Points off	1	2	3	4			Raw Points Off

Your Name: _____

Your UTEID: _____

Circle your TA's Name: **Anna Brad David Emma Justin Lilly Natalee Pavan Pranav Skyler**

Instructions:

1. There are **4** questions on this test. 100 points available. Scores shall be scaled to 250 points.
2. You have 2 hours to complete the test.
3. Place your final answers on this test. Not on the scratch paper. **Answer in pencil**. Exams not completed in pencil are not eligible for a regrade. You may use highlighters on the exam.
4. You may not use a calculator or **outside resources of any kind** while taking the test. Please remove any smart watches and put them and any mobile devices away.
5. When answering coding questions, ensure you follow the restrictions of the question.
6. Do not write code to check the preconditions.
7. On coding questions, you may implement your own helper methods.
8. On coding questions make your solutions as efficient as possible given the restrictions of the question.
9. Test proctors will not address any questions regarding the content of the exam. If you think a question is ambiguous or has an error, state your assumptions and answer based on those assumptions.
10. When you complete the test show the proctor your UTID and give them the test. Please place used and used scratch paper in the appropriate boxes at the front of the room. Please leave the room quietly.

1. (2 points each, 50 points total) Short answer. Place your answer on the line next to or under the question. Assume all necessary imports have been made.

   a. If a question contains a syntax error or compile error, answer **compile error**.
   b. If a question would result in a runtime error or exception, answer **runtime error**.
   c. If a question results in an infinite loop, answer **infinite loop**.
   d. Recall when asked for Big O your answer shall be the most restrictive correct Big O function. For example, Selection Sort is average case $O(N^{2})$, but per the formal definition of Big O it is correct to say Selection Sort is $O(N^{3})$, $O(N^{4})$ and so forth.
      Give the most restrictive, correct Big O function. (Closest without going under.)
   e. Assume $\log_2(1{,}000) = 10$ and $\log_2(1{,}000{,}000) = 20$.

A. Using the techniques and rules from lecture, what is the T(N) of the following method? N = n

_____

```
public static int a(int n) {
    int t = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int temp = i * j;
            t += temp;
        }
    }
    return t;
}
```

B. Using the techniques and rules from lecture, what is the T(N) of the following method? N = data.length _____

```
public static int b(int[] data) {
    int t = 0;
    for (int i = 0; i < data.length; i++) {
        t += i * i;
        if (i == data.length / 2) {
            for (int j = 0; j < data.length; j++) {
                t += i * j;
            }
        }
    }
    return t;
}
```

C. A method is O(N!). The method takes 1 second to complete when N = 10. What is the expected time for the method to complete when N = 12?

_____

D. The following method takes 2 seconds to complete when **`data.length`** = 1,000,000. What is the expected time for the method to complete when **`data.length`** = 2,000,000?

_____

```
private static int d(int[] data) {
    int r = 0;
    for (int i = data.length - 1; i >= 0; i--)
        for (int j = 0; j < i; j++)
            if (data[j] != 0)
                r += data[i] % data[j];
    return r;
}
```

E. The following code takes 1 second to complete when **`list.size()`** = 500,000. What is the expected time for the method to complete when list**`.size()`** = 2,000,000? Typically, 10% of the elements of **`list`** are **`null`**. The non-null elements of **`list`** always have a length of 5 or less.

_____

```
public static int e(ArrayList<String> list) {
    int r = 0;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) != null) {
            for (int j = 0; j < list.get(i).length(); j++)
                if (list.get(i).charAt(j) == 'e')
                    r++;
        }
    }
    return r;
}
```

F. The following method takes 3 seconds to complete when n = 10,000. What is the expected time for the code to complete when n = 20,000? The **`Math.random`** method is O(1)

_____

```
public static ArrayList<Double> f(int n) {
    ArrayList<Double> result = new ArrayList<>(n);
    for (int i = 0; i < n; i++) {
        double a = Math.random();
        if (a < 0.5) {
            result.add(0, a); // insert method (pos, val)
        } else {
            result.add(a);
        }
    }
    return result;
}
```

G. The following method takes 10 seconds to complete when **`data.length`** = 10,000. What is the expected time for the method to complete when **`data.length`** = 20,000?

