Midterm 1: Compsci 201

Tabitha Peck

February 12, 2014

Print your name and NetID legibly in ALL CAPITAL letters. Make sure that we can clearly determine L vs. 1 and S vs. 5. It will affect your grade if you do not follow these instructions or we cannot read your name or netID.

Name: ____________________________________________

NetID/Login (CAPITAL LETTERS): ______________

Honor code acknowledgment (signature) ____________________________

This test has 15 pages (with a help page at the end), be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages are imported in any code you write.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>32 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>13 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>85 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1: (Olympics: 32 points)

Your friend Harper is writing a class Country to print a sorted list of gold medals won by a single country for several events in the winter Olympics. Harper wrote code to print the list sorted first by the number of gold medals, with ties broken alphabetically by name of the event. An example output, and the main that calls your code are below.

<table>
<thead>
<tr>
<th>Medal Count</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Alpine Skiing</td>
</tr>
<tr>
<td>3</td>
<td>Curling</td>
</tr>
<tr>
<td>2</td>
<td>Figure Skating</td>
</tr>
<tr>
<td>2</td>
<td>Ski Jumping</td>
</tr>
<tr>
<td>1</td>
<td>Bobsleigh</td>
</tr>
</tbody>
</table>

public static void main(String[] args){
    String[] sport = {"Alpine Skiing", "Bobsleigh", "Curling", "Figure Skating", "Ski Jumping"};
    int[] gold = {3, 1, 3, 2, 2};

    Country usa = new Country();
    System.out.println("Medal Count   Sport");
    usa.medalCount(sport, gold);
}

Part A The medalCount method of class Country below uses the Sport inner class found on the following page. Help Harper complete the method medalCount as indicated by the comments — you should complete the parameters based on the call shown in main above, the fill in the body of the loop (8 points).

//add the appropriate parameters to the method definition

public void medalCount( String[] sportName, int[] goldMedals ){

    Sport[] sportMedalCount = new Sport[sportName.length];
    for(int i = 0; i < sportName.length; i++){

        // add objects to fill sportMedalCount with instances of the inner Sport class
        for(int i = 0; i < sportName.length; i++){
            sportMedalCount[i] = new Sport(sportName[i], goldMedals[i]);
        }
    }

    Arrays.sort(sportMedalCount);

    for(Sport s: sportMedalCount){
        System.out.println(s);
    }
}

Harper wrote the following inner-class, named Sport to help implement medalCount as shown above.

```java
public class Sport implements Comparable<Sport> {
    private String myName;
    private int myGoldMedals;

    public Sport(String name, int goldMedals) {
        myName = name;
        myGoldMedals = goldMedals;
    }

    public int compareTo(Sport o) {
        int medalDif = o.myGoldMedals - this.myGoldMedals;
        if (medalDif == 0) {
            return this.myName.compareTo(o.myName);
        }
        return medalDif;
    }
}
```

Given the code above for inner class Sport the output is:

```
Medal Count   Sport
Country$Sport@25d398dd
Country$Sport@be05661d
Country$Sport@5e947d47
Country$Sport@8988901d
Country$Sport@8a470a3e
```

**Part B** Explain to Harper what is currently being printed. (2 points)

The memory address of each Sport Object.

**Part C** The class Sport is using the default implementation of a particular method — this default implementation is the cause of the output shown. What is the method that is not implemented in Sport and where is the default implementation called in the code above? (2 points)

The method is .toString() and it is called with System.out.println(s);

**Part D** Add code for the (missing) method below so that when added to the current implementation, Harper’s code prints the output displayed at the beginning of this question. (4 points)

```java
public String toString() {
    return myGoldMedals + " " + myName;
}
```
Everyone knows that curling is the best sport at the winter olympics. You want to help Harper change the code so that it sorts winter events as described, by number of gold medals, such that **Curling is always printed first**, regardless of the medal count for Curling. You must also break ties in **reverse alphabetical order**. The original output would then be:

<table>
<thead>
<tr>
<th>Medal Count</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Curling</td>
</tr>
<tr>
<td>3</td>
<td>Alpine Skiing</td>
</tr>
<tr>
<td>2</td>
<td>Ski Jumping</td>
</tr>
<tr>
<td>2</td>
<td>Figure Skating</td>
</tr>
<tr>
<td>1</td>
<td>Bobsleigh</td>
</tr>
</tbody>
</table>

**Part E** What method would you change to make Harper’s code sort as defined directly above? (2 points)

```java
compareTo(Sport o);
```

**Part F** Rewrite the code for the method named above such that the code sorts with:

- curling always printed first
- then sorted by gold medals
- with ties broken by reverse alphabetically by sport name.

