Points off	1	2	3	4	5		Total off	Net Score

Your Name: _____

Your UTEID: _____

Circle your TAs Name: **Amir Andrew Ben Claire Ethan Hailey Jacob Nina Sam Suhas Smruti Terrell Tony**

Instructions:

1. There are **5** questions on this test. 100 points available. Scores will be scaled to 200 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**.
4. You may not use a calculator or any other electronic devices while taking the test.
5. When answering coding questions, ensure you follow the restrictions of the question.
6. Do not write code to check the preconditions unless the question requires it. (Question 3)
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 answer 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, give them the test and all the scratch paper, used or not, and leave the room quietly.

1. (1 point each, 20 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 from lecture, what is the T(N) of the following method?
N = `a.length`

_____

```
// pre: a.length == b.length
public static int tn(int[] a, int[] b) {
     int r = 0;
     for (int i = 0; i < a.length; i++) {
          int temp = a[i];
          for (int j = 0; j < a.length; j++) {
               int bt =  b[i];
               r += temp * bt;
          }
     }
     return r;
}
```

B. What is the best case order of the following method? N = `data.length`

_____

```
public static int[] methodB(int[] data, int tgt) {
     int t1 = 0;
     for (int i = 1; i < data.length; i *= 2) {
          if (data[i] <= tgt) {
               t1 += data[i];
          }
     }
     int t2 = 0;
     for (int i = 0; i < data.length; i++) {
          if (data[i] >= tgt) {
               t2++;
          }
     }
     return new int[] {t1, t2};
}
```

C. What is the worst case order (Big O) of the following method? N = `data.length`. _____

```
// pre: data is a square matrix, tgt.length = data.length
public static int[][] methodC(int[][] data, int[] tgt) {
     int dim = data.length;
     int[][] result = new int[dim][];
     for (int row = 0; row < dim; row++) {
          int a = data[row][row];
          int i = tgt[row];
          if (a == i) {
               result[row] = new int[dim];
               if (0 <= i && i < dim)
                    result[row][i] = a;
               else
                    result[row][row] = a;
          }
     }
     return result;
}
```

D. What is the worst case order (Big O) of the following method? N = `data.length` _____

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

E. What is the average case order of the following method? Assume roughly half the elements of `data` have an even length. N = `data.length` _____

```
// pre: no elements of data == null
public static ArrayList<String> methodE(String[] data) {
     ArrayList<String> result = new ArrayList<>(data.length);
     for (int i = 0; i < data.length; i++) {
          String s = data[i];
          if (s.length() % 2 == 0) {
               result.add(0, s);
          }
     }
     return result;
}
```

F. A method is $O(N\log_2 N)$. It takes 10 seconds for the method to complete when N = 1,000,000. _____
What is the expected time in seconds for the method to complete when N = 4,000,000?

G. A method is $O(N^2)$. It takes 1 second for the method to complete when N = 10,000. _____
What is the expected time for the method to complete when N = 40,000?

H. A method is $O(2^N)$. It takes 1 second for the method to complete when N = 30. _____
What is the expected time in seconds for the method to complete when N = 40?

I. What is output by the following code?

```
ArrayList<String> listI = new ArrayList<>();
listI.add("P");
listI.add(0, "Y");
listI.add(1, "T");
listI.add("H");
listI.set(3, "O");
listI.add(1, "N");
System.out.print(listI);
```

_____

J. What is output by the following code? The list in the question has already been declared and set to store the following elements: `[5, 4, -2, 7, -6, 8]` _____

```
Iterator<Integer> it = list1.iterator();
while (it.hasNext()) {
     if (it.next() <= 0) {
          System.out.print(it.next() + " ");
     }
}
```

K. What is output by the following code? _____
Recall the `toString` of Maps:
`{key1=value1, key2=value2), ... keyN=valueN}`

```
TreeMap<String, Integer> mk = new TreeMap<>();
mk.put("Z", 5);
mk.put("A", 8);
mk.put("M", 7);
mk.put("A", mk.get("Z") - 10);
System.out.print(mk);
```

L. What is output by the following code? _____

```
ArrayList<String> list2 = new ArrayList<>(20);
Iterator<String> it2 = new Iterator(list2);
System.out.print(it2.hasNext());
```

M. The following method takes .0001 seconds to complete when data.length = 1000. _____
What is the expected time for the method to complete when data.length = 1,000,000?

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

For questions N through T, refer to the classes on the last page. You may detach that page if you wish.