_____

```
public static String g(int[] data) {
    String st = "";
    for (int i = 0; i < data.length; i++) {
        st += data[i] % 10;
    }
    return st;
}
```

H. What is output by the following code? _____

```
ArrayList<Integer> list = new ArrayList<>();
for (int i = 12; i >0; i -= 2) {
     list.add(i);
}
list.remove(list.size() / 2);
list.remove(list.size() - 4);
System.out.println(list);
```

I. Which of the following is the most likely internal storage container for a **`StringBuilder`**? Answer with a single letter. _____

A. **`char[]`** B. **`String`** C. **`ArrayList<Character>`**

D. **`ArrayList<String>`** E. **`String[]`**

J. What is output by the following method if the **`ArrayList<String> list`** contains the following values in the order shown: **`["Justin", "Anna", "Pavan", "Skyler"]`**

_____

```
public static void j(ArrayList<String> list) {
    int r = 0;
    Iterator<String> it = list.iterator();
    while (it.next() != null) {
        r += it.next().length();
    }
    System.out.print(r);
}
```

K. What is output by the following code? _____

```
ArrayList<Integer> list1 = new ArrayList<>();
list1.add(4);
list1.add(2);
ArrayList<Integer> list2 = new ArrayList<>();
list2.add(3);
list2 = list1;
list1.add(0);
list2.add(7);
System.out.print(list1);
```

L. What is output by the following code? _____

```
Object o1 = new ArrayList<>();
Object o2 = new ArrayList<>();
System.out.print((o1 == o2) + " " + o1.equals(o2));
```

M. What is output by the following code when it is run? Recall the output of the **`toString`** method from the **`Map`** interface: **`{key_1=value_1, key_2=value_2, ..., key_n=value_n}`**

_____

```
int[] data = {5, -2, 0, -2, 3, 0, 0, 5};
TreeMap<Integer, Integer> map = new TreeMap<>();
for (int i = 0; i < data.length; i += 2) {
    map.put(data[i], data[i + 1]); // (key, value)
}
System.out.print(map);
```

N. What is the average case order of the following method? N = **`list.size()`**. _____
The **`Math.random`** method is O(1).

```
public static void n(ArrayList<String> list) {
    Iterator<String> it = list.iterator();
    while (it.hasNext()) {
        it.next();
        if (Math.random() <= 0.3) {
            it.remove();
        }
    }
}
```

O. What is the average case order of the following method? **`N = list.size()`**. Assume half of the elements of list are even. Methods **`check1`** and **`check2`** are both O(N) where N is the size of the list passed to the method.

_____

```
public static int o(ArrayList<Integer> list) {
    int r = 0;
    for (int i = 0; i < list.size(); i++) {
        int x = list.get(i);
        r += check1(list, x);
        if (x % 2 == 0) {
            r += check2(list, i);
        }
    }
    return r;
}
```

For questions 1.P - 1.Y refer to the classes at the end of the exam.
You may detach that page from the exam.

P. What is output by the following code? _____

```
PhysicalBook b1 = new PhysicalBook();
System.out.print(b1.length());
```

Q. What is output by the following code? _____

```
HardCover h1 = new HardCover(25, "pink");
System.out.print(h1.getColor() + " " + h1.getFont());
```

R. What is output by the following code? _____

```
Object ob = new PhysicalBook(100, "teal");
System.out.print(ob.toString());
```

S. What is output by the following code? _____

```
HardCover h2 = new HardCover(50, "black");
PhysicalBook p2 = new PhysicalBook(75, "red");
System.out.print(h2.compareTo(p2));
```

T. What is output by the following code? _____

```
Book b3 = new SoftCover(40);
System.out.print(b3.length());
```

U. What is output by the following code? _____

```
Book b4 = new SoftCover(60);
System.out.print(b4.material());
```

V. What is output by the following code? _____

```
PhysicalBook p5 = new PhysicalBook();
HardCover h5 = new HardCover(50, "fuchsia");
System.out.print(p5.getClass().equals(h5.getClass()));
```

