Points off	1	2	3	4				Raw Points Off

Your Name: ____

Your UTEID: ____

Circle your TA's Name:

Aditya	**Ahmad**	**Aman**	**Anna**	**Bersam**	**Brad**
Brayden	**Casey**	**Eliza**	**Gracelynn**	**Lauren**	
Namish	**Nidhi**	**Pavan**	**Sai**		

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 shall **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 `log₂(1,000) = 10` and `log₂(1,000,000) = 20.`

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

```
public static int a(int[] data) {                _______________________________
    int t = 0;
    for (int i = 0; i < data.length; i++) {
        if (i % 2 == 0) {
            int x = data[i] * 5;
            data[i] = i + 3;
            t += x;
        }
    }
    return t;
}
```

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

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

C. A method is O(2^N). The method takes 10 hours to complete when N = 60. What is the expected time for the method to complete when N = 63?

____

D. A method is O(N^2). The method takes 1 second to complete when N = 100,000. What is the expected time for the method to complete when N = 300,000?

____

E. The following code takes 0.1 second to complete when **data.length** = 1,000,000. What is the expected time to complete when **data.length** = 2,000,000?

____

```
public static int e(int[] data) {
    int r = 0;
    for (int i = 1; i < data.length; i *= 2) {
        for (int j = 0; j < data.length; j++) {
                r += data[j] * data[i];
        }
    }
    return r;
}
```

F. What is output by the following code? The code uses the **java.util.ArrayList** class.

____

```
ArrayList<Integer> listF = new ArrayList<>(5);
listF.add(5);
listF.add(2);
listF.add(5);
listF.add(1, listF.size()); // position, value
listF.add(listF.get(2));
System.out.print(listF);
```

G. What is output by the following code? The code uses the **`java.util.ArrayList`** class.

____

```
ArrayList<String> listG = new ArrayList<>(5);
listG.add("U");
listG.add("S");
listG.add("J");
listG.add("V");
listG.add(1, listG.remove(listG.size() - 1));
listG.add(listG.remove(2));
System.out.print(listG);
```

H. The following method takes 5 seconds to complete when **`data.length`** = 100,000. What is the expected time for the method to complete when **`data.length`** = 200,000?
The method uses the **`java.util.ArrayList`** class.

____

```
public static ArrayList<Integer> h(int[] data) {
    ArrayList<Integer> r = new ArrayList<>();
    for (int x : data) {
        r.add(x);
    }
    for (int i = 0; i < data.length / 2; i++) {
        r.add(0, data[i]);              // position, value
        r.add(r.size() / 2, data[i]); // position, value
    }
    return r;
}
```

I. What is the order (Big O) of the following code? N = **`list.size()`** The method uses the **`java.util.ArrayList`** class.

____

```
// pre: list.size() > 10
public static void methodI(ArrayList<String> list) {
    final int LIMIT = list.size() / 3;
    while (list.size() > LIMIT) {
        list.remove(list.size() - 3);
    }
}
```

J. The following method takes 5 seconds to complete when **n** = 25,000. What is the expected time for the method to complete when **n** = 50,000? The code uses the **GenericList** method developed in lecture and the **resize** method for the **GenericList** class is as shown.

____

```
public static GenericList<Integer> j(int n) {
    GenericList<Integer> r = new GenericList<>();
    for (int i = 0; i < n; i++) {
        r.add(i * i);
    }
    return r;
}

// resize method in the GenericList class
private void resize() {
    con = Arrays.copyOf(con, con.length + 5);
}
```

K. What is output by the following code? _______________________________

```
Comparable c1 = "Bersam";
Comparable c2 = new Comparable();
Comparable c3 = null;
System.out.print(c1.compareTo(c2) + " " + c3.compareTo(c1));
```

L. Why doesn't the following class compile?

____

```
public class Book extends Object implements Comparable<Book> {

    private String title;
    private String genre;

    public Book(String t) {
        title = t;
        genre = "fiction";
    }

    public String toString() { return "" + title.compareTo(genre); }
}
```

M. What is output by the following code when it is run?

```
Integer m1 = Integer.valueOf(37); // Creates a new Integer object
Integer m2 = Integer.valueOf(1000); // with the given value.
if (m1.compareTo(m2) > 0) {
    System.out.print("+");
} else {
    System.out.print("-");
}
if (m1.equals(m2)) {
    System.out.print("+");
} else {
    System.out.print("-");
}
```

