CS2110 Final Exam SOLUTION

10 December 2018, 9:00AM–9:30PM

	0	2	3	4	5	6	7	Total
Question	Name	Short Answer	Recursion	Hashing	Object Oriented	Data Structures	Graphs	
Max	0	34	7	9	19	18	13	100
Score								

The exam is closed book and closed notes. Do not begin until instructed.

You have **150 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of **every** page! There are 6 questions on 10 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your pages are still stapled together. If not, please use our stapler to reattach all your pages!

We have scrap paper available. If you do a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam so that we can make sense of what you handed in.

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space provided.

In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken the final.

_____

(signature)

1. Name (0 points)

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

2. Short Answer (34 points)

(a) True / False (10 points) **Circle** T or F in the table below.

(a)	T	F	The try-catch block can be used to handle an `Exception` but not an `Error`. False.
(b)	T	F	A for-loop with a control variable that ranges over the numbers 1..n takes expected time $O(n)$. False. It depends on what the body of the loop does.
(c)	T	F	Suppose class `C` implements interface `I1`. Then a subclass of `C` cannot implement `I1`. False.
(d)	T	F	To create and start a new thread, (1) have a class C that implements interface *Runnable*, (2) in C implement method *run*(), and (3) create an instance M of class C. False. You also have to call $M.start()$.
(e)	T	F	A graph is bipartite if it is 2-colorable. True.
(f)	T	F	There can be two shortest paths between two nodes of a graph. True. Suppose all weights are 1. Let edges be (A, B), (B, D), (A, C), and (C, D). The 2 paths (A, B, D) and (A, C, D) both have distance 2.
(g)	T	F	If a graph is drawn so that 2 edges cross, it is non-planar. False.
(h)	T	F	Execution of `System.out.println('a' + 1);` prints the character $'b'$. False. it prints the integer 98.
(i)	T	F	To resize the array used in implementing hashing using linear probing, double its size and copy the old array into the beginning of the new one. False. Every element in the array has to be rehashed.
(j)	T	F	A recursive method must return a value. False.

(b) Proof that $f(n)$ is $O(g(n))$ (3 points) Prove that $8n^3 + 2n^4$ is $O(10 + n^4)$.

$8n^3 + 2n^4$
$\leq$ <Assume $1 \leq n$, so $n^3 \leq n^4$ >
$8n^4 + 2n^4$
$=$ <Arithmetic>
$10n^4$
$\leq$ <Arithmetic>
$10n^4 + 100$
$=$ <Arithmetic>
$10(n^4 + 10)$ So choose $c = 10$ and $N = 1$

(c) Complexity (2 points) State the time complexities:

(i) Worst-case insertion in the beginning of a linked list of size n: O(1)

(ii) Worst-case insertion in an `ArrayList` of size n: O(n)

(iii) Worst case insertion time in a heap of size n: O(log n)

(iv) Worst-case search for a value in a `HashSet` of size n: O(n)

(d) Concurrency (7 points) We show part of class `BankAccount`, which was written for a program that supports only *one thread*:

```
public class BankAccount {
    private String id;          // 8-digit account identifier
    private double balance;     // current account balance

    /** Return the balance of the bank account */
    public double getBalance()  { return balance;  }

    /** If the balance is less than amount, return false.
      * Otherwise, withdraw amount and return true. */
    public boolean withdraw(double amount)  {
        if (balance < amount) return false;
        balance= balance - amount;
        return true;
    }
}
```

Now suppose this code is to be used without modification in a program with multiple threads.
(1) If multiple threads call `withdraw()` at the same time, could this produce unexpected behavior? If not, explain why. If so, describe a scenario and its result.

Yes, a race condition can occur. 2 threads could execute `balance= balance - amount;` at the same time, creating the same issue as shown in class with 2 threads executing `x= x + 1;` concurrently.

(2) If multiple threads call `getBalance()` at the same time, could this produce unexpected behvaior? If not, explain why. If so, describe a scenario and its result.
No, there is no problem with 2 threads reading a shared memory location at the same time.

