:00 am (Robert, Yuxiang)
3. M 4:10 pm (Andrew, Ryan K)
4. T 11:00 am (Kevin, John)
5. T 10:00 am (Blake, Kelsey)
6. T 4:10 pm (Lige, Ryan Y)
7. W 4:10 pm (Alex, Susan)
8. W 11:00am (Mike, Jake)

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	27	
2	23	
3	26	
4	24	
Total	100	

1. (27 pts) Refer to the class hierarchy on pages 13-16 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 code snippet, fill in the box stating one of the following:

- the output, if any, OR
- that there are **one or more** compile errors (briefly explain them), OR
- the type of exception that occurs at runtime

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

`BigCat c = new SiberianTiger("Sergei", 600);` `c.eat();`	
`Tiger vijay = new BengalTiger("Vijay", 570);` `vijay.showStripes();`	

`Tiger t;`
`t = new Tiger("Vijay", 570);`
`t = new Lion("Mufasa", 550);`	

`Sound s = new Lion("Simba", 500);` `((Lion) s).speak();`	

`BigCat c = new Lion("Nala", 400);`
`Lion twin = ((Lion) c).makeClone();`
`System.out.println(twin.getName());`	

`Sound s;`
`BengalTiger b = new BengalTiger("Vijay",`
`570);`
`s = b;`
`Lion t = new Lion("Mufasa", 550);`
`t = (Lion) ((BigCat) s);`	

`Lion[] pride = new Lion[2];`
`pride[0] = new Lion("Simba", 500);`
`pride[1] = new Lion("Nala", 400);`
`LionPride serengeti = new LionPride`
`("Serengeti", pride);`
`System.out.println((serengeti.getLion`
`(serengeti.getSize()-2)).getName());`	

2. (23 pts) Still refer to the same class hierarchy on pages 13-16.

a) (8 pts) For the `Tiger` class, override the methods `equals()` from `java.lang.Object`. Please pay attention to the following requirements:

Two `Tiger` objects are equal (i.e., `equals()` will return `true` when invoked by one of them with the other one as an argument) if they are from the **same** `subspecies` (i.e., with the **same** dynamic type) and of **equal** weight, and have the **same** name.

```
@Override
public boolean equals(Object o)
{

}
```

b) (10 pts) For the `LionPride` class, override the method `clone()` to perform deep copying. You are **not allowed** to call the constructor. Note that the `Lion` class does not override `clone()`.

```
@Override
public Object clone()
{









}
```

c) (5 pts) Complete the code for a class `BigCatComparator` which implements the `Comparator` interface to compare by weight two objects from the same or different subclasses of the `BigCat` class. The weight of an object is the value of its `weight` field. The heavier object is considered greater.

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

```
public class BigCatComparator

        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 static methodA(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

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

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

iii) Time for *each* iteration of the inner **for** loop: _____

iv) Worst-case execution time: _____

b) (8 pts) Assume that the method `bar()` takes $O(n^2)$ time, method `foo1()` takes $O(n)$ time, and method `foo2()` takes $O(n \log n)$ time.

```
public static void methodB (int[] arr)
{
    int n = arr.length;
    bar(arr);
    while (n > 0)
    {
        foo1(arr);
        foo2(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/4);
}
```

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 B and C are two sorted arrays of integers. Assume that neither B nor C contain duplicates (there may, however, be common elements between B and C). The following algorithm returns an array that contains all numbers that are in B or in C, removing any duplicates.

```
Union(B,C)
    p = B.length, q = C.length
    create an empty array D of length p + q
    i=0; j=0
    while i < p && j < q
        if B[i] < C[j]
            append B[i] to D
            i++
        else if B[i] == C[j]
            append B[j] to D
            i++; j++
        else
            append C[j] to D
            j++
    if i >= p
        for r = j to q-1
            append C[r] to D
    else
        for r = i to p-1
            append B[r] to D
    return D
```

Suppose n = `B.length + C.length`. What is the big-O time complexity of `Union` as a function of n?

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 (*Selection Sort*, *Insertion Sort*, *Merge Sort*, and *Quicksort*). The two $O(n^2)$ sorts print internal results as the last operation of their outer loops. *Merge Sort* prints the output array after each call to *Merge*. *Quicksort* prints after each call to *Partition* (of the version presented in the lecture). 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)

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

A) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning:

b) (4 pts)

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

A) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning:

c) (4 pts)

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

A) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning:

d) (4 pts)

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

A) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning:

e) (4 pts)

7	2	1	0	3	4	5	6
0	2	1	7	3	4	5	6
0	1	2	7	3	4	5	6
0	1	2	7	3	4	5	6
0	1	2	3	7	4	5	6
0	1	2	3	4	7	5	6
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) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning

f) (4 pts)

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

A) Selection Sort B) Insertion Sort C) Merge Sort D) Quicksort

Reasoning

Sample code for Problems 1 & 2

```
public interface Sound
{
	void speak();
}


public class BigCat
{
	protected String name;
	protected int weight;

	protected BigCat(String name, int weight)
	{
		this.name = name;
		this.weight = weight;
	}


	public String getName()
	{
		return name;
	}


	public int getWeight()
	{
		return weight;
	}
}


public abstract class Tiger extends BigCat implements Sound
{
	public Tiger(String name, int weight)
	{
	    super(name, weight);
	}


	@Override
	public void speak()
	{
		System.out.println("Growl!");
	}
```

```
      public abstract void showStripes();
}


public class SiberianTiger extends Tiger
{
      private String diet;

      public SiberianTiger(String name, int weight)
      {
            super(name, weight);
            diet = "Wild Boar";
      }


      @Override
      public void showStripes()
      {
            System.out.println("Sparse");
      }


      public void eat()
      {
            System.out.println(diet);
      }
}


public class BengalTiger extends Tiger
{
      public BengalTiger(String name, int weight)
      {
            super(name, weight);
      }


      @Override
    public void showStripes()
    {
      System.out.println("Dense");
    }
}
```

```
public class Lion extends BigCat implements Sound
{
    public Lion(String name, int weight)
    {
        super(name, weight);
    }


    @Override
    public void speak()
    {
     System.out.println("Roar!");
    }


    public Lion makeClone()
    {
     return new Lion(name, weight);
    }
}


public class LionPride implements Cloneable
{
     private String name;
     private int size;
     private Lion[] lions;

     public LionPride(String name, Lion[] lions)
     {
          this.name = name;

          size = lions.length;
          this.lions = new Lion[size];
          for (int i = 0; i < size; i++)
          {
               this.lions[i] = lions[i].makeClone();
          }
     }


     public String getName()
     {
          return name;
     }
```

```
	public void setName(String name)
	{
		this.name = name;
	}


	public int getSize()
	{
		return size;
	}


	public Lion getLion(int i)
	{
		if (i < 0 || i >= size)
			return null;
		return lions[i];
	}
}
```

(Scratch only)

(Scratch only)

(Scratch only)

(Scratch only)