N. For each line of code write **valid** if the line will compile without error or **invalid** if it causes a compile error. (Must get both correct for full credit.)

```
Apt a2 = new Split();          _____

House h2 = new Object();       _____
```

O. What is output by the following code? _____

```
House h1 = new House(10);
h1.inc();
System.out.print(h1);
```

P. What is output by the following code? _____

```
Split s3 = new Split(50);
System.out.print(s3);
```

Q. What is output by the following code? _____

```
Living g5 = new Apt();
g5.add(10);
System.out.println(g5.toString());
```

R. What is output by the following code? _____

```
Split s5 = new Split();
s5.add();
s5.add(10);
System.out.print(s5.get());
```

S. What is output by the following code? _____

```
Apt a8 = new Apt();
a8.fs += 10;
a8.add(10, 20);
System.out.print(a8);
```

T. What is output by the following code? _____

```
House h9 = new House(20);
h9.inc();
h9.add(h9.space());
String s9 = h9.toString();
System.out.print(s9.equals(h9));
```

2. The `GenericList` class (22 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 N elements of a native array. 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.

Complete an instance method for the `GenericList` class named `scale`.

The method creates and returns a new `GenericList` with `factor` (an `int` parameter) shallow copies of the elements of the calling list **except** elements that equal some given target value.

The relative order of the elements in the resulting list is the same as the original list.

```
/*   pre: factor >= 1, tgt != null
     post: Per the problem description. This list is not altered. */
public GenericList<E> scale (int factor, E tgt) {
```

Examples of calls to the `scale` method. (The values shown are `String` objects).

```
[].scale(3, A) -> returns []

[A, A].scale(2, A) -> returns []

[AA, B, B, C, B].scale(3, B) -> returns [AA, AA, AA, C, C, C]

[A, BB, C, A].scale(2, X) -> returns [A, A, BB, BB, C, C, A, A]
```

The `GenericList` class:

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

     public GenericList() {
          con = (E[]) new Object[10];
     }
```

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

Do not use any other Java classes or methods except the `equals` method and native arrays. You may not use static methods from the `Arrays` class.

The list does not allow the client to add null elements.

Complete the method on the next page.

```
/*   pre: factor >= 1, tgt != null
     post: Per the problem description. This list is not altered. */
public GenericList<E> scale (int factor, E tgt) {
```

3. MathMatrix (14 Points) Write an instance method for the `MathMatix` class that returns `true` if the calling object ***diagonally dominates*** a given `MathMatrix` sent as a parameter.

For this question a matrix diagonally dominates another matrix if all elements on and below the main diagonal (that runs from the top left cell to the bottom right cell) are greater than the corresponding elements (at the same location) in the other matrix.

Examples. Does the left hand matrix diagonally dominate the right hand matrix? The elements in question are shown in bold. Elements above the main diagonal do not matter in this question.

```
1 5 7          1 9 6     No, due to cells (0,0) 1 = 1, and
5 3 9          4 2 6     (2, 2) 4 > 3
9 8 3          7 4 4

1 5 7         -1 9 9     Yes, all values in left hand side on and
5 3 3          4 2 6     below the main diagonal are greater than the
9 8 3          7 4 0     corresponding values (same position) from the
                         right hand matrix
```

The `MathMatix` class for this question.

```
public class MathMatrix {

     /* Hold the values for this MathMatrix object.
        No extra capacity.
        Always rectangular. (same number of columns in each row.) */
     private int[][] cells;

     /* pre: this and other are square matrices (number of rows equals
             equals number of columns. this and other have the same
             dimensions.)
        post: return true if this MathMatrix diagonally dominates
              other, false otherwise. Neither MathMatrix is altered.*/
     public boolean diagonallyDominates(MathMatrix other) {
```

Do not use any other Java classes or methods.

On this question only, you must implement code to check the precondition of the method. Throw an IllegalArgumentException if any part of the precondition is not met.

Complete the method on the next page.

```
/* pre: this and other are square matrices (number of rows equals
    equals number of columns. this and other have the same dimensions.
  post: return true if this MathMatrix diagonally dominates other,
    false otherwise. Neither MathMatrix is altered.*/

public boolean diagonallyDominates(MathMatrix other) {
          // recall, you must write code to check the preconditions
```

4. Baby Names (22 points) Write an instance method for the `Names` class that removes all `NameRecord`s from the `Names` object that are unranked in a given number of decades or more. The method returns the number of `NameRecords` removed by the method

You may use the `get(int pos)`, `remove(int pos)`, and `size()` methods from the `ArrayList` class.

You may use the given methods from the `NameRecord` class.

Do not use any other Java classes or methods. Do not create any new `ArrayList` objects.

The `Names` class for this question:

```
public class Names {
    private ArrayList<NameRecord> records;

    // All NameRecords in this name object have NUM_DECADES ranks.
    private final int NUM_DECADES;

    /* pre: numUnranked >= 1
       post: remove all NameRecords that are unranked numUnranked or
          more times. The method returns the number of NameRecords
          removed by this method. */
    public int remove(int numUnranked) {
```

You may use the following method from `NameRecord`: **You may not add methods to the NameRecord class.**

```
int getRank(int decade) returns the rank for the given decade. Uses 0
based indexing. 0 -> first decade, 1 -> second decade, ...
NUM_DECADES - 1 -> last decade. Returns 0 if unranked in the given
decade.
```

Examples. If `numUnranked` was 4 the following names would be removed from the `Names` object.