(e) Sorting times (4 points)

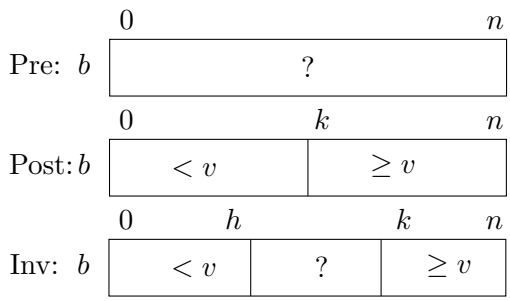
(i) Consider this array of n elements: {1, 2, 3, 4, ..., n}. Circle the sorting algorithm that will perform best on this above input: insertionsort, selectionsort, quicksort, mergesort. What would be its time complexity on this array? insertion sort is best. $O(n)$

(ii) When will quicksort take its worst-case run time and why? What is the worst-case run time? If the input is sorted, partitioning using the first value as pivot always gives two partitions of sizes 0 and $n-1$ to sort. The runtime is $O(n^2)$.

(iii) Describe an array when insertion sort has its worst-case runtime. The array is in descending order.

(iv) Is there a case where selection sort takes less than $O(n^2)$ time? no.

(f) Binary search (5 points)
To the right are the precondition, postcondition, and invariant for a binary search, looking for v in $b[0..n]$. We assume with stating it anywhere that array b is sorted (in ascending order). Below, complete the algorithm.




```
h= -1; k= n+1;
while ( h+1 < k                                  ) {
   int e= (h+k)/2;
   if (b[e] < v) h= e;
   else k= e;
 }
```

(g) New-expression (3 points) Write the steps in evaluating the new-expression $C(5, 3)$.
1. Create (draw) an object of class C. 2. Execute the constructor call $C(5, 3)$. 3. Yield as value of the new-expression the name of (pointer to) the newly created object.

3. Recursion (7 points)

Consider the following class `Person`. We give only the fields in the class and the specification of method `wasBornIn`. Note carefully the part in the class invariant about the relation between the child's and parent's birth years.

Complete the body of method `wasBornIn`. *A portion of the grade will depend on using the class invariant to cut the search short where possible.*

```
public class Person {
   int yrBorn;   // The year this person was born.
                 // It is greater than the years its known parents were born
   Person par1;  // one parent, null if unknown
   Person par2;  // another parent, null if unknown

   /** Return true iff someone in the tree with this as root was born in year y. */
   public boolean wasBornIn(int y) {
       if (y == yrBorn) return true;
       if (y < yrBorn) return false;
       if (par1 != null  &&  par1.wasBornIn(y)) return true;
       return par2 != null  &&  par2.wasBornIn(y);
   }
}
```

4. Hashing, Iterator, and Iterable (9 points)

Below is part of class `HshSet<E>` for hashing using linear probing. It is iterable. The constructor of inner class `SetIterator` is complete. Yes, its body is empty.

(a) (7 points) Complete method `iterator()` and methods `hasNext()` and `next()` of inner class `SetIterator`. Read carefully the class invariant of `SetIterator` and the comment in method `hasNext()`.

(b) (2 points) Do any changes have to be made in the methods you wrote if we decide to use quadratic probing instead of linear probing? No. All array elements must still be checked and enumerated if they are not null.

```
public class HshSet<E> implements Iterable<E> {
   private E[] b;  // Each b[i] is either null or a non-null value of type E;
                   // in the latter case, that value is in the set.

   /** = an Iterator over this hashed set. */
   public Iterator<E> iterator() {
       return new SetIterator();
   }

  /** An iterator over the elements of the hashed set. */
  private class SetIterator implements Iterator<E> {
       // 0 <= k <= b.length. Elements in b[0..k-1] have been enumerated
       // If k < b.length, b[k] may be null or an element of the set
       private int k;

       /** Constructor: an iterator over the set. */
       public SetIterator() {}

       @Override public boolean hasNext() {
          // Note: When hasNext() returns true, b[k] IS next element to enumerate
          while (k < b.length  &&  b[k] == null) k= k+1;
          return k < b.length;
       }

       @Override public E next() {
          if (!hasNext()) throw new NoSuchElementException();
          k= k+1;
          return b[k-1];
       }
  }
}
```

