Test 1: Compsci 100

Owen Astrachan

February 22, 2010

Name: _____ (1 point)

NetID/Login: _____

Honor code acknowledgment (signature) _____

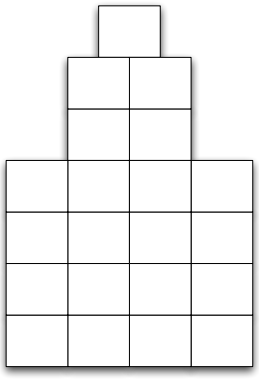
	value	grade
Problem 1	20 pts.	
Problem 2	16 pts.	
Problem 3	8 pts.	
Problem 4	18 pts.	
Problem 5	18 pts.	
TOTAL:	80 pts.	

This test has 15 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes. The last page is blank, if you use it make a note for the problem.

In writing code you do not need to worry about specifying the proper `import statements`. Don't worry about getting method names exactly right. Assume that all libraries and packages we've discussed are imported in any code you write.

PROBLEM 1 : (***Tower of Power (20 points)***)

The picture below shows a *3-tower*. The bottom square is a $2^2 \times 2^2$ square which is topped by a $2^1 \times 2^1$ square which is topped by a $2^0 \times 2^0$ square. In an *n-tower* there are n squares, the bottom square is a $2^{n-1} \times 2^{n-1}$ square, and it is otherwise similar to the *3-tower* in terms of each of the n squares being formed by an incrementally smaller power-of-two square than the one below it.



Part A (3 points)
The *3-tower* is 7 units tall. How tall is a *5-tower*? Justify your answer briefly.

Part B (3 points)
How tall is a *10-tower*? Justify briefly.

Part C (3 points)
A *3-tower* is made of 16+4+1 = 21 blocks as shown. How many blocks are there in a *5-tower*?

Part D (3 points)
By definition we define `B(n)` to be the number of blocks in an *n-tower* so that `B(1) = 1`, `B(2) = 5`, and and `B(3) = 21`. Explain why the following relationship/equation is true for any $n > 1$.

$$B(n) = B(n-1) + (2^{2n-2})$$

Part E (5 points)

Write a method `blocks` so that `blocks(n)` returns the number of blocks in an *n-tower*, e.g., so that `blocks(3)` returns 21 and `blocks(2)` returns 5. To earn 5/5 points you must write the method recursively. You can earn 4/5 points for writing a correct iterative method. To raise a number to a power, use `Math.pow`, e.g., `Math.pow(2,5)` is 32.

```
public int blocks(int n){









}
```

Part F (3 points)

Using big-Oh (in terms of n), how **tall** is an *n-tower*? For example, a *3-tower* is 7 units tall. Justify your answer briefly. The correct answer is either n, n^2, n^4, or 2^n.

PROBLEM 2 : (*Apollo O-Notation* 16 points)

For each of the methods below indicate **both** the *running time* of the method, in terms of n, using O-notation and the *value returned*, using O-notation. You must justify your answers for credit. In all examples assume n is positive.

For part A, you're given the correct O-notation for both run-time and value-returned, and you must justify these. For the other parts you must both provide the O-notation and the justification as to why they're correct.

Part A (4 points)

The running time of `calc(n)` is $O(n)$, and the value returned is $O(n^2)$. Justify this.

```
public int calc(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += k;
    }
    for(int k=0; k < n; k++){
        sum += k;
    }
    return sum;
}
```

Part B (4 points)

Provide big-Oh expressions for both the running time and the value returned for `clunk(n)` below. Justify these.

```
public int clunk(int n){
    int sum = 0;
    for(int j=0; j < n; j++){
        for(int k=0; k < j; k++){
            sum++;
        }
    }
    return sum;
}
```

Part C (4 points)

Provide big-Oh expressions for both the running time and the value returned for `goduke(n)` below. Justify these. You can assume that n is a power of two.

```
public int goduke(int n){
    int sum = 0;
    for(int k=1; k <= n; k = k * 2){
        sum += k;
    }
    return sum;
}
```

Part D (4 points)

Provide big-Oh expressions for both the running time and the value returned for `stuff(n)` below. Justify these.

```
public int stuff(int n){
    int sum = 0;
    for(int k=0; k < n; k++){
        sum += n;
    }
    for(int k=0; k < n; k++){
        sum += 2;
    }
    for(int k=0; k < n; k++){
        sum += 3;
    }
    return sum;
}
```

PROBLEM 3 : (***mapquest (8 points)***)

Information about songs stored in several students' mp3 players is stored in an array. Each array element contains a *play list*, a comma-delimited list of titles for one student. Write the method `topSong` that returns the title of the song that appears in the most play lists. If there's a tie for the song that appears in the most play lists return the song with the alphabetically/lexicographically smallest title. No song will appear more than once in the same play list.