W. What is output by the following code? _____

```
Book b6 = new PhysicalBook(60, "beige");
Book b7 = new HardCover(25, "yellow");
System.out.print(b6.compareTo(b7));
```

X. What is output by the following code? _____

```
Book b8 = new HardCover(15, "blue");
System.out.print(b8.length() + " " + ((HardCover) b8).getFont());
```

Y. What is output by the following code? _____

```
PhysicalBook p9 = new PhysicalBook(100, "gray");
System.out.println(p9.getColor() + " " + ((SoftCover) p9).material());
```

2. **Lists.** (20 points) To demonstrate encapsulation and the syntax for building a class in Java, we developed a **`GenericList`** class that can store elements of any data type. Recall our **`GenericList`** class stores the elements of the list in the first **`size`** elements of a native array of **`Object`**s. An element's position in the list is the same as the element's position in the array. The array may have extra capacity and thus may be larger than the list it represents. **The list does not allow client code to add null elements.**

Complete a constructor for the **`GenericList`** class that builds a new **`GenericList`** from a preexisting **`GenericList`** and an array of i**`nt`**s representing the number of times to repeat a given element from the original **`GenericList`** in the new **`GenericList`**. The array of **`int`**s shall be equal in length to the size of the given **`GenericList`**. (In other words, it is a parallel array to the given **`GenericList`**.)

```
/*   pre: org != null, counts != null, counts.length = org.size(), all
            element of counts >= 0
     post: Per the problem description. org and counts are not altered
            by this constructor. */
public GenericList(GenericList<E> org, int[] counts) {
```

Examples of the **`GenericList`** resulting from calls to the constructor.
The values shown are **`String`** objects.

```
new GenericList([A, C, A], [1, 3, 2]) -> [A, C, C, C, A, A]

new GenericList([A], [5]) -> [A, A, A, A, A]

new GenericList([A, D, E, C], [2, 0, 0, 1]) -> [A, A, C]

new GenericList([], []) -> []

new GenericList([AT, C, CS], [3, 1, 2]) -> [AT, AT, AT, C, CS, CS]
```

The **`GenericList`** class:

```
public class GenericList<E> {
     private E[] con;
     private int size;
```

You may not use any methods from the `GenericList` class unless you implement them as a part of your solution.

You may not use any other Java methods or classes except native arrays.

Complete the constructor on the next page.

```
/*   pre: org != null, counts != null, counts.length = org.size(), all
            element of counts >= 0
     post: Per the problem description. org and counts are not altered
            by this constructor. */
public GenericList(GenericList<E> org, int[] counts) {
```

3. Baby Names (15 points) Complete an instance method in the **`Names`** class that returns an **`ArrayList<String>`** of all the names that are out of sync with a given name based on a minimum allowed difference.

A name is *out of sync* with another if the differ by at least some minimum value for **every** decade. Recall, a value of 0 indicates the name had a rank greater than 1000 for the decade. **For this question assume a 0 indicates a rank of 1001 when calculating the difference between ranks.** Consider the following example:

```
Isabelle        193 192 259 426 619 987   0   0   0 440 144
Becky             0   0   0 661 260 151 141 195 419   0   0
Abs Difference 808 809 742 235 359 836 860 806 582 561 857
```

The absolute value of the difference between the ranks of Isabelle and Becky is always at least 235. Therefore, if the minimum difference allowed is 235 or more Isabelle and Becky are out of sync. If the minimum allowed difference is 234 or less then Isabelle and Becky are not out of sync.

It turns out with a **`minDiff`** of 235 Isabelle is out of synch with 37 names including Becky, Cathy, Darcy, Debra, Greg, Marcy, Rusty, Scotty, and Valarie.