5. Object-Oriented Programming (19 points)

To the right are two interfaces and an abstract class that represent universities, private universities, and universities in the NYS SUNY system. Below is a class that represents Cornell.

```
/** A part-private and part-public
  * university far above ... */
public class Cornell extends SUNY
           implements PrivateUniv {
  String pName; // president's name

  /** Constructor: Cornell with
   * initial state funding and
   * president pName */
  public Cornell(String pName) {

     super();
     this.pName= pName;

   }

  /** Constructor: Cornell with
   * initial state funding,
   * private funding pf
   * and president pName */
  public Cornell(String pName,
                       int pf) {
     this(pName);
     if (funding < MAX_PRIV_DON) {
         funding= funding + pf;
     }
  }

  /** Graduate Cornell students */
  public void graduateStudents() {
     // Consider this implemented
  }
}
```

```
/** A place of higher learning that
  * graduates students */
public interface Univ {
  /** Graduate the current class. */
  public void graduateStudents();
}

/** Privately funded university */
public interface PrivateUniv extends Univ {
  /** IRS-imposed max for private donations*/
  final int MAX_PRIV_DON= 100000;
}

/** A public university in the SUNY system
  * funded by New York tax payers */
public abstract class SUNY implements Univ {
  /** New York State tax dollars */
  public static final int STATE_FUNDING= 1000000;

  /** Current funding total for the year */
  private int funding;

  /** Constructor: a SUNY univ funded by NYS */
  public SUNY() {
    funding= STATE_FUNDING;
  }

  /** return the funding total */
  public int getFunding() { return funding; }

  /** Set the funding total to t*/
  public void setFunding(int t) { funding= t; }
}
```

(a) 3 points Does the Cornell constructor with two parameters compile? If not, how would you change it to compile? It doesn't compile because field funding is private. To change it, use the setter and getter methods for funding.

(b) 3 points Which methods does class `SUNY` need to implement in order to compile, if any? If none, explain why in one sentence. None, since it is abstract.

(c) 4 points Above, complete the constructor `Cornell(String pName)` according to its specification.

(d) 3 points The IRS has imposed an extremely large tax penalty for all private donations exceeding `MAX_PRIV_DON` in `PrivateUniv`. Hoping to circumvent this penalty and take advantage of its affluent alumni base, Cornell would like to assign a larger value to `MAX_PRIV_DON` in its second constructor to allow for larger donations. Is this possible? If not, explain why in one sentence. No, final fields cannot be modified. Note: `final` in interface fields is optional.

(e) 3 points The NYS Education Dept. decided that each university should be in charge of its own admission process and created a method `admitStudents()` in class `SUNY`. How can it ensure that all `SUNY` universities implement `admitStudents()`? Make `admitStudents()` abstract.

(f) 3 points Cornell has mandated that its students pass a rudimentary swim test before graduating and have modified method `public void graduateStudents()` to `public void graduateStudents( boolean swimPassed)`. Is this code valid Java code? If not, explain in a sentence. No, Cornell must declare method `graduateStudents()` with no parameters, since it is declared that way in class `Univ`.

6. Data Structures (18 points)

(a) Queues (5 points)

To the right below are the fields of class `ArrayQueue`, which implements a queue of bounded size —the size of array `b`. Notice how queue elements wrap around in the array: if a queue element is in the last array element `b[b.length-1]`, the next one is in `b[0]`. We discussed this implementation in lecture.

Below, implement methods `peek` and `put`.

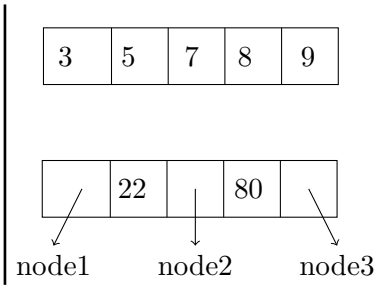
```
/** Return first element of the queue.
  * Precondition: queue is not empty. */
public int peek() {
    return b[h];
 }

/** Add e to the queue..
  * Precondition. queue is not full */
public void put(int e) {
    b[(h+n) % b.length]= e;
    n= n+1;
 }
```

```
public class ArrayQueue {
  // The n elements of the queue are in
  // b[h], b[h+1], ... b[h+n-1] (all
  // indexes mod b.length)
  // 0 <= h < b.length
  private int[] b;
  private int n;
  private int h;
}
```

(b) Printing a B-Tree (9 points)

The Binary Search Tree is difficult to keep balanced, In the 1970's other data structures were developed to overcome this difficulty. Hopcroft of Cornell CS came up with a 2-3 tree. But the data structure that received the most attention was Bayer and McCreight's B-tree. B-trees are new to you. This question explores some aspects of B-trees, using B-trees containing sets of integers, called *keys*.

