# Prelim 2, CS2110. SOLUTION

7:30 PM, 25 April 2017

## 1. Name (1 point)

Write your name and NetID at the top of **every** page of this exam.

## 2. Short Answer (26 points.)

**(a) Asymptotic complexity. 8 points.** Be sure to answer both (1) and (2) of this question.

(1) Two correct algorithms $B1$ and $B2$ have running times given by the functions $g(n)$ and $f(n)$ respectively. Assume $f(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$. Could there be a reason to choose one algorithm over another in a practical setting? Answer YES or NO and give a short reason for your answer.

YES. Here are two possible answers. Space complexity could be different. The constants involved could differ tremendously.

(2) Give the tightest asymptotic complexity in terms of $n$ (e.g. $O(n^3)$) of a call p(n) on the following procedure. Explain your answer. Note: the procedure doesn't have a result; it just takes time.

```
public static void p(int n) {
   if (n == 0) return;
   int s= 0;
   for (int k= 1; k <= Math.min(100, n); k= k + 1) {
       s= s + 10;
   }
   p(n-1);
}
```

Answer. $O(n)$. The loop performs at most 100 iterations each time it's executed; that's constant time, $O(1)$. The number of calls on p is $n + 1$.

**(b) Hashing. 5 points.** We are implementing a set in an array $b$ using chaining. The set currently has size $n$. Give the tightest worst-case complexity formula for enumerating all elements of the set. Do not use the load factor, which for the purpose of this question is unknown.

Hint: think about what has to be done to find all the elements in the set.

Answer. Suppose the set contains $n$ elements. Each element of array $b$ has to be tested. That's $b.length$ tests. For each element that is a linked list, the items in the linked list have to be enumerated. In total, that's the size of the set. So it is $O(b.length + n)$.

**(c) Hashing. 7 points.** An instance of class Team below maintains information about teams of two tennis players. We include only information that is necessary for this question. Class Player, among other things, has its own functions *equals* and *hashCode*.

We expect some program written by people managing a tennis tournament to use a hash set of Team objects to maintain information about teams, so class Team needs functions *equals* and *hashCode*. Complete these functions below. Notes: (1) The order of the Players in a Team object shouldn't matter. (2) Choose a reasonable simple *hashCode* based on the *hashCodes* of the players.

```
/* An instance maintains info about a team of 2 players. **/
public class Team {
    Player p1;  // p1 and p2 are different Players
    Player p2;  // according to function equals in class Player.

    /** Return true iff t is a Team whose players are equal to
         those in this object. */
    public @Override boolean equals(Object t) {
         if (!(t instanceof Team)) return false;
         Team t1= (Team) t;
         return (p1.equals(t1.p1) && p2.equals(t1.p2)) ||
                (p1.equals(t1.p2) && p2.equals(t1.p1));    }

    public @Override int hashCode() {
       return p1.hashCode() + p2.hashCode();
    }
}
```

**(d) 6 points** Write the steps involved in evaluating a new expression **new** Player(b, c).

1. Create a new object of class Player, with default values for its fields.

2. Execute the constructor call **new** Player(b, c).

3. Yield as value of the expression the name of (i.e. pointer to) the new object.

## 3. Sorting (18 points.)

**(a) 8 points** A stable sorting algorithm maintains the relative order of equal values. For example, suppose a 3-element array contains [**3**, 2, 3]. A sorting algorithm that changes it to [2, 3, **3**] is not stable because it switched the order of the two 3's. To be stable, it would have to change the array to [2, **3**, 3]

For each of the following sorting algorithms, tell us whether (1) it is stable, (2) it is unstable, or (3) it depends on the implementation.

1. selection sort Answer: unstable
2. insertion sort Answer: stable
3. mergesort Answer: it depends on the implementation. When merging two adjacent sorted segments, and there are two equal values to move, the choice of which one to move could make it stable or unstable.
4. quicksort Answer: unstable

**(b) 6 points** Procedure Quicksort (called QS below) is given below. It contains errors. Fix the errors. Assume that method partition has the specification shown after procedure QS.

```
/** Sort b[h..k]. */
public static void QS(Comparable[] b, int h, int k) {
    if (h >= k) return;     // fixed the if-condition
    j= partition(b, h, k);  // Moved this from bottom to here
    QS(b, h, j-1);
    QS(b, j+1, k);
}


/** b[h] contains some value x. Swap values of b[h..k] so that
    b[h..j-1] <= b[j] = x <= b[j+1..k] and return j .
    Note: in the line above, <=  and = are tested using compareTo. */
public static int partition(Comparable[] b, int h, int k) {...}
```

**(c) 4 points** Give the following information about the time- and space-complexity of MergeSort to sort an array of size $n$.

