1. (1 point) Community Standard acknowledgment (signature): ____________________________

2. (7 points) Short answers
   a. (1 pt) What is the type name value (TNV) in the following statement:
      ```java
      char letter = 'x';
      ```
      Type: char Name: letter Value: 'x' //or x
   b. (1 pt) When you implement a Comparator Interface, you promise to provide compare method.
   c. (1 pt) You would use a HashMap over TreeMap when you want ______ speed ____________.  
   d. (1 pt) You would use a TreeSet over HashSet when you want to maintain the elements in sorted order____
   e. (1 pt) A hashCode methods that returns a constant leads to many _____ collisions_______
   f. (2 pts) When you declare HashMap<K,V>, the hash value is computing using the call back method
      called HashCode() in the class named _______ K__.

3. (8 points) Big Oh time
   Give the running time of the following code segments as big-Oh in terms of n (tight bound).
   a. (1 pts) Code segment 0
      ```java
      for (int i = 0; i < n; i=i+2) {
          m++;
      }
      ```
      Running time = O(N)
   b. (2 pts) Code segment 1
      ```java
      for (int i = n; i > 1; i=i/2) {
          if (i%2 == 0) m++;
      }
      ```
      Running time = O(log n)
   c. (2 pts) Code segment 2
      ```java
      for (int i = 1; i < n; i++) {
          if (i/2 > 1) m++;
      }
      ```
      Running time = O(n)
      ```java
      for (int i = 1; i < n; i=i*2) {
          if (i*2 < N) m++;
      }
      ```
      Running time = O(n)
d. (3 pts) Code segment 3
for (int i = 1; i < n; i++) {
  for (int j = 0; j < n; j=j*2) {
    if (i + j < n) m++;
  }
}

Running time = Infinite

4. (8 points) Be There and Be Square

The picture below shows the first five rows printed by a simple program starting from 0th row.

```
0
1 2 3
4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24
```

The pattern continues for n rows so that each row has two more numbers than the previous row.

Answer the following questions. You may give an expression instead of an exact number for the first two parts (e.g., for example 2^7, 7^2, \( \log(7) \), etc.) Show your work for this question below the answers to get full credit.

a. (2 points) What is the exact value of the left-most number of the 7th row? 
   \( 7^2 = 49 \)

b. (2 points) What is the exact value of the right-most number of the 10th row?
   \( 11^2 - 1 = 120 \)

c. (2 points) What is the big-Oh of the number of values in the nth row?
   \( O(n) \) based on \( O((n+1)^2 - n^2) \) or \( (2n+1) \)

d. (2 points) What is the big-Oh of the rows if the whole structure has n squares?
   \( O(n^{1/2}) \)

5. (6 points) Extend and Implement

Consider the following Interface IPolygon and Classes Shape and Square (in three different files).

Complete the file Square with:

a. (3 pts) A constructor that initializes all instance variables that are accessible to Square

b. (3 pts) Any methods that Square must implement based on the interface IPolygon.

// Interface for a regular polygon (all sides are equal)
public interface IPolygon {
    public double getArea(); // Returns the area of a regular polygon
    public double getPerimeter(); // Returns the perimeter of the regular polygon
}

// Generic shape... implements perimeter because it is the same for all shapes
public class Shape {
    protected double size, sides;
    protected Color outline;
    public getPerimeter() {
        return size*sides;
    }
}

// Square is a regular polygon with 4 sides
public class Square extends Shape implements IPolygon {

    // The square has 4 sides, so we don't need sides parameter.
    // but other 2 parameters are essential
    public Square(double size, Color c) {
        this.size = size;
        outline = c;
    }

    @Override
    public double getArea() {
        return size * size;
    }

    // Square extends Shape, which provides the implementation of getPerimeter
    // So, we only need to implement getArea (getPerimeter is inherited from Shape)
}

6. (10 points) Common Words

You are given two arrays for Strings and need to return an int to the number of unique Strings that are common in two arrays. For example:
- If the two input String arrays are{"I", "am", "your", "father"} and{"I", "am", "your", "mother"}, then the method returns 3."I", "am", and "your" are the 3 unique common words
- If the two input String arrays are{"I", "am", "what", "I", "am"} and{"I", "am", "I", "am"}, then the method returns 2."I" and "am" are the 2 unique common words

public class CommonWords {
    // Return the number of words common between Strings a and b
    public int count(String[] a, String[] b) {
        Set<String> setA = new HashSet<String>(Arrays.asList(a));
        Set<String> setB = new HashSet<String>(Arrays.asList(b));
        setA.retainAll(setB);
        return setA.size();
    }
}