To the right is an example of a leaf of a B-tree: a leaf contains an ascending sequence of keys, in this case, the integers 3, 5, 7, 8, 9.



Underneath the leaf we show an internal node of a B-tree. An internal node consists of a sequence of at least two children, each a BTree node, and between each pair of children is a key.

Here are the important properties of all internal nodes (non-leaves):

- The keys (in this case, integers) are in ascending order.
- The keys in the subtree before a key are less than the key. For example, all keys in subtree node1 are less than 22 and all keys in subtree node2 are less than 80.
- The keys in the subtree after a key are greater than the key. For example, all keys in subtree node2 are greater than 22 and all keys in subtree node3 are greater than 80.

Below is part of class `BTree`, showing the fields that contain the keys and the children, both as `ArrayLists`. Also shown is method `print`. Write the body of method `print`. Assume that method `printArray(ArrayList b)` prints the values in `b`, and use the standard `println` statement to print a single integer.

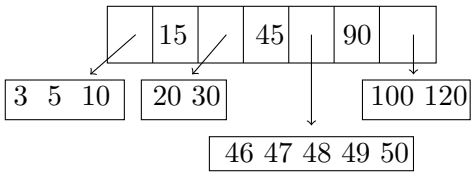
```
public class BTreeNode {
   private ArrayList<Integer> keys;    // These are the integers
   private ArrayList<BTreeNode> children;  // null only if this is a leaf

   /** Print the values of this Btree in ascending order. */
   public void print() {
      if (children == null) { printArray(keys); return; }

      children.get(0).print();
      for (int k= 0; k < keys.size(); k= k+1) {
         System.out.println(keys.get(k));
         children.get(k+1).print();
      }
   }
```

(c) Deleting an element from a B-tree (4 points)

We look briefly at one aspect of maintaining a B-tree: deleting a value in a simple case.

The complete definition of a B-tree involves an integer t. Here are three further properties of B-trees:

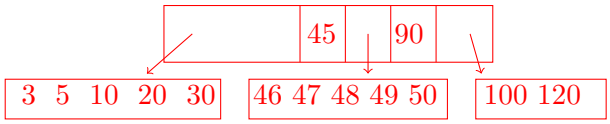
- Every node except the root contains at least $t - 1$ keys.
- Every node has at most $2t - 1$ keys.
- All leaves are at the same level.



For example, the tree to the right has $t = 3$: each node has a least 2 keys and at most 5 keys. Further, the four leaves are at the same level.

Below, draw the B-tree that would result from deleting the value 15. It should still be a tree with $t = 3$, and as little change should be made to the B-tree as possible.

Deleting the 15 requires eliminating one of the children. But together, the two left children have a total of 5 keys, so they can be combined into a single child, giving the B-tree shown to the right,



7. Graphs (13 points)

(a) (2 points)

If all edge weights on a graph are 1, to what algorithm is Dijkstra's shortest-path algorithm equivalent? BFS —Breadth-first search.

(b) (3 points)

Will Dijkstra's shortest-path algorithm work if the edge weights are negative? If no, explain why briefly. No. Here's 1 reason. The theorem we proved about choosing a node from F with minimum d-value cannot be proved. Another reason. If there is a cycle, a shortest path doesn't exist because going around the cycle again reduces the "distance".

(c) (2 points)

Kruskal's algorithm for constructing a spanning tree starts with all nodes and no edges. It repeatedly adds a minimum-weight edge that does not form a cycle until no more edges can be added. What does Prim's algorithm do? Same thing, except that the added edge must be connected to all the other previously added edges.

(d) (3 points)

You have a directed graph of all airplane flights in the U.S. Thus, there is an edge from city `C1` to city `C2` if there is a flight from `C1` to `C2`. Of the several graph algorithms we have discussed, which one is best suited to find a flight from Ithaca to Kalamazoo with as few stops as possible? BFS —Breadth-first search. Use it to find all cities 1 stop from Ithaca, then all cities 2 stops from Ithaca, etc.

(e) (3 points)

How many edges does a complete undirected graph of n nodes have? Describe or draw the smallest non-planar complete graph. A complete graph has as many edges as possible: $n(n-1)/2$ edges. Every graph of fewer than 5 nodes is planar. The complete graph K_5 has 5 nodes and 10 edges. It is not planar. We don't draw it here.