**COMP 313-413 IOOD Quiz 0 (Course Prerequisites) – 10 points – <u>Double-Sided!!</u>          Pg. 2 of 2**

**Dr. Yacobellis • Fall 2016 • Tuesday, August 30 • 15 minutes**

**<u>Your Name:</u>**          **<u>Answer sheet</u>**

**Background (N/A)**

a) What programming languages have you programmed with in the past?

b) Approximately how many total lines of code have you written in each of these languages?

c) What libraries, frameworks, and/or APIs (application programming interfaces) have you used?

d) What integrated development environments (IDEs) have you used?

e) What operating system(s) will you be using in this class, and what version(s)?
**Note: If you plan to use a Mac, I strongly recommend upgrading for free to OS X El Capitan or later.**

**Problem 1 (1.5 points)**

a) To implement an electronic phone book that allows you to **look up a number given a name**, **which** of the following abstract data types **would work <u>best</u>**? (Circle exactly one.)

[✓ • map]     • priority queue     • queue     • set     • stack

**b) To model the checkout line in a supermarket (first-come, first-served), which of the following abstract data types would work <u>best</u>? (Circle exactly one.) first-come, first served = FIFO = queue**

• map     • priority queue     [✓ • queue]     • set     • stack

**c) Which of the following data structures are good choices for implementing an efficient <u>set</u> abstract data type? (Circle one or more.) Sets do not allow duplicate entries (check each addition to make sure).**

• array-based list • binary tree (non-search) • [✓ binary search tree] • [✓ hash table] • linked list

**Problem 2 (2 points)**

You are given an empty stack, **s**, upon which the following operations are performed <u>in this order</u>:
**Stacks are last-in, first-out (LIFO); peek() looks at the top entry but doesn't remove it (pop() does).**
s.push(5), s.push(10), s.peek(), s.pop(), s.push(9), s.pop(), s.push(1), s.push(3), s.peek()

a) How many items are on the stack at the end of this sequence?          **<u>3</u>**

b) In what order are the items on the stack arranged, from top to bottom?          **<u>3, 1, 5</u>**

**Problem 3 (2.5 points)**

For each of the following use cases, what is the <u>worst-case asymptotic order of complexity</u> (big-Oh as a formula of **n**) of the best-known algorithm? For each, specify something like O(n), O(n log n), O($n^2$), etc.

a) Looking up a name in an (alphabetically <u>ordered</u>) phone book of **n** names: **<u>O(log n) à binary search</u>**

a) Looking up a name in an (alphabetically ordered) phone book of **n** names: **O(log n) à binary search**

b) Looking up a name in an unordered sequence of **n** names: **O(n) à worst case looks at all of them**

**also OK: O(n log n) à sort first (item e)**

c) Finding the median value in an unordered sequence of **n** numbers: **__O(n log n) à sort first (item e)**

d) Finding the largest value in an unordered sequence of **n** numbers: **O(n) à must process all entries**

**also OK: O(n log n) à sort first (item e)**

e) Putting an unordered sequence of **n** numbers in sorted order: **O(n log n) à heap sort or merge sort**

**Problem 4 (4 points)**

Your job is to **read one word per line from the standard input** and keep track of how many times each "word" occurs in the input. After the end of the input is reached, print how many times each unique word has occurred (in no particular order). *Use any language or pseudocode (preferred); focus on concepts, not syntax*. For example, if the input is (**assuming each word on a separate line**):

hello hello world goodbye hello world

then one possible output is:

```
world 2
goodbye 1
hello 3
```

For full credit, **make sure your program works with an arbitrarily large input** (assuming the number of unique words is reasonably small). Write your **program pseudocode** (or actual language code) here:

1) **Initialize something like a Dictionary or Map (Python or Java or C#)**

2) **Read in each word until there are none left (but don't retain the words after reading them)**

3) **If the word is not in the dictionary, initialize the count for that word to 1 (add a key-value pair of <word, 1> to the dictionary)**

4) **If the word is in the dictionary, add one to the count seen so far for that word**

5) **Once all words have been read in (but not stored, except as keys), print out the key (the word) and its value (the count) for every item in the dictionary (an "enhanced *for* loop" in Java)**

**In Java this can be done using a Map<String, Integer> (plus autoboxing and unboxing).**