Write your method below. (4 points)

```java
public int compareTo(Sport o) {
    if(this.myName.equals("Curling")){
        return -1;
    }
    if(o.myName.equals("Curling")){
        return 1;
    }
    int medalDif = o.myGoldMedals - this.myGoldMedals;
    if(medalDif == 0){
        return o.myName.compareTo(this.myName);
    }
    return medalDif;
}
```
Harper wants to do more than just sort medals with the `Sport` class. Harper adds the methods below to the `Sport` class to ensure that all `Sport` objects are different.

```java
public int hashCode(){
    return myGoldMedals;
}

public boolean equals(Object obj){
    if(obj==this){
        return true;
    }
    if(obj==null || obj.getClass() != this.getClass()){,
        return false;
    }
    Sport temp = (Sport)obj;
    if(this.myName.equals(temp.myName)){
        if(this.myGoldMedals == temp.myGoldMedals){
            return true;
        }
    }
    return false;
}
```

Harper uses the code below to determine whether all `Sport` objects are different:

```java
HashSet<Sport> set = new HashSet<Sport>();
for(Sport s: sportMedalCount){
    set.add(s);
}
if (set.size() == sportMedalCount.length){
    System.out.println("all objects are different");
} else {
    System.out.println("at least one duplicate");
}
```
**Part G** Suppose that the `HashSet` class is implemented with a simple hash table such as we discussed in class. Show how the five `Sport` objects would be stored in such a table by diagramming where each object `Alpine Skiing/3`, `Bobsleigh/1`, `Curling/3`, `Figure Skating/2`, and `Ski Jumping/2` would be stored in the hashtable using the `hashCode` function given on the previous page where the `HashSet` hash-table implementation uses $key = \text{hashCode}\%10$ to determine where an object belongs in the hash table. You may draw outside of the table. (5 points)

<table>
<thead>
<tr>
<th>Key</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**Part H** The code on the previous page works correctly, but the implementation of either `hashCode` or `equals` might be less efficient than desirable if many, many `Sport` objects were being used. Explain which method might cause this inefficiency, why it might lead to the inefficiency, and a simple way to change the method so that the code above would be efficient even with many, many `Sport` objects. (5 points)

Which method: `.hashCode()`

Why is it inefficient?
The `hashCode` method is only taking into account the `myMedalCount` variable. All sports with the same medal count will collide in the hashTable, even if the Sport is different. The collisions will cause the inefficiency.

Rewrite the method so that the code would be more efficient:

```java
public int hashCode(){
    return 13*myGoldMedals + 31*myName.hashCode();
}
```
PROBLEM 2: (APT Time: 20 points)

The problem description for the APT IsomorphicWords can be found at the end of this problem.

Your friend, Jessie is working on the Isomorphic Words APT and would like your help. Earlier in the week you went to the Link, and the amazing UTA helped you complete the following helper method.

```java
private String ALPH = "abcdefghijklmnopqrstuvwxyz";

public String makeMapWord(String word){
    HashMap<Character, Character> wordMap = new HashMap<Character, Character>();
    for(int i = 0; i < word.length(); i++){
        if(!wordMap.containsKey(word.charAt(i))
            wordMap.put(word.charAt(i), ALPH.charAt(wordMap.size()));
    }
    StringBuilder s = new StringBuilder(); //works just like a string, but faster
    for(int i = 0; i < word.length(); i++){
        char mapLetter = wordMap.get(word.charAt(i));
        s.append(mapLetter); //append a letter to the end of s
    }
    return s.toString();
}
```

In the code above, the call `makeMapWord("roller")` would return "abccda".

You can’t show Jessie your code since that would be against the honor code, but you want to help Jessie by explaining what your code does. Help Jessie by answering the following questions.

Part A In the code above, what is the purpose of the variable ALPH? (2 points)
ALPH is a global variable that stores all of the letters in the alphabet and is used to make the map such that each letter is a String is mapped to a distinct letter in the alphabet.

Part B Explain in words what happens in the following line of code. (4 points)
`wordMap.put(word.charAt(i), ALPH.charAt(wordMap.size()));`

The character at location i is put into the map as the key. The value associated with the key is the n^th letter of the alphabet such that n is the size of the map. (the first input is mapped to a, the second input is mapped to b, etc.)