```
Abe 248 328 532 764 733 0 0 0 0 0 0 (6 unranked decades)
Agnes 37 46 68 134 229 372 612 0 0 0 0 (4 unranked decades)
Aisha 0 0 0 0 0 0 0 486 518 561 682 (7 unranked decades)
Asa 493 554 682 900 0 0 0 0 839 962 659 (4 unranked decades)
Audie 751 861 0 0 0 667 965 0 0 0 0 (7 unranked decades)
```

If `numUnranked` was 4 the following names would be not removed from the `Names` object.

```
Bernadette 416 408 356 386 240 271 287 426 555 891 0 (1 unranked decade)
Bob 443 345 138 89 122 225 199 583 947 0 0 (2 unranked decades)
Bonita 947 784 602 392 218 244 379 813 0 0 0 (3 unranked decades)
```

```
/* pre: numUnranked >= 1
   post: remove all NameRecords that are unranked numUnranked or more
     times. The method returns the number of NameRecords removed by
     this method. */
public int remove(int numUnranked) {
```

5. Other Data Structures (22 points) The list we implemented in class stored every element explicitly. However, what if most of the elements of the list equal the same value? Consider the following list:

```
 0  1  2  3  4  5  6  7  8  9   10 11  12 13 14 15 16  // position
[A, B, A, A, A, A, A, A, A, A, AAA, A,  B, A, A, C, A] // element
```

Storing all those A's seems like a waste of space. In a ***sparse list***, only the elements not equal to the default value are explicitly stored. The default value is set when the list is created and does not change for a given list. Internally we use a native array as our storage container so we also store the position of each element, because the position in the container does not necessarily equal the position in the list.

Consider the following internal representation of the list shown above. Each element in the array is a `ListElem` object that stores one element of data and the position of that element in the list. The elements not equal to the default element, are stored in the array in ascending order based on their position in the list.

`(10, AAA)` in the array below indicates **`AAA`** is at position **`10`** in the list.

```
     0          1         2         3     <- index in array
[(1, B), (10, AAA), (12, B), (15, C), null, null, null, null, null]

default element = A
size of list = 17
elements stored = 4
All elements not stored explicitly equal A for this list.
```

Complete the `void removeFirstN(int n)` method for a `SparseList` class. The method removes the first `n` elements from the list and updates instance variables appropriately.

For example if we called `removeFirstN(12)` on the example list, the list would become, from the client's perspective:

```
 0  1  2  3  4  // position
[B, A, A, C, A] // element
```

Here is the `ListElem` class:

```
public class ListElem<E> {

     public ListElem(int position, E data) // create element

     public E getData() // return data of this element
     public int getPosition() // return position of this element

     public void setPositon(int pos) // set position of this element
     public void setData(E data) // set data of this element
```

The properties of the `SparseList` class are:

- the internal storage container is a native array of `ListElem` objects
- there may be extra capacity in the native array
- only elements not equal to the default element are stored explicitly
- the non-default elements are stored in the first `elementsStored` elements of the array. They are store in ascending order based on their position in the list
- the size of the list, the number of elements stored explicitly in the array, and the default value are stored in separate instance variables
- any elements in the array that are not referring to active elements of the list are set to `null`
- the list does not store elements equal to `null`

Here is the `SparseList` class:

```
public class SparseList<E> {

     private ListElem<E>[] values;
     private int sizeOfList;

     // All values not stored explicitly in values equal defaultValue.
     // defaultValue never equals null.
     private final E defaultValue;

     // Number of elements stored explicitly in values.
     // The elements are stored at the beginning of values.
     private int elementsStored;
```

Do not use any Java methods or classes other than the provided `ListElem` class.

Do not create any new arrays in your solution. O(1) space.

Do not assume any other methods in the `SparseList` class exist.

Complete the `removeFirstN(int n)` method for the `SparseList` class on the next page.

```
/* pre: 0 < n <= size of this list
   post: removes the first n elements from this list. */
public void removeFirstN(int n) {
```

For questions N through T, refer to the following classes. You may detach this page if you wish.

```
public class Living {
      private int sp;

      public Living() { sp = 5; }

      public Living(int s) { sp = s;}

      public int get() { return sp; }

      public void add(int s) { sp += s; }

      public void inc() {  sp += 10; }

      public String toString() { return "L" + get(); }
}

public class House extends Living {
      public House (int x) { add(x); }

      public void add() { inc(); }

      public int space() { return 100; }
}

public class Split extends House {
      private int num;

      public Split() { super(20); }

      public Split(int n) {
            this();
            num = n;
      }

      public void add(int n) { num += n; }

      public int get() { return num; }
}

public class Apt extends Living {
      private int fs;

      public String toString() { return super.toString() + " " + fs; }

      public void set(int f) { fs = f; }

      public void add(int f, int s) {
            fs += f;
            add(s);
      }
}
```