1. MergeSort, worst-case time: $O(n\ log\ n)$
2. MergeSort, expected time: $O(n\ log\ n)$
3. MergeSort, worst-case space: $O(n)$
4. MergeSort, expected-case space: $O(n)$

## 4. Collections classes/interfaces (15 points.)

This question concerns interface Set<C>, which, among others, has these methods:

- boolean contains(C ob): Return true iff this set contains ob.
- Iterator<C> iterator(): Return an iterator over the elements in this set.
- int size(): Return the number of elements in this set.
- C[] toArray(): Return an array containing all the elements in this set.

For this question, consider a class Student, whose objects are Cornell students.

**(a) 5 points.** Interface Set<C> lacks methods to extract elements from the set, but a Set<C> is iterable. Write the body of function pickAStudent below —it will probably use a for-each loop.

```
/** Return any element of s. Precondition: s is not empty. */
public static Student pickAStudent(Set<Student> s) {
    for (Student c: s) return c;
    return null; // Note: this is needed for syntax-compilation.
                 // But if students leave it out, thats OK.
}
```

**(b) 5 points.** Is the element returned by (a) guaranteed to be a randomly selected element? BRIEFLY justify your answer. Hint: Remember that Set<C> is an interface.

Answer. In some implementations of Set<C>, an iterator enumerates the items in a fixed order. For example, a priority queue (such as a min-heap) will return values in increasing order. Thus the implementation shown above would always return the student considered to have minimum value within the deck, which isn't random.

**(c) 5 points.** Assume that a variable $r$ of type *Random* is available and that $r$ was initialized correctly. Variable $r$ has an instance method $nextInt(int\ max)$, and each call $r.nextInt(m)$ returns a new random integer in the range $0..m$.

Write a one-line implementation of pickAStudent given below. Hint: look at the methods available in interface Set<C>. You don't have to use a for-each loop (but you can)! We give partial credit for solutions that require several lines of code, but for full credit, your solution must (1) be a single return statement, (2) be correct, and (3) if called often enough, would return every card in the deck at least once. Your solution need not be time-efficient.

```
/** Return a random element of s. Precondition: s is not empty. */
public static Student pickAStudent(Set<Student> s) {
     return s.toArray()[r.nextInt(s.size() - 1)];
 }
```

## 5. Trees (20 points.)

**(a) 10 points** Consider the following class Node. Complete instance function isBST.

```
/** An instance is a node of a tree (or a subtree rooted at that node). */
public class Node {
   public int val;   // the value in this node
   public Node left; // left subtree (or null if none)
   public Node rite; // right subtree (or null if none)
   ...

  /** Return true if the tree rooted at this node satisfies the properties of
       a binary search tree (BST) and all of its values are in the range h..k. */
   public boolean isBST(int h, int k) {
      if (val < h || k < val) return false;
      return (left == null || left.isBST(h, val-1)) &&
             (rite == null || rite.isBST(val+1, k));
     }
}
```

**(b) 5 points** A heap of integers is maintained in an int array b. Suppose the heap has size 9 and b[0..8] contains these 9 integers:

```
[1  4  6  5  6  9  8  8  7]
```

Below, show what b[0..9] is like after adding value 3 to the heap. We cannot give partial credit if you don't show your work.

Draw the heap as a tree, add the value 3 to the heap, bubbling up as necessary, and write down the answer:

```
[1  3  6  5  4  9  8  8  7  6]
```

**(c) 5 points** It has been said that a binary tree with no duplicate values in its nodes can be reconstructed from its inorder and postorder traversals. Write down the first two steps in doing this —how do you tell (1) what the root of the tree is and (2) what is in its left and right subtrees? Do not give us an example. Just tell us in English what these two steps are.

The last element of the postorder traversal is the root of the tree. Find that element in the inorder traversal; everything to its left (right) is in its left (right) subtree.

## 6. Graphs (20 points.)

**(a) 8 points** Complete the body of the following procedure. Keep things abstract: to visit a node p write simply "visit p" and to ask whether p has been visited write "if (p has been visited)" or "if (p has not been visited)". Similarly, you can write about the neighbors of a node in an abstract way, as we have done in lecture.

```
/** Visit all nodes reachable from node p along paths of unvisited nodes.
    Precondition: p has not been visited. */
public void DFS(Node p) {
    Visit p;
    for each neighbor w of p:
        if (w is unvisited) DFS(w);
}
```

**(b) 4 points** State the theorem that is proved in our development of Dijkstra's shortest path algorithm. It talks about a node that can be moved to the Settled set.

Answer. For a node f in the Frontier set with minimum d-value over nodes in the Frontier set, d[f] is indeed the length of the shortest path from the start node to f.

**(c) 4 points** You know that an undirected graph $G$ is *bipartite* if its vertices can be partitioned into two sets such that no edge connects two vertices in the same set. Below are two other equivalent definitions of a bipartite graph, except that they have mistakes. Fix the mistakes.

- $G$ is bipartite Definition given above; it's correct.
- $G$ is 3-colorable. Should be: $G$ is 2-colorable.
- $G$ has no cycle of even length Should be: $G$ has no cycle of odd length.