N. What is output to the screen when method **n** is called with a **java.util.ArrayList** that contains the following values?

```
["Sai", "Brad", "Anna", "Eliza", "Aman", "Pavan", "Lauren", "Brayden"]

    public static void n(ArrayList<String> list) {
        int c = 0;
        Iterator<String> it = list.iterator();
        while (it.hasNext()) {
            if (it.next().length() <= 4) {
                c++;
            }
        }
        System.out.print(c);
    }
```

O. Which line of code below causes a compile error? ______________________________

```
    ArrayList<String> nameList = new ArrayList<>(); // 1
    nameList.add(nameList.toString()); // 2
    nameList.add(new StringBuilder()); // 3
    nameList.add("Pavan".substring(2, 3)); // 4
    nameList.add("Aditya"); // 5
    nameList.add("Casey".toLowerCase()); // 6
    nameList.add("Gracelynn".substring(30)); // 7
```

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. For each of the following circle if the code compiles or causes a compile error. (1 point each)

```
Object ob1 = new Eagle(); //    compiles             compile error

Bird b1 = new Emu();      //    compiles             compile error
```

Q. For each of the following circle if the code compiles or causes a compile error. (1 point each)

```
Eagle e1 = new Bird(5);   //    compiles             compile error

Animal a1 = new Eagle(5); //    compiles             compile error
```

R. What is output by the following code? ______________________________

```
Eagle ea2 = new Eagle(2);
ea2.bday();
ea2.bday();
System.out.print(ea2.getAge() + " " + ea2.move());
```

S. What is output by the following code? ______________________________

```
Animal a3 = new Emu();
System.out.print(a3.getAge() + " " + a3.move());
```

T. What is output by the following code? ______________________________

```
Eagle ea4 = new Eagle(5);
Bird b4 = ea4;
b4.bday();
ea4.bday();
b4.bday();
System.out.print(ea4.getAge());
```

U. What is output by the following code? ______________________________

```
Animal a5 = new Emu();
System.out.print(a5);
```

V. What is output by the following code? _____________________________________

```
Animal a6 = new Eagle(10);
System.out.print(a6);
```

W. What is output by the following code? _____________________________________

```
Snake s7 = new Snake();
Emu e7 = new Emu();
System.out.print(s7.equals(e7) + " " + s7.move());
```

X. What is output by the following code? _____________________________________

```
Animal a8 = new Snake();
int b8 = a8.bodyTemp(10);
System.out.print(b8 + " " + a8.getAge());
```

Y. What is output by the following code? _____________________________________

```
Emu e9 = new Emu();
e9.bday();
e9.bday();
System.out.print(e9.getAge() + " " + e9.bodyTemp());
```

2. **Lists.** (18 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. This version of 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 ALLOWS client code to add `null` elements.

Create an instance method for the **`GenericList`** class **`getMatchingElements`**. The method creates and returns a new **`GenericList`** that contains all the matching elements at the same indices from the calling list and another list sent as a parameter.

For example, given these two lists that contains **`String`**s and **`null`** values.

```
list1 [A, B, C, A, null, A,    null, A, BB, null]
list2 [A, A, C, D, null, null, BBB]
```

The method would return a new list with elements **`[A, C, null].`** (**`A`** and **`C`** are **`Strings`.)**

The elements in the resulting list are in the same relative order as the elements from the original lists.

If the lists are different sizes only valid elements are considered.

For example, given these two lists

```
list1 [D, B, C, null, Z,    M]
list2 [A, B, Z, M,    null, X, BB, AA, CAT]
```

the method shall return the following list: **`[B]`** **`(B`** is a **`String)`**

Final example. Given these two lists:

```
list1 [D, B, C, A, Z, M]
list2 [A, C, Z, M]
```

the method shall return the following list: **`[]`** **`(an empty GenericList)`**

The **`GenericList`** class:

```
public class GenericList {

    private Object[] con;
    private int size;

    public GenericList(int initialCapacity) {
        con = new Object[initialCapacity];
    }
```

You may only use the constructor from the `GenericList` class shown above. You may not use any other methods from the `GenericList` class unless you implement them yourself as a part of your solution.

You may call the `equals` method on objects. Of course, you may use the `length` field for arrays.

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

```
// pre: other != null
// post: Neither this or other are altered and per the problem description.
public GenericList getMatchingElements(GenericList other) {
```

3. Baby Names (18 points) Complete an instance method in the **`Names`** class that returns an **`ArrayList<String>`** of all the names that are "rare-ish" given a max (really min) rank and a required number of decades to be ranked.

