Name: _____

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

CS 455 Final Exam
Spring 2015 [Bono]

May 13, 2015

There are 10 problems on the exam, with 73 points total available. There are 10 pages to the exam, including this one; make sure you have all of them. For on-campus students: if you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

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 Net ID at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	6 pts.	
Problem 2-5	12 pts.	
Problem 6	9 pts.	
Problem 7	5 pts.	
Problem 8	6 pts.	
Problem 9	20 pts.	
Problem 10	15 pts.	
TOTAL	73 pts.	

Problem 1 [6 points]

The following **Java** code attempts to use recursion to compute the maximum value in an array of numbers, but it has a bug:

```
// max: Computes the maximum value in array nums
// PRE: nums.length >= 1
public static int max(int[] nums) {
   return maxR(nums, 0);
}

// maxR (recursive helper function for max)
// Computes the maximum value in the sub-array of nums in positions startLoc
//   through nums.length-1
// PRE: nums.length >= 1
private static int maxR(int[] nums, int startLoc) {
   int maxSoFar = 0;
   if (startLoc == nums.length) {
      return maxSoFar;
   }
   if (nums[startLoc] > maxSoFar) {
      maxSoFar = nums[startLoc];
   }
   return maxR(nums, startLoc + 1);
}
```

Part A. Give a sample input array for `max` and the corresponding return value for max, such that the code above computes the *wrong* value for that array:

`nums`:

`max(nums)`:

Part B. Give a sample input array for `max` and the corresponding return value for max, such that the code above computes the *correct* value for that array:

`nums`:

`max(nums)`:

Problem 2 [2 points]

Not counting any eliminated in the first iteration, roughly how many elements are eliminated from further consideration in the **second** loop iteration of a binary search in an array of *n* elements (i.e., assuming we didn't find the value in that iteration)?

Problem 3 [2 points]

What is true about the first *k* elements in the array before pass *k* of insertion sort? (circle the answer that best describes what is known)

a. they are *k* values in sorted order

b. they are the *k* smallest values

c. they are the *k* smallest values in sorted order

d. none of the above

Problem 4 [4 points]

If you wanted to sort an array of `String`s in decreasing order by length using the **Java** `Arrays.sort` method you would also need what additional code (besides the call to the sort method)? (describe briefly– do not write code)

Problem 5 [4 points]

The **Java** `ListIterator` interface (used for iterating over a `LinkedList`) includes methods `hasNext()` and `next()`. The `Iterator` interface (used for iterating over a `Set`) also has methods `hasNext()` and `next()`.

In addition, the `ListIterator` interface also has the following method (where `ElmtType` is the type of an element in the `LinkedList`):

```
// Replaces the last element returned by next with the specified element
public void set(ElmtType newVal) . . .
```

Why is this method not provided in the `Iterator` interface?

Problem 6 [9 points]

Consider the following **Java** code to compute the number of occurrences of each score from an array of scores:

```
// Returns an array of the number of occurrences of each score from
//   scores array.
//   E.g., after we call it as follows:
//          int[] counts = getCounts(scores, 100);
//   counts[0] is the number of 0's in scores,
//   counts[1] is  the number of 1's in scores, . . . ,
//   counts[100] is the number of 100's in scores
// PRE: all the elements in scores are in the range [0,maxScore]
public static int[] getCounts(ArrayList<Integer> scores, int maxScore) {

   int[] counts = new int[maxScore+1];  // all values init'd to 0

   for (int i = 0; i < scores.size(); i++) {

      for (int score = 0; score <= maxScore; score++) {

         if (scores.get(i) == score) {

            counts[score]++;

         }

      }

   }
   return counts;
}
```

Part A [2]. What is the worst case big-O time for the **current** `getCounts` method in terms of `scores.size()` and `maxScore`?

Part B [5]. Modify the `getCounts` function to improve the big-O running time. You can make your modifications directly in the code above, using arrows to show where your new code should be inserted, crossing out code that you would get rid of, etc.

Part C [2]. What is the worst case big-O time for your **new version** of `getCounts`?

Problem 7 [5 points]

(**Java**) Consider a spell-checker that finds misspellings in a document, along with the number of times each misspelling occurred, and prints them out in alphabetical order.

Example input document:

```
hte bigest burger wase the best burger and hte wort bruger was teh
smallest bruger hte gril prefered the big burger
```

Example output:

```
bigest 1
bruger 2
gril 1
hte 3
prefered 1
teh 1
wase 1
wort 1
```

Write variable declaration(s) for storing the data to solve this problem, such that we can use this data structure to solve this problem efficiently; also explain your approach via the prompts below. You may use Java library classes. Note: The spell-checker will also have at its disposal a dictionary of English words stored in a `HashSet`.