Part C Briefly explain to Jessie the purpose of your `makeMapWord` method? (2 points)
The `makeMapWord` method converts each String into an isomorphic word using the same mapping system. These isomorphic words will then be used to count the total number of isomorphic words within each pattern.

Part D What would the call `makeMapWord("hello")` return? (2 points) "abccd"
Part E Now that Jessie understands the helper method, you can start working on countPairs, the method that returns the total number of isomorphic pairs. Complete the code as designated by the comments such that you will get all green on the IsomorphicWords APT. You must use the instance variable myMap, a HastMap that maps strings to ints and will be used as a frequency table. In your implementation, you must first populate myMap based on the strings returned by makeMapWord(word) in the first for-loop. Then using myMap, you can compute and return the number of isomorphic word pairs.

```java
private HashMap<String, Integer> myMap = new HashMap<String, Integer>();

public int countPairs(String[] words) {
    for(String word: words) {
        String mapWord = makeMapWord(word);

        // fill myMap below (5 points)
        if (myMap.containsKey(mapWord)) {
            myMap.put(mapWord, myMap.get(mapWord) + 1);
        } else {
            myMap.put(mapWord, 0);
        }
    }

    // Compute the final answer below (5 points)
    int answer = 0;
    for (String iso: myMap.keySet()) {
        int count = myMap.get(iso);
        answer += (count * count + count) * 0.5;
    }

    return answer;
}
```
Isomorphic Words

Problem Statement

Two words are called isomorphic if the letters in one word can be remapped to get the second word. Remapping a letter means replacing all occurrences of it with another letter. The ordering of the letters remains unchanged. No two letters may map to the same letter, but a letter may map to itself.

For example, the words "abca" and "zbxz" are isomorphic because we can map 'a' to 'z', 'b' to 'b' and 'c' to 'x'.

Given a string[] words, return how many (unordered) pairs of words are isomorphic.

Constraints

- words will contain between 2 and 50 elements, inclusive.
- Each element of words will contain between 1 and 50 lowercase letters ('a'-'z'), inclusive.
- All elements of words will have the same length.
- There will be no duplicates in words.

Examples

1. 

   ("abca", "zbxz", "opqr")

   Returns: 1

   "abca" and "zbxz" are isomorphic, but none of the two is isomorphic with "opqr".
**PROBLEM 3 :** *(Big-Oh: 20 points)*

For each of the following, give the running time in terms of the parameter \( n \) in big-Oh notation. You must justify your answer. (2 points for each running time, and 2 points for each justification)

```java
public void numberZero(int n){
    int foo = 0;
    for(int i = 0; i < 1000000; i++){
        System.out.println(n);
    }
}
```

**Big-Oh: O(1)**  
Justification: The loop runs a constant (1000000) number of times. The print call inside the loop is constant.

```java
public void numberOne(int n){
    int[] intArray = new int[n];
    for(int i = 0; i < n; i+=2){
        intArray[i] = i;
    }
}
```

**Big-Oh: O(n)**  
Justification: The loop runs \( n/2 \) times, inserting into an array is constant. Drop the constant from the big-oh statement.

```java
public void numberTwo(int n){
    ArrayList<Integer> list = new ArrayList<Integer>();
    for(int i = 0; i < n; i++){
        list.add(i);
    }
    for(int i = 0; i < n; i++){
        if(list.contains(i)){
            System.out.println(i);
        }
    }
}
```

**Big-Oh: \( O(n^2) \)**  
Justification: The first for-loop runs \( n \)-times, and adding to a list is constant. Loop 1 is \( O(n) \). The second loop runs \( n \) times, but contains on a list of size \( n \) is \( O(n) \). Hence, \( O(n^2) + O(n^2) = O(n^2) \)
public void numberThree(int n){
    HashSet<Integer> set = new HashSet<Integer>();
    for(int i = 0; i < n; i++){
        set.add(i);
    }
    for(int i = 0; i < n; i++){
        if(set.contains(i)){
            System.out.println(i);
        }
    }
}

Big-Oh: $O(n)$

Justification: First loop runs n-times and adding to a HashSet is constant, $O(n)$. The second loop runs n times and contains on a SashSet is constant, $O(n)$. $O(n) + O(n) = O(n)$

public void numberFour(int n){
    ArrayList<Integer> list = new ArrayList<Integer>();
    for(int i = 0; i < n; i++){
        list.add(i);
    }
    for(int i = 1; i <= n; i++){
        System.out.println(list.remove(0));
    }
}

Big-Oh: $O(n^2)$

Justification: From above, first loop is $O(n)$. Second loop runs n times. Remove costs n for the first run, n-1 for the second, n-2, etc. That is $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$
PROBLEM 4 : (Distinct value: 13 points)

You are given an array of n integers, for some large n. Your array has one value that is distinct. You want to find the single distinct value in linear, that is O(n), time.