For example, for the array of three play lists below, the song *Tik Tok* appears in all three play lists, no other song appears in all the play lists, so `topSong` should return the string `"Tik Tok"`.

```
{
  "Tik Tok,Imma Be,Bad Romance,Who Are You" ,
  "How Low,Tik Tok,Thunder Road",
  "Need You Now,Tik Tok,Stupidity"
}
```

Complete method `topSong` below.

```
String topSong(String[] playlists} {









}
```

PROBLEM 4 : (*Counting Medals (18 points)*)

The APT *Medal Table* is available as a handout with the test. In this problem you'll be asked to fill in missing pieces of a solution that is all green before the pieces are removed.

The idea realized in the code is to keep a map of countries to medal-tables. The country is a three-character string. The medal table is a three-element `int[]` storing the number of gold, silver, and bronze for a country in indexes 0, 1, and 2, respectively. For example the key-and-value pairs below show USA with 2 gold, 3 silver, and 5 bronze medals; SUI with 1 gold, 2 silver, and 4 bronze medals; and CAN with 3 gold, 1 silver, and 0 bronze medals.

```
USA  ==> [2,3,5]    SUI ==> [1,2,4]     CAN ==> [3,1,0]
```

First you'll be shown the code with three missing pieces, then you'll be asked to fill in the pieces in parts A, B, and C of this problem.

The code uses the `java.util` class `Map.Entry` which stores the (key,value) pairs in a map. You'll see that methods `.getKey()` and `.getValue()` are used to access the key and value of such an object in the code below. The `Map.Entry` class is part of the `java.util` package.

```
public class MedalTable {

    // questions about inner class APTComp are in part C

    public class APTComp implements Comparator<Map.Entry<String, int[]>>{

        public int compare(Map.Entry<String, int[]> a, Map.Entry<String, int[]> b) {

            int gdiff = a.getValue()[0] - b.getValue()[0];
            int sdiff = a.getValue()[1] - b.getValue()[1];
            int bdiff = a.getValue()[2] - b.getValue()[2];

            if (gdiff == 0 && sdiff == 0 && bdiff == 0){
                return a.getKey().compareTo(b.getKey());
            }
            if (gdiff != 0) return -gdiff;
            if (sdiff != 0) return -sdiff;
            return -bdiff;
        }

    }
```

(code continued next page)

```
public String[] generate(String[] results){

    Map<String,int[]> map = new HashMap<String,int[]>();
    for(String s : results){
        String[] all = s.split(" ");
        for(int k=0; k < all.length; k++){
            String ccode = all[k];

            // fill in code for Part A
        }
    }

    ArrayList<Map.Entry<String, int[]>> list = new ArrayList<Map.Entry<String, int[]>>();
    for(Map.Entry<String, int[]> entry : map.entrySet()) {
        list.add(entry);
    }
    Collections.sort(list, new APTComp());

    String[] ret = new String[list.size()];
    for(int k=0; k < list.size(); k++){
        ret[k] = list.get(k).getKey();


        // fill in code for part B


    }
    return ret;
}
```

Questions start on next page

Part A (6 points)

Code is mssing from the nested for loop near the beginning of method `generate`. The missing code should update the appropriate medal count in the map for the key/country whose code is stored in variable `ccode`. Write code that replaces the comment labeled *fill in code for Part A* so that it does two things:

1. if `ccode` is not a key in the map, it is stored as a key with a three-element `int[]` which will represent the medal table.
2. update the appropriate element of the `int[]` associated with `ccode` to represent one more medal, either gold, silver, or bronze, for the `ccode`

Complete the code below.

```
Map<String,int[]> map = new HashMap<String,int[]>();
for(String s : results){
    String[] all = s.split(" ");
    for(int k=0; k < all.length; k++){
        String ccode = all[k];
        // fill in code for Part A









    }
}
```

Part B (6 points)

The array returned from `generate` is an appropriately sorted arary of strings where each string is a country code followed by the number of gold, silver, and bronze medals for that country code. The code below stores the country code in the appropriate array element, but the medal counts must be extracted and appended/concatenated. Complete the code below so that `ret[k]` represents the correct String for each array element; you'll do this using an appropriate call to `list.get(k).getValue()` with other code.

```
String[] ret = new String[list.size()];
for(int k=0; k < list.size(); k++){
    ret[k] = list.get(k).getKey();
    // fill in code for part B









}
return ret;
```

Part C (6 points)

The comparator `APTComp` used to sort the `ArrayList` is coded correctly. First explain why the value returned is negated in the code `if (gdiff != 0) return -gdiff` and in each of the return statements that follow it.