Variable declaration(s):

Description of what problem data is stored in this data structure and how it's organized:

Description of how we are using the data structure to solve the problem (1-2 sentences maximum; do not write code):

Problem 8 [6 points]

Suppose we are creating a `Queue` class in **C++**. This `Queue` class will be for a queue of `int`s (i.e., it won't be a template class). Furthermore suppose we have a `main` program that uses the `Queue` class, and the whole program is in a single file, whose contents is shown here (some details left off so it all fits here). The different program elements are numbered at right for your convenience. (Note: unlike the later Queue problem, this one does not use any additional outside class or struct in its implementation.)

```
#include <iostream>                          (1)


using namespace std;                         (2)


class Queue {
 public:
  Queue();
  bool isEmpty() const;
  int front() const;
  void enQ(int val);                         (3)
  int deQ();

 private:
  . . . [instance variables here]
};


Queue::Queue() {
  . . .
}

bool Queue::isEmpty() const {
  . . .
}                                            (4)

int Queue::front() const {
  . . .
}

. . . [other Queue method definitions here]


int main() {
   Queue q;

   while (cin >> val) {
      q.enQ(val);
   }                                         (5)

   while (!q.isEmpty()) {
      cout << q.deQ() << " ";
   }
}
```

[problem continued next page]

Problem 8 (cont.)

In the space provided below show what would be in each file for a version of this program that can use separate compilation. That is, we want to be able to compile the `Queue` module separately from the program that uses queues. We're calling that client program `qProg.cpp`. Show everything that would need to go in each file, such that, for things that were in the original file, you can **use the identifying numbers from the previous page** (i.e., you do not need to rewrite the code from the previous page) to indicate the contents that those numbers refer to. For your convenience, we provided the `#ifndef` – stuff that goes in header files.

```
#ifndef QUEUE_H
#define QUEUE_H




#endif
```

Queue.h

Queue.cpp

qProg.cpp

Problem 9 [20 points]

Implement the static Java method `isPermutation` which returns true iff its array parameter `b` is a permutation of its array parameter `a`, where we assume `a` has no duplicate values. Your solution may use no additional arrays.

Examples below:

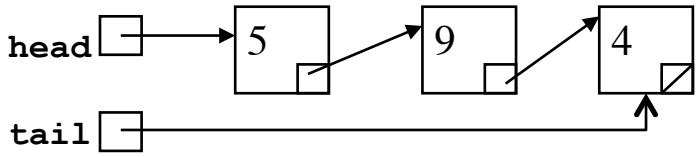
a	**b**	**isPermutation(a, b)**
[1, 2]	[3, 2, 1]	false
[1, 2]	[2, 1]	true
[1, 3, 5]	[4, 2]	false
[1, 3, 5]	[3, 2, 1]	false
[]	[]	true
[3]	[3]	true
[2, 5, 4, 7]	[7, 5, 2, 4]	true

```
// is array b a permutation of array a?
// (a and b are not changed by this method)
// PRE: a has no duplicate values
public static boolean isPermutation(int[] a, int[] b) {
```

Problem 10 [15 points]

Complete the implementation of the *C++* class `Queue`, below, which stores a queue of `int`s. You are required to use a linked list with head and tail pointer, described further here, as the representation.

A linked list with head and tail pointer means one where we store pointers to *both* the first element in the list (the "head") and the last element in the list (the "tail"). That means, instead of carrying around one variable for a linked list, we would need two variables. In this problem those two variables will be instance variables of the `Queue` class. Here's a diagram of a linked list with head and tail pointer that has three values in it:



With this representation, **your implementation must do all the operations in constant time.** In contrast, if we used a regular singly linked list instead, i.e., like the ones we've used in pa5 and in lab, not all the operations would be constant time. **Also, to get full credit, your code must have no memory leaks.**

Here is the class definition including instance variables. Complete the method definitions on the next page:

```
class Queue{

 public:
  Queue() // create an empty queue
  bool isEmpty() const;   // is the queue empty?
  int front() const;    // return front element in queue (queue is unchanged)
                        // pre: !isEmpty()
  void enQ(int val); // put a value on the queue
  int deQ();   // take a value off the queue and return it
                      // pre: !isEmpty()

 private:
   Node *head;
   Node *tail;
};
```

You may assume the following definitions have already been made, and you can use them:

```
struct Node {
  int data;
  Node * next;
  Node() { data = 0; next = NULL; }
  Node(int d) { data = d; next = NULL; }
  Node(int d, Node * n) { data = d; next = n; }
};
typedef Node * ListType;
```

[space for your answer on the next page]

Problem 9 (cont.)

```
// create an empty queue
Queue::Queue() {




// is the queue empty?
bool Queue::isEmpty() const {




// return front element in queue (queue is unchanged)
// pre: !isEmpty()
int Queue::front() const {




// put a value on the queue
void Queue::enQ(int val) {




// take a value off the queue and return it
// pre: !isEmpty()
int Queue::deQ() {
```