Example: int[] a = {0, 0, 1, 2, 1, 1, 3, 3}. The distinct value is 2.

The solution below does run in O(n) time.

```java
private int setUnique(int[] a) {
    HashSet<Integer> set = new HashSet<Integer>();
    HashSet<Integer> xtra = new HashSet<Integer>();
    for(int x : a) {
        if (! set.add(x)){
            xtra.add(x);
        }
    }
    for(int x : xtra){
        set.remove(x);
    }
    for(int x : set){
        return x;
    }
    return Integer.MIN_VALUE;  // this line never reached if unique element found
    //Integer.MIN_VALUE is the minimum value an int can have in Java
}
```

Part A (4 points)

Explain why the solution above runs in O(n) time by justifying why each loop is O(n) and why that makes setUnique O(n)

Loop 1: Loop runs the length of a, i.e. n times. adding to a set is constant. O(n)

Loop 2: Loops length of extra times. This is worst case n/2 − 1 times and remove on a hashSet is constant. This is O(n).

Loop 3: There will only be one item in set, this is O(1).

O(n) + O(n) + O(1) = O(n)
Part B (9 points)
The solution above uses two sets, more resources than using one set or one map. Write another solution that still runs in O(n) time, but also uses only one map or one set. Justify why your solution is O(n).

//input: a is an integer array with one distinct value
//output: the value of the distinct integer from the input array, a

public int findUnique(int[] a){
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for(int i: a){
        if(map.containsKey(i)){
            map.put(i, map.get(i) + 1);
        }
        else{
            map.put(i, 1);
        }
    }
    for(int i: map.keySet()){  
        if(map.get(i) == 1){
            return i;
        }
    }
    return a[0];
}

Loop 1 runs n-times (the length of a). Contains and put on a HashMap are constant. This is O(n).
Loop 2 runs at most \(\frac{n}{2} - 1\) times and get on a HashMap is constant. This is O(n).
\(O(n) + O(n) = O(n)\)
String

- `.length()` Get the length of the String. \(O(1)\).
- `.charAt(i)` Get the char at index \(i\). \(O(1)\).
- `.split(" ")` Split a string by spaces and store it in a string[].
- `.substring(i, j)` Get the substring between indices \(i\) and \(j\). Index \(i\) is inclusive, and index \(j\) is exclusive. \(O(1)\). For example:
  
  ```java
  String x = "abcdefg";
  String y = x.substring(2, 4);
  // y now has the value "cd"
  ```

ArrayList<T> // Where T is a type, like String or Integer

- `.add(i, X)` Add element \(X\) to the list at index \(i\). If no \(i\) is provided, add an element to the end of the list. Adding to the end runs in \(O(1)\).
- `.get(i)` Get the element at position \(i\). Runs in \(O(1)\).
- `.set(i, X)` Set the element at position \(i\) to the value \(X\). \(O(1)\).
- `.size()` Get the number of elements. \(O(1)\).

HashSet<T> // Where T is a type, like String or Integer

- `.size()` Compute the size. \(O(1)\).
- `.add(X)` Add the value \(X\) to the set. If it’s already in the set, do nothing. \(O(1)\).
- `.contains(X)` Return a boolean indicating if \(X\) is in the set. \(O(1)\).
- `.remove(X)` Remove \(X\) from the set. If \(X\) was not in the set, do nothing. \(O(1)\).

HashMap<K, V> // Where K and V are the key and value types, respectively.

- `.size()` Compute the size. \(O(1)\).
- `.containsKey(X)` Determines if the map contains a value for the key \(X\). To get that value, use `.get()`. \(O(1)\).
- `.get(X)` Gets the value for the key \(X\). If \(X\) is not in the map, return null. \(O(1)\).
- `.put(k, v)` Map the key \(k\) to the value \(v\). If there was already a value for \(k\), replace it. \(O(1)\).
- `.keySet()` Return a Set containing the keys in the map. Useful for iterating over. \(O(1)\).

To iterate over a HashSet<T>, use

```java
for (T v : nameOfSet) {
    // v is the current element of the set.
}
```

This can be combined with HashMap’s `.keySet()` to iterate over a HashMap.

\[
\sum_{i=1}^{n} i = \frac{n(n+1)}{2}
\]