Name: _____

USC NetID (e.g., `ttrojan`): _____

CS 455 Midterm Exam 2
Spring 2017 [Bono]

Tuesday, April 4, 2017

There are 7 problems on the exam, with 64 points total available. There are 10 pages to the exam (5 pages **double-sided**), including this one; make sure you have all of them. If you need additional space to write any answers or scratch work, pages 8, 9 and 10 are left blank for that purpose. If you use those pages for answers you just need to direct us to look there. *Do not detach any pages from this exam.*

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Java `String` class (selected methods)

`char charAt(int index)`
Gets the char at position index (counting from 0)

`String substring(int beginIndex, int endIndex)`
Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`.

`String substring(int beginIndex)`
Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the end of this string.

`int length()` The number of characters in the String

`int isEmpty()` Returns true iff the String is empty

Problem 1 [6 pts. total]

Consider a hash table with chaining using an array called `hashArray`. The keys in this hash table are strings. Suppose we have a key stored in variable `ourKey`, whose hash value is `52`. (i.e., `hash(ourKey)` returns `52`).

Part A (2). Circle the answer that best matches:

If `hashArray[52]` ***is not*** empty we know that:

i. `ourKey` *is definitely present* in the table

ii. `ourKey` *may be present* in the table

iii. `ourKey` *is definitely not present* in the table

Part B (2) . Fill in the box below with an answer from the above choices (i, ii, iii) that best matches for the following:

If `hashArray[52]` ***is*** empty we know that: ☐

Part C (2). Suppose we have a second key stored in variable `ourKey2`, whose hash value is also `52`. What do we know about the relationship between `ourKey` and `ourKey2`:

a. they are the same object (i.e., two object variables referring to the same object)

b. they have the same value

c. `ourKey2` comes right after `ourKey` in an alphabetical ordering of all the table elements

d. none of the above

Problem 2 [7 pts. total]

Number each of the following time complexities in order from fastest running to slowest running (e.g., number 1 is fastest, and number 7 is slowest):

_____ O(n) _____ O(1)

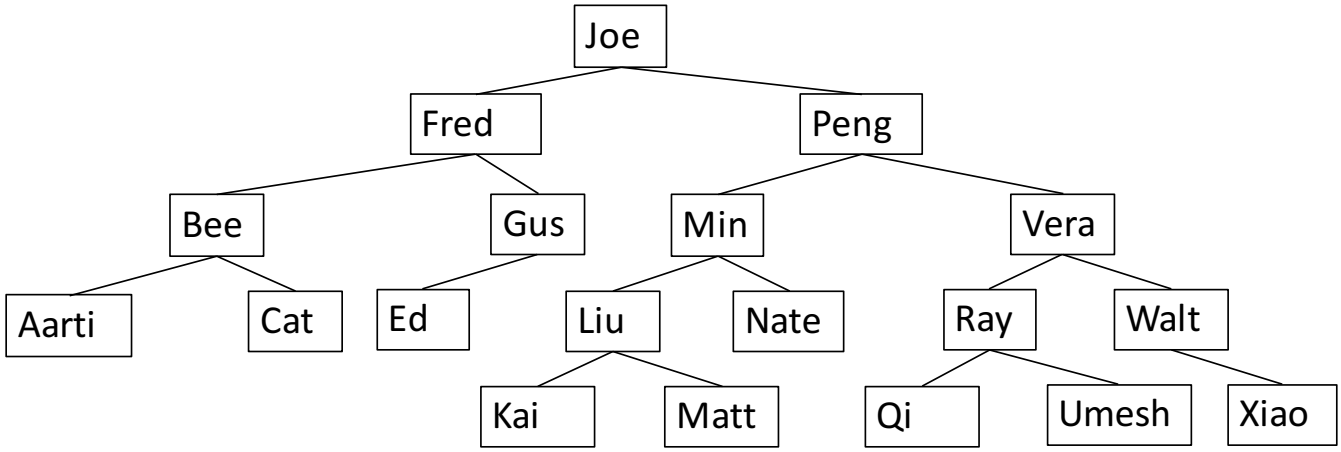
_____ O(n^3) _____ O($\log n$)

_____ O($n \log n$) _____ O(n^2)

_____ O(2^n)

Problem 3 [8 pts. total]

Part A. Consider the following balanced binary search tree, whose root is shown nearest the top of the page (i.e., it's not a sideways tree):



For each of the following lookups from the tree shown above, give the sequence of keys that the target key would have to be compared with to do the lookup.

Part A. lookup *Min*

Part B. lookup *Hank*

Part C. lookup *Qi*

Part D. lookup *Sally*

Problem 4 [5 pts.]