Most US websites providing medal counts sort countries by the total number of medals earned, not first by gold, then by silver, then by bronze (perhaps because the US has the most total number of medals). Complete the new version of the `compare` method below so that it sorts only by total number of medals, then by country-codes alphabetically if the medal totals are the same.

```
public int compare(Map.Entry<String, int[]> a, Map.Entry<String, int[]> b) {

    int[] amedals = a.getValue();
    int atotal = amedals[0] + amedals[1] + amedals[2];









}
```

PROBLEM 5 : (*Reasoning About Lists (18 points)*)

The method `duplicate` in the code shown below changes parameter `list` so that it's *doubled in place* – for example, the list

```
("ape", "bat", "cat", "dog")
```

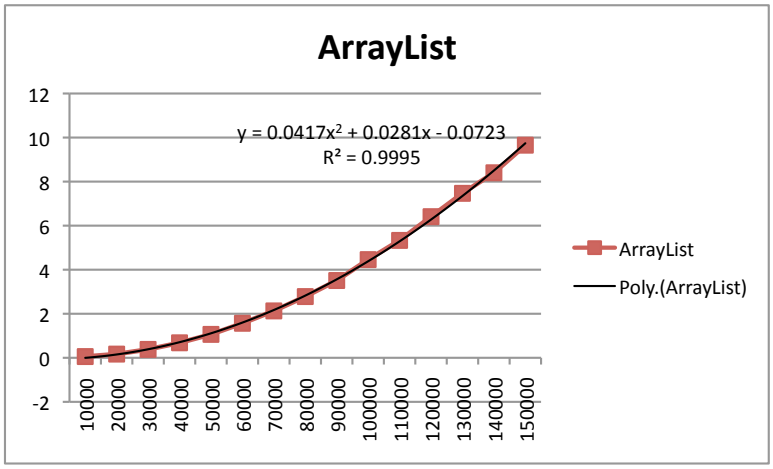
is changed to the list below as a result of the call `duplicate(list)`.

```
("ape", "ape", "bat", "bat", "cat", "cat", "dog", "dog")
```

In this problem, the method `duplicate` is called with both an `ArrayList` and a `LinkedList` as parameters – the times for duplicating the different lists are shown tabularly and graphically below the code where the size of the list varies from 10,000 to 150,000 strings. The graphic is only for the `ArrayList` call to `duplicate` since the `LinkedList` timings are too small in comparison.

```
public void duplicate(List<String> list){
    ListIterator<String> iter = list.listIterator();
    while (iter.hasNext()){
        String s = iter.next();
        iter.add(s);
    }
}
```

size (10^3)	link	array
10	0.001	0.056
20	0.001	0.168
30	0.002	0.381
40	0.002	0.681
50	0.003	1.063
60	0.003	1.567
70	0.005	2.129
80	0.005	2.780
90	0.006	3.509
100	0.007	4.458
110	0.008	5.33
120	0.009	6.408
130	0.010	7.467
140	0.010	8.401
150	0.010	9.654



Part A (4 points)

Using the same code on the same computer how much time will it take to duplicate both an `ArrayList` and a `LinkedList` list with 1,000,000 (one million) values. Justify your answers (your answer will be considered approximate, we're looking for close enough.) Don't rely on the coefficients of the quadratic shown, use reasoning based on the empirical timings.

(continued)

Part B (4 points)

Using big-Oh, what is the complexity of duplicating both an N-element `ArrayList` and `LinkedList` using the code above. Justify your two answers empirically, by making explicit references to the timings.

Part C (4 points)

Although we haven't discussed `ListIterator` in class, it runs quickly for `LinkedList` because new elements are added to the list, in place, as the iteration over the list takes place. Explain why doing this is fast for `LinkedList` and slow for `ArrayList` based on how these classes are implemented. Make explicit references to the implementations of each class in your reasoning, assuming the implementations are reflected by the names of the classes and using the empirical timings to justify your answers.

(continued)

Part E (6 points)

The code below is an alternate version of the `duplicate` code above. This code correctly modifies `list` so that it is doubled-in-place — its behavior is exactly the same as the code that uses the `ListIterator`, but the timings are different.

```
public void duplicate2(List<String> list){

        int originalSize = list.size();
        list.addAll(list);

        for(int k=originalSize-1; k >= 0; k--){
            String current = list.get(k);
            list.set(2*k, current);
            list.set(2*k+1,current);
        }
    }
```

Explain why the `for` loop runs down to zero rather than up from zero (which would not work).

Provide a justification for why this code runs very quickly for `ArrayList` parameters and very slowly for `LinkedList` parameters. On the same computer on which the first timings were made, the `LinkedList` duplication takes 2.28 secionds for 20,000 strings and 18.7 for 40,000 strings whereas the `ArrayList` timings are all below 0.01 seconds.

Explain the timings by providing a reason for them based on the code and your understanding of how the classes `LinkedList` and `ArrayList` are implemented.

(nothing on this page)