The **`Names`** class for this question:

```
public class Names {
     // the NameRecords in this Names object.
     // All NameRecords have the same number of decades.
     private ArrayList<NameRecord> names;

     // get a NameRecord for the given name. Returns null if not present
     public NameRecord getRecord(String name)
}
```

Methods you may use from **`NameRecord`**: **You may not add methods to the NameRecord class.**

```
String getName() - return the name for this NameRecord
int numDecades() - return the total number of decades, including unranked
int getRank(int decade) - return the rank for the given decade. Uses 0
based indexing. Returns 0 if unranked in the given decade.
```

From the **`ArrayList`** class:

```
ArrayList() - construct an empty ArrayList
add(E obj) - add obj to the end of this ArrayList
int size() - number of elements in this ArrayList
E get(int pos) - access element at given position
```

You may also use **`ArrayList`**s as the target of **`for-each`** loops.

You may use the **`Math.abs`** method.

Do not use any other Java classes or methods besides those listed above.

```
/* pre: name != null, there is a NameRecord with name in this
     Names object, minDiff >= 1
  post: return an ArrayList<String> of names out of sync with name.
     Names is not altered as a result of this method call. */
public ArrayList<String> outOfSync(String name, int minDiff) {
```

4. **Other Data Structures** (15 points) Complete the **`remove(K key)`** instance method for an **`ArrayMap`** class. Recall a map consists of key-value pairs. Each key maps or corresponds to a value. Also recall, keys are unique in a map. A given key can appear at most one time in the map.

```
/* pre: key != null. post: per the problem description. */
public V remove(K key)
```

The method accepts a key. If the key is present in this map, the key-value pair is removed from the map and the value that was associated with that key is returned. If the key is not present in this map then the method returns **`null`**.

This **`ArrayMap`** class uses a 1d array of **`Entry`** objects. Each Entry object stores a key and a value. **`null`** keys are not allowed. Like our list-based example the internal array may have extra capacity. Unlike our list-based array example, the **`Entry`** objects may be scattered throughout the array in no particular order, with **`null`** values in between. The example below represents the 1d array for an ArrayMap object. The Entry objects are shown abstractly as (key, value) in the array.
`/` indicates an array element that currently stores **`null`**. The size of the **`ArrayMap`** in this example is 5 as it currently holds 5 key-value pairs. The keys in this example below are **`String`** objects and the values are **`Integer`** objects.

/	(of, 101)	/	/	(the, 217)	(is, 156)	/	/	(to, 101)	/	/	(are, 202)

The **`ArrayMap`** class for this question:

```
public class ArrayMap<K, V> {

    private Entry<K, V>[] con;
    private int size; // The number of key-value pairs in this Map.

    private static class Entry<K, V> {
        private K key;
        private V value;
    }
}
```

Do not use any other Java methods or classes except the nested `Entry` class and the equals method for objects. Of course, you can use the native arrays **`length`** field.

Do not use any other `ArrayMap` methods unless you implement them as a part of your answer.

Do not create any new data structures.

Complete the method on the next page.

```
/* pre: key != null. post: per the problem description. */
public V remove(K key) {
```

For questions P through Y, refer to the following classes. You may detach this page from the exam.

```
public abstract class Book implements Comparable<Book> {
    private int pages;

    public Book() {}

    public Book(int p) { pages = p; }

    public String toString() { return "len: " + length(); }

    public int length() { return pages; }
}

public class PhysicalBook extends Book {
    public String color;

    public PhysicalBook() { color = "orange"; }

    public PhysicalBook(int pages, String c) {
        super(pages);
        color = c;
    }

    public int compareTo(Book b) { return length() - b.length(); }

    public String getColor() {return color; }
}

public class HardCover extends PhysicalBook {

    public HardCover(int pages, String color) { super(pages * 2, color); }

    public int getFont() { return 16; }
}

public class SoftCover extends PhysicalBook {

    private int saved;

    public SoftCover(int pages) {
        super(pages, "black");
        saved = pages / 2;
    }

    public int length() { return saved; }

    public String toString() { return "small"; }

    public String material() { return "paper"; }
}
```