Suppose we want to write a static method `randomStats(n)` that generates *n* random numbers and gives us some information on how random the values generated actually are. Each random value generated will be in the range [0, 9] (i.e., we could generate one with the call `rand.nextInt(10)`, where `rand` is a `Random` object). `randomStats(n)` will print out each outcome and the number of times that outcome appeared.

Do not implement `randomStats`.

What data structure or Java class we have studied this semester would be most appropriate to use in this method to get the results efficiently? (I.e., besides `Random` itself.) If it is some kind of container, make clear what type of objects are being contained.

Describe in more detail below what data is being stored in your data structure. Be specific (e.g., how do these values relate to the problem we are solving).

Problem 5 [3 pts.]

Show what would be in the following array after the first *four* passes of the insertion sort algorithm (sorting in increasing order). Hint: For an n-element array, it's completely sorted after n-1 passes.

array before call to insertion sort:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
17	12	2	7	15	5	36	4	18	16

(array indices shown above the line)

array after four passes of insertion sort:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Problem 6 [15 points]

Write a *recursive* method `removeXs`, that removes all instances of the character '`x`' from the given `String`. The only `String` methods you may use for this problem are shown on the cover of the exam; you may also use the '+' operator. A solution that doesn't use recursion or has a loop will receive little to no credit. Notes: solutions which include calls to `substring` can receive full credit, and (as usual) you are allowed to define a helper method to do the actual recursion. Examples:

str	***removeXs(str)***
`""`	`""`
`"ab"`	`"ab"`
`"axb"`	`"ab"`
`"xabcxyzxx"`	`"abcyz"`
`"xxx"`	`""`

```
// returns a String like str, but with all the 'x' characters removed.
// (uses recursion)
public static String removeXs(String str)
```

Problem 7 [20 points]

Consider an additional constructor for the `Polynomial` class from PA2 that takes an `ArrayList` of `Term`'s as an argument. The parameter would be allowed to have data for an unsimplified polynomial with terms in any order, i.e., just like the list of terms entered by the user for the `create` command of the `PolynomialCalculator`. With this constructor we could have implemented "doCreate" without repeated calls to the `Polynomial add` method.

Here is an example of valid data for this constructor (a term is shown as an ordered pair, (coeff, expon) in the following):

```
[ (3,2),  (7,0),  (10,2),  (2,1),  (5,3),  (-5,1),  (3,1),  (0,4) ]
```

After passing that data to our new constructor, the resulting polynomial (shown in `toFormattedString` format) would be:

```
5.0x^3 + 13.0x^2 + 7.0
```

Implement this new constructor, but using a *different* representation for the Polynomial than we used previously: the new representation is a Map that maps from exponents to the coefficient for that exponent. Just like before, we will store only non-zero terms. Further details given in the representation invariant comments next to the instance variable definition on the next page. Space for your answer is given there.

Here is a reminder of some `Term` and `Map` methods:

`Term` class

`Term()`	Create the term whose coefficient and exponent are both 0
`Term(double coeff, int expon)`	Create the term with the given coefficient and exponent
`double getCoeff()`	Get the coefficient of this term
`int getExpon()`	Get the exponent of this term
`String toString()`	Return string version of object for debugging purposes

`Map<KeyType, ValueType>` interface (selected methods)

The classes that implement this interface are: **`TreeMap`** and **`HashMap`.**

`ValueType put(key, value)`
: Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced by the specified value. Returns the previous value associated with specified key, or `null` if there was no mapping for key.

`ValueType get(key)`
: Returns the value to which this map maps the specified key or `null` if the map contains no mapping for this key.

`ValueType remove(key)`
: Removes the mapping for this key from this map if it is present, otherwise returns `null`.

`int size()`	Number of key-value mappings in this map.
`boolean isEmpty()`	Returns true if this map contains no key-value mappings.

Problem 7 (cont.)

Complete the implementation of the constructor below – more details of this problem are on the previous page. Do not implement the rest of the class.

```
public class Polynomial {

   private Map<Integer, Double> polyData;

   //   Representation invariant:
   //     Maps from an each exponent to its coefficient.
   //     Values are in decreasing order by exponent
   //     No terms with coefficient 0 are stored in the Map


   //  Create a polynomial from the arraylist of terms given.
   public Polynomial(ArrayList<Term> terms) {

                              // the following line creates an empty TreeMap.
                              // the parameter indicates the TreeMap will be
                              // organized in decreasing order by key.
      polyData = new TreeMap<Integer, Double>(Collections.reverseOrder());
```

Extra space for answers or scratch work.

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.