7. (10 points) TheBestName

Write a method "public void sorter(String[] names)" that takes as input an array of names (Strings) and reorders that array so that it is in sorted order based on the following rules:
A. If two names start with different letters then the one that comes lexicographically (alphabetically) first comes first in the sorted version of names. (Amy comes before Yaqub).
B. If two names start with the same letter then the shorter name comes first but if they are the same length then the one that comes lexicographically (alphabetically) first comes first in the sorted version of names. (Yaquub comes before Yacoub and Bobbi comes before Bobby.)
- If the input String arrays is
  then the method reorders it into
public class TheBestName {
  // Use .sort to reorder the names based on the criteria above
  public void sorter(String[] names) {
    Arrays.sort(names, new SortHelper());
  }
}

public class SortHelper implements Comparator<String> {
  @Override
  public int compare(String str1, String str2) {
    // TODO Auto-generated method
    if (str1.charAt(0) == str2.charAt(0)) {
      if (str1.length() != str2.length()) {
        return str1.length() - str2.length();
      } else {
        return str1.compareTo(str2);
      }
    }
    return str1.compareTo(str2);
  }
}

8. (25 points) Grades: Complete the class Grades by implementing the following in the space provided...
   a. (5 pts) Complete the constructor by filling in the five blanks with the correct “new” statements and the i\(^{th}\) element of scores is assigned the value of data[i].
   b. (5 pts) Write a method public boolean equals(Object other) so that two instances are equal if each element at each position in scores is equal to the corresponding element at that position. (Note: that the other two instance variables are completely dependent on scores). (For example, {1, 2, 3} and {1, 2, 3} are equal... {1, 2, 3} and {3, 2, 1} are not equal... {1, 2, 3} and {1, 2, 3, 4} are not equal...).
   c. (5 pts) Write a method public int hashCode() that computes a hash value that is dependent on all the values stored in scores. Any implementation of hashCode is acceptable as long as you use all the values of scores.
   d. (5 points) Write a method public void setUnique() that sets the values for unique HashSet so that the each score occurs only once in unique. For example:
      If scores is {80, 90, 100} then unique is <80,90,100>
      If scores are {80, 90, 100, 90, 100, 100} then unique is <80,90,100>
   e. (5 points) Write a method public void setCounts() that sets the values for counts HashMap so that the key is the score and the value is the frequency of that score.
      The median is the number separating the higher half of scores from the lower half. For example:
      If scores is {80, 90, 100} then counts is <80,1> <90,1> <100,1>
      If scores is {80, 90, 100, 90, 100, 100} then counts is <80,1> <90,2> <100,3>
public class Grades {
    private ArrayList<Integer> scores;
    private HashSet<Integer> unique;
    private HashMap<Integer, Integer> counts;

    public Grades(int[] data) {
        scores = new ArrayList<>(); //or new ArrayList<Integer>();
        unique = new HashSet<>(); //or new HashSet<Integer>();
        counts = new HashMap<>(); //or new HashMap<Integer,Integer>();
        for (int i = 0; i < scores.length; i++) {
            scores.add(data[i]); //or scores.set(i, data[i]);
        }
        setUnique();
        setCounts();
    }

    // Returns true if this .equals other
    public boolean equals(Object other) {
        if(other == null) return false;
        //check same address, same class
        if(this == other && this.getClass() == other.getClass() return true;
        //check contents
        if(this.scores.length != other.scores.length) return false;
        for(int i=0; i<this.scores.length; i++){
            if(this.scores[i] != other.scores[i]) return false;
        }
    }

    // Returns a hashCode that considers ALL values in scores
    // Note: that the other two instance variables are completely dependent on scores
    public int hashCode() {
        // Lazy implementation:
        return scores.hashCode();
        // This is very lazy, but satisfies all requirements.
        // So I cannot take away any points on this.

        // Standard implementation:
        int sum = 0;
        for (int score: scores) {
            sum += score;
        }
        return sum;

        // As long as all scores are considered, you will get all points.
        // And make sure you are treating scores correctly as an ArrayList, not array
        // (you should use scores.get(i) instead of scores[i])
    }

    // Stores the contents of scores into HashSet unique without any duplicates
    public void setUnique() {
        // Lazy implementation:
        unique = new HashSet<>(scores);

        // Standard implementation:
        for(int i = 0;i<scores.size();i++) {
            unique.add(scores.get(i));
        }
    }
// Make sure you are not redefining unique in this scope by
// saying HashSet<Integer> unique = .... Otherwise, you are just creating a new
// variable in setUnique, which will vanish after exiting this method. Also make
// sure you are treating scores correctly as an ArrayList, not array
// (you should use scores.get(i) instead of scores[i]).
// When calling unique.add, you don’t need to check whether unique contains this
// element or not. If the element is already there, the add operation will have
// no effect and not throw an error.

// Stores the counts each score into HashMap unique without any duplicates
public void setCounts() {
    for (int i = 0; i < scores.size(); i++) {
        int counter = 0;
        for (int j = 0; j < scores.size(); j++) {
            if (scores.get(i) == scores.get(j)) {
                counter++;
            }
        }
        counts.put(scores.get(i), counter);
    }
}