CS151 Midterm 1

Name:

Start Time:

Finish Time:

Accommodation (if applicable):

I have abided by the Honor Code. I have not discussed this test with anyone.
(Sign below)

If you take this test on separate sheets of paper, put all of the items above on your first page. If you need more space, feel free to add extra pages. Just make sure everything is well labelled.

There are 5 questions in this test. All questions have equal point values. They may not be of equal difficulty. As indicated in the table below some questions have several parts. Be sure to answer all parts of all questions.

Question	**parts**	**points**
1	1 (or 4)	20
2	2	20
3	(3 or 7 depending on how you count)	20
4	2	20
5	2	20
Possible Points:		100

Question 1 (20 points — each of the 4 functions is 5 points). Write a class that implements the interface documented below.You may assume that SepChainHT and List151Impl are exactly as discussed and as they appear in these URLS: https://cs.brynmawr.edu/cs151/L05/List151Impl.java.txt, https://cs.brynmawr.edu/cs151/L08/SepChainHT.java.txt Also you may assume that their implementations are in the directory in which you are writing your class. No documentation is expected.

```
public interface SignOfFour {
    /**
     * return the number 4
     * @return the number 4;
     */
    public int number();
    /**
     * Add 4 to the provided number
     * @param num the number to which 4 is to be added
     * @return the provided number + four
     */
    public int add(int num)

    /**
     * Returns an instance of a Separate Chaining hashtable containing
     * pairs of items in which the key and value are identical.
     * If the count is a less than or equal to zero then an empty
     * hashtable should be returned.
     * If greater then zero, then the contents of the hashtable
     * should be multiples of 4. For instance, given 3, then the
     * returned hashtable will contain the following pairs:
     * <4,4>, <8,8>, <12,12>
     * @param count the number of multiples of 4 to put into
     *        the hashtable
     * @return a hashtable, with contents as described above
     */
    public SepChainHT<Integer, Integer> makeHT(int count);

   /**
     * Return an instance of a List151Impl containing powers of 4.
     * If the count is less than or equal to zero, return null.
     * If positive, return the powers of 4 (starting at 1).
     * For instance, if the provided count is 5, then the
     * returned list should contain: 1, 4, 16, 64, 256, 1024.
     * Ignore the problem of numeric rollover.
     * @param count the highest power of 4 to be returned
     * @return a List151Impl (or null) as described above.
     */
    public List151Impl<Integer> makeList151(int count);
}
```

```java
public class SignFour implements SignOfFour {

    @Override
    public int number() {
        return 4;
    }

    @Override
    public int add(int num) {
        return 4+num;
    }

    @Override
    public SepChainHT<Integer, Integer> makeHT(int count) {
        if (count <= 0) {
            return new SepChainHT<>();
        } else {
            SepChainHT<Integer, Integer> ret = new SepChainHT<>();
            for (int i = 0; i < count; i++) {
                ret.put(i, i);
            }
            return ret;
        }
    }

    @Override
    public List151Impl<Integer> makeList151(int count) {
        List151Impl<Integer> ret = new List151Impl<>();
        int mul = 1;
        for (int i = 0; i <= count; i++) {
            ret.add(mul);
            mul *= 4;
        }
        return ret;
    }

}
```

Question 2 (20 points) :

Part 1 (19 points): Implement the class and method as documented below

You are given a complete implementation of the StuffBag class which is identical to https://cs.brynmawr.edu/cs151/L03/StuffBag.java.txt except that the instance variable stuffArray is defined to be protected rather than private. You have been asked to create a new class, named StuffBagEx, that extends StuffBag. StuffBagEx has a single new method that is documented below. Write the StuffBagEx class. (If you cannot figure out how to do the reversing, just do the move into a List151Impl (max of 14 points)). Hint, get everything into the List151Impl, then do the reversing thing.)

(List151Impl is as in https://cs.brynmawr.edu/cs151/L05/List151Impl.java.txt).

```java
    /**
     * Return an instance of List151Impl that contains everything
     * in the current bag, twice — in a spacial pattern.
     * Suppose that the
     * StuffBag contains A, B and C. Then the List151Impl with contain
     * A, B and C in some order, followed by those items in exactly
     * the reverse of the order in which they first appear.
     * eg, A,C,B,B,C,A
     * @return an instance of List151Impl containing everything
     * that was in the bag, twice as described above.
     */
    public List151Impl<R> toListI151ImplTwo()
```

```java
public class StuffBagEx<R> extends StuffBag<R> {
    public List151Impl<R> toListI151ImplTwo() {
        List151Impl<R> ret = new List151Impl<>();
        for (int i=0; i<stuffArray.length; i++) {
            if (stuffArray[i] != null) {
                ret.add(stuffArray[i]);
            }
        }
        for (int i = ret.size() - 1; i >= 0; i--) {
            ret.add(ret.get(i));
        }
        return ret;
    }
}
```

Question 2, part 2 (1 point): What is the time complexity of your new method. This is for your method, irregardless of its correctness. Briefly (at most 2 sentences) explain your answer.

O(n). The first loop is actually not dependent on the number of items in the bag, so it is O(1). The second loop touches each item in the bag exactly once, so it is O(n). Two independent loops O(n) and O1) —> O(1)+O(n)==O(n)

Question 3 (20 points): Algorithmic Complexity.

PART 1: (2 points) Place the following complexity descriptors in order based on their expected asymptotic run-time (ie, when n gets really large).

