Com S 228
Spring 2014
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 (Chris, Caleb B)
2. R 2:10 pm (Bryan, Ben)
3. R 1:10 pm (Jesse, Monica)
4. R 4:10 pm (Caleb V, Brad)
5. R 3:10 pm (Kyle, Nick)
6. T 9:00 am (Brady, Ade)
7. T 2:10pm (Kyle, Shana)

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	26	
2	24	
3	24	
4	26	
Total	100	

1. (26 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

It helps to know that the liger is a hybrid cross between a male lion and a tigress, while the tigon is one between a male tiger and a lioness. [Hint: It is helpful to sketch a UML diagram showing the class hierarchy.]

`BigCat mufasa = new Lion("Mufasa", Sex.MALE);`
`mufasa.speak();`	

`Interspecies das = new Tiger("Das", Sex.MALE);`	

`IRoar kofi = new IRoar();`
`BigCat thema = new BigCat("Thema", Sex.FEMALE);`	

`BigCat sanjeev;`
`sanjeev = new Tiger("Sanjeev", Sex.MALE);`
`sanjeev.speak();`
`sanjeev = new Liger("Vijay", Sex.MALE,`
`                new Lion("Simba", Sex.MALE),`
`                new Tiger("Maha", Sex.FEMALE));`
`sanjeev.speak();`	

`IRoar nala = new Lion("Nala", Sex.FEMALE);`
`Tiger rita = (Tiger) nala;`	

`BigCat vijay, nala;`
`vijay = new Tiger("Vijay", Sex.MALE);`
`nala = new Lion("Nala", Sex.FEMALE);`
`Interspecies ife = new Tigon("Ife", Sex.MALE,`
`                    (Tiger) vijay, (Lion) nala);`
`ife.getParents();`	

`IRoar nala;`
`nala = new Lion ("Nala", Sex.FEMALE);`
`nala.getParents();`	

2. (24 pts) For the `Dictionary` class below, override the method `equals()` from `java.lang.Object`, and implement the method `makeClone()`. You just need to fill in the blanks.

```
public class Dictionary
{
      private String[] word;
      public Dictionary(String[] w)
      {
            word = w;
      }


      /* Two words are equal if they have the same content
       * or both are null. This method has been implemented
       * for you.
       */
      public static boolean equals(String w1, String w2)
      {
            if ( w1 == null && w2 == null)
                  return true;

            if ( (w1 == null && w2 != null) ||
                   (w1 != null && w2 == null) )
                   return false;

            // now we are sure both words are not null
            if (w1.equals(w2))
                  return true;

            return false;
      }
```

```
/* (12 Pts)
Two objects of type Dictionary are equal if their copies of the
private array word[] have the same length with identical String
contents at every index (or both array references are null). Two
array elements that both refer to null are considered equal.
*/
@override
public boolean equals(Object another)
{
      // TODO
















}
```

```
    /* (12 pts)
    This method returns a copy of the dictionary. All data (i.e.,
    word[]) of the copy must NOT share the memory with that of the
    original dictionary. In other words, any change of one dictionary
    (e.g., adding/removing/modifying a word) will not affect the
    other.
    */
    public Dictionary makeClone()
    {

          // TODO

    }


}
```

3. (24 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)

```
public static int methodA(int [] arr1, int [] arr2)
{
   int x = 0;
   for (int i = 0; i < arr1.length; i++)
      for (int j = i; j < arr2.length; j++)
         if (arr1[i] > arr2[j])
            x = x + arr1[i]
         else
            x = x + arr2[j]

   return x;
}
```

Suppose `arr1` and `arr2` have the same length `n`.

Number of iterations of the outer for loop: ______________

Number of iterations of the inner for loop: ____________________________

Worst-case execution time: ____________________

b) (6 pts)

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

}
```

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

Number of recursive calls to `methodB`: _____

Worst-case execution time: ___________________

c) (6 pts)

```
// assume that the method foo() takes
// O(n^2) time
public static void methodC(int[] arr)
{
   int n = arr.length;
   while (n > 0)
   {
      foo(arr);
      if (n % 2 == 0)
         n = n/2;
      else
         n = n/3;
   }
}
```

Number of iterations: _____________

Time per iteration: ______________________________

Worst-case execution time: ________________________

d) (6 pts) Consider the following algorithm, which takes two arrays A[ ] and B[ ] of length n consisting of integers, neither of which contains duplicates, and returns an array C[ ] containing all the elements from A[ ] and B[ ] but with no duplicates:

```
sort A using mergesort
sort B using mergesort
i = 0
j = 0
while i < n && j < n
   if A[i] < B[j]
      add A[i] to C
      i++
   else if B[j] < A[i]
       add B[j] to C
       j++
if i >= n
    append B[j],...,B[n-1] to C
else
    append A[i],...,A[n-1] to C
return C
```

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

4. (26 pts) We consider the insertion sort and quicksort. There are four parts a)-d).

a) (6 pts) Perform the insertion sort on the array `arr[]` below to arrange its elements in the non-decreasing order. The algorithm repetitively inserts `arr[i]` into its proper place amongst `arr[0]` through `arr[i]`. Fill in the entries of the array immediately **after each time** an element is inserted (or determined to stay) at its final position.

You may not need all rows of boxes. But add more rows if you need.

32	4	57	6	13	2

b) (2 pts) How many right shifts were performed during the sorting in a)?

c) Our version of quicksort always performs the insertion sort on an input array `arr[]` with three or fewer elements. When the array size exceeds three, the `partition` subroutine uses the **median** of the first three elements (indexed at `first`, `first+1`, and `first+2`) as the pivot. See the code for `partition` on page 16.

1. (2 pts) Given the input array below, the position `pivotIndex` of the pivot in the array **before** the first swap has the value

`pivotIndex ==`

`left: 0,  right: 9`

`arr:`

40	12	34	52	78	9	21	67	11	93
left									right

2. (12 pts) Trace the execution of one call to the `partition` method, **exactly as it is written**, over the above array `arr[]`.

- Show the contents of the array, along with the positions of `i` and `j`, just **before** the main `while` loop starts. [It helps to draw a square around the current location of the pivot element.]

- Each time **point A** is reached, do the following:

  a. write out the contents of the array in a separate row, showing the positions of `i` and `j`, just **before** swapping, and then

  b. **circle the elements that will be swapped.**

- If a swap happens **after** the main `while` loop, show the contents of the array just **before the swap** into a separate row, and **circle those elements to be swapped**. Also, show the positions of `i` and `j` at the moment.

- Show the contents of the array when the partition step is complete, and **draw a square around the final location of the pivot element**.

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□

d) (4 pts) Describe a worst case execution scenario on an input array of n elements. What is the running time in Big-O notation for this case?

Sample code for problem 1

```java
public enum Sex
{
      FEMALE, MALE
}


public interface IRoar
{
      void speak();
}


public interface Interspecies
{
      void getParents();
}


public abstract class BigCat implements IRoar
{
      protected String name;
      protected Sex sex;

      protected BigCat(String name, Sex sex)
      {
            this.name = name;
            this.sex = sex;
      }

      public String getName()
      {
            return name;
      }

      public abstract void speak();
}


public class Lion extends BigCat
{
    public Lion(String name, Sex sex)
    {
        super(name, sex);
    }

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


public class Tiger extends BigCat
{
      public Tiger(String name, Sex sex)
      {
          super(name, sex);
      }

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


public class Liger extends BigCat implements Interspecies
{
      private Lion dad;
      private Tiger mom;

      public Liger(String name, Sex sex, Lion dad, Tiger mom)
      {
            super(name, sex);
            this.dad = dad;
            this.mom = mom;
      }

      public void getParents()
      {
            System.out.println("Dad: Lion" + " ("
                                    + dad.getName() + ")");
            System.out.println("Mom: Tiger" + " ("
                                    + mom.getName() + ")");
      }


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

```
public class Tigon extends BigCat implements Interspecies
{
      private Tiger dad;
      private Lion mom;

      public Tigon(String name, Sex sex, Tiger dad, Lion mom)
      {
            super(name, sex);
            this.dad = dad;
            this.mom = mom;
      }

      public void getParents()
      {
            System.out.println("Dad: Tiger" + " ("
                                    + dad.getName() + ")");
            System.out.println("Mom: Lion" + " ("
                                    + mom.getName() + ")");
      }

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

}
```

Sample code for problem 4

```
private static int partition(int[] arr, int first, int last)
{
      int pivotIndex;  // initial index of the pivot

      // Code that sets pivotIndex to be the index of the median of
      // arr[first], arr[first+1], and arr[first+2].
      //
      // ...

      swap(arr, first, pivotIndex);
      int pivot = arr[first];

      int i = first + 1;
      int j = last;

      while (i <= j)
      {
            while (i <= last && (arr[i] < pivot))
                  ++i;
            while (j > first && (arr[j] >= pivot))
                  --j;
            if (i < j)
            {
               swap(arr, i, j); // <--- point A
            }
      }

      if (j != first)
          swap(arr, first, j);

      return j;
}
```

(Scratch only)

(Scratch only)

(Scratch only)

(Scratch only)