A rare-ish name is one that never is **more** popular than some cutoff value and appears in some minimum number of decades. For example, given a cutoff rank of 700 and a minimum of 8 decades being ranked in the top 1000, the following names are rare-ish.

```
Eliseo,  ranks:   0   0 900 946 775 803 928 896 840 831 960
Isidro,  ranks: 956   0 840 921 778 799 989 916 880 896 948
Luciano, ranks: 766 823 817 700 862 924   0 954   0 982 873
```

Luciano's rank in the fourth decade is 700, right at the cutoff. Had the rank been 699 Luciano would not have met the criteria. Likewise, Luciano is ranked in the top 1000 for 9 of the 11 decades. Had the last two ranks been 0 and 0 instead of 982 and 873 Luciano would not have met the criteria.

If no names meet the criteria return an empty **`ArrayList<String>`**.

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;
}
```

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.

Do not use any other Java classes or methods besides those listed above.
Create a single `ArrayList<String>`, the result to return.
Do not create any other new data structures.

Complete the method on the next page.

```
/* pre: 1 <= cutoff <= 1000, requiredRanks <= the number of decades
       NameRecords in this Names object are ranked.
  post: this Names object is not altered and per the problem
        description. */
public ArrayList<String> getRareish(int cutoff, int requiredRanks) {
```

4. **Other Data Structures** (14 points) Complete the **`void remove(int position)`** instance method for a **`RunLengthList`** class. A **`RunLengthList`** does not explicitly store all the elements in the list. Rather it stores the number of occurrences of an element in the current run and the element itself. This implementation can save space if elements typically occur in runs or are repeated.
For example, consider the following list with a size of 23 from the client's perspective:

```
[5, 5, 5, 5, 7, 7, 7, 7, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3]
```

Instead for storing each element explicitly a **`RunLengthList`** stores an **`ElementRun`** object with two instance variables, the element itself and the number of times it appears in a row. Given this implementation the above list is represented as follows (Notice how this requires less space?):

```
[(5, 4), (7, 4), (3, 8), (2, 1), (3, 6)]   (element, runLength)
size = 23, numRuns = 5
```

If we were to call **`remove(0)`** on the above list it would become:

```
[(5, 3), (7, 4), (3, 8), (2, 1), (3, 6)] (run of 5 at the start now 3)
```

If we were to call **`remove(15)`** on the resulting list shown just above it would become:

```
[(5, 3), (7, 4), (3, 8), (3, 6)]   (element, runLength)
```

Note, there is no longer a run of 2, so the corresponding **`ElementRun`** is removed. **For this question, if removing an element results in two initially separated runs of the same element to become adjacent do NOT attempt to combine them into a single `ElementRun` object. In the above example with the adjacent runs of 3's we *could* combine into a single `ElementRun` of `(3, 14)` but do NOT attempt this in your solution. (No joke, we will take off points if you try.)**

```
public class RunLengthList<E> { // Does NOT allow client to add null.

    private ElementRun<E>[] con; /* May have extra capacity.
             ElementRun objects stored in first numRuns spots.*/

    private int size; /* The number of elements in the list from the
                         client's perspective. */

    private int numRuns; // The number of ElementRun objects present.

    private static class ElementRun<E> {
        private E element;     // Never null.
        private int runLength; // Always >= 1
    }
}
```

Do not use any other Java methods or classes except the `RunLengthList` instance variables and the nested `ElementRun` class. Of course, you can use the native arrays **`length`** field.

Do not use any other `RunLengthList` methods unless you implement them as a part of your answer. Do not create any new data structures.

```java
/* pre: 0 <= pos < size()
   post: per the problem description. */
public void remove(int pos) {
```

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

```
public class Animal {
    private int age;

    public Animal(int a) { age = a; }

    public void bday() { age++; }

    public int getAge() { return age; }

    public int bodyTemp() { return 100; }

    public String move() { return "oof"; }
}


public class Bird extends Animal {
    public Bird(int age) { super(age); }

    public String move() { return "fly"; }
}


public class Eagle extends Bird {
    public Eagle(int age) { super(age); }

    public int bodyTemp() { return 106; }

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


public class Emu extends Bird {
    public Emu() { super(3); }

    public String move() { return "run"; }
}


public class Snake extends Animal {
    public Snake() { super(5); }

    public int bodyTemp() { return 80; }

    public int bodyTemp(int x) { return bodyTemp() + x; }

    public String move() { return "slith"; }
}
```