O(n)
O(lg n)
O(n*n)
O(1)
O(n * lg n)

O(1), O(lg n), O(n), O(n * lg n), O(n*n)

PART 2: (consists of 5 parts — 3 points each): For each of the methods of the Complx class on the next page, give its algorithmic complexity.

Fill in the table below with the algorithmic complexity of the methods in the Complx class

Method	**Complexity**
sumA	O(n)
sumB	O(1) typo in question. Answer intended was O(lg n) so that as accepted also
sumC	O(1)
sumD	O(N*N)
sumE	O(lg n)

PART 3: (3 points) Write a method, named sumF, to be put in the Complx class. sumF should have O(n^3) runtime

```
public void sumF() {
        int sm=0
        for (int i=0; i<arrA.length; i++) {
                for (int j=0; j<arrA.length; j++) {
                        for (int k=0; k<arrA.length; k++) {
                                sm = sm + i+j+k;
}}}
System.out.println(sm);
}
```

```
public class Complx {
    int[] arrA;
    public Complx(int[] inp) {
        arrA = inp;
    }
    public int sumA() {
        int summ=0;
        for (int i = 0; i < arrA.length; i = i+1) {
            summ += arrA[i];
        }
        return summ;
    }
    public int sumB() {
        int summ = 0;
        double cc = arrA.length;
        while (true) {
            if (cc >= 1)
                break;
            summ += arrA[(int)cc];
            cc = cc/1.001;
       }
        return summ;
    }
    public int sumC() {
        int summ=0;
        for (int i = 0; i < arrA.length; i = i + 1) {
            if (i > 5) {
                return arrA[i];
            }
        }
        return summ;
    }
    public int sumD() {
        int summ = 0;
        for (int i = 0; i < arrA.length/10; i++) {
            summ += sumA();
        }
        return summ;
    }
    public int sumE() {
        int summ = 0;
        for (int i = 0; i < 40000000; i++) {
            for (int j = 1; j < arrA.length; j = j * 2) {
                if (i <= j) {
                    summ += arrA[i] + arrA[j];
                }
            }
        }
        return summ;
    }
}
```

Question 4: (20 points) Create a class called Vehicle that holds at least 3 pieces of information about cars or trucks (eg, number of wheels, kind of engine). Then create a class that inherits from Vehicle called Car. Car should have at least one new piece of information beyond that in Vehicle. For each of Car and Vehicle, write: a constuctor, toString method and equals method. The constructor should initialize all instance variables to values given in parameters to the construtor. The toString should show the values of all instance variables. The return from equals should be based on some or all of the instance variables in the class. Your Car class should use inheritance appropriately from Vehicle.

Extra Credit: (up to 4 points) For each class, write a main method that illustrates the use of the constructor, toString and equals that you just wrote. The illustration should be just that, an illustration, not an exhaustive test.

```
public class Vehicle {
    private int wheels;
    private String maker;
    private String engineType;

    public Vehicle(int w, String m, String e) {
        this.wheels = w;
        this.maker = m;
        this.engineType = e;
    }

    public String toString() {
        return "Maker:" + maker + " Engine:" + engineType + "
Wheels:" + wheels;
    }

    public boolean equals(Object ob) {
        if (ob instanceof Vehicle) {
            Vehicle v = (Vehicle) ob;
            return this.wheels == v.wheels;
        }
        return false;
    }
}
```

```java
public class Car extends Vehicle {
    String color;

    public Car(int w, String m, String e, String c) {
        super(w, m, e);
        this.color = c;
    }

    public String toString() {
        return super.toString() + "Color: " + color;
    }

    public boolean equals(Object ob) {
        if (ob instanceof Car) {
            Car v = (Car) ob;
            return this.color.equals(v.color);
        }
        return false;
    }
}
```

Question 5 (20 points): Consider a probing hashtable. Keys are strings with length 2. All keys are composed only of the letters a, b, c. All values are a single letter. The key strings are converted into an integers using a variation of Horner's method as follows.

- Letters have the following integer values. a=1, b=3, c=5. (We are **not** using the ASCII value of the letters).
- The key is converted to an integer from beginning to end using 7 as the prime multiplier.

So the calculation of the hash value for the key "ac", given a backing array of size 8, would be:

result = a + c*7
result = 1 + 5*7
result = 1 + 35 = 36

Finally do the standard modulus thing to get into the range for the hashtable.

hashValue = 36 % 8 = 4

Part 1 (12 points): Show the contents of the hash table after these 6 additions when using the hashing function described above, and a hashtable of size 8. Use quadratic probing. Be sure to legibly show your work to be able to receive partial credit.

Add <"ab", a>
Add <"aa", b>
Add <"bb", c>
Add <"cc", d>
Add <"cb", e>
Add <"ab", f>
Add <"ba", g>

	calculation of hash	hash value	mod 8	mod 10
ab	1+3*7	22	6	2
aa	1+7	8	0	8
bb	3+3*7	24	0	4
cc	5+5*7	40	0	0
cb	5+3*7	26	2	6
ab	REPEAT	———		
ba	3+1*7	10	2	0

	0	1	2	3	4	5	6	7	8	9
ht=8	<aa,b>	<bb,c>	<cb,e>	<ba,g>	<cc,d>		<ab,f>		XXXX	XXXX
ht=10	<cc,d>	<ba,g>	<ab,f>		<bb,c>		<cb,e>		<aa,b>	

Question 5 PART 2 (8 points): The hash table has gotten too full. Rehash into a new table of size 10.