Name: _____

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

Midterm Exam 2
CS 455, Fall 2014

November 11, 2014

There are 8 problems on the exam, with 62 points total available. There are 8 pages to the exam, including this one; make sure you have all of them. There is also a separate one-page double-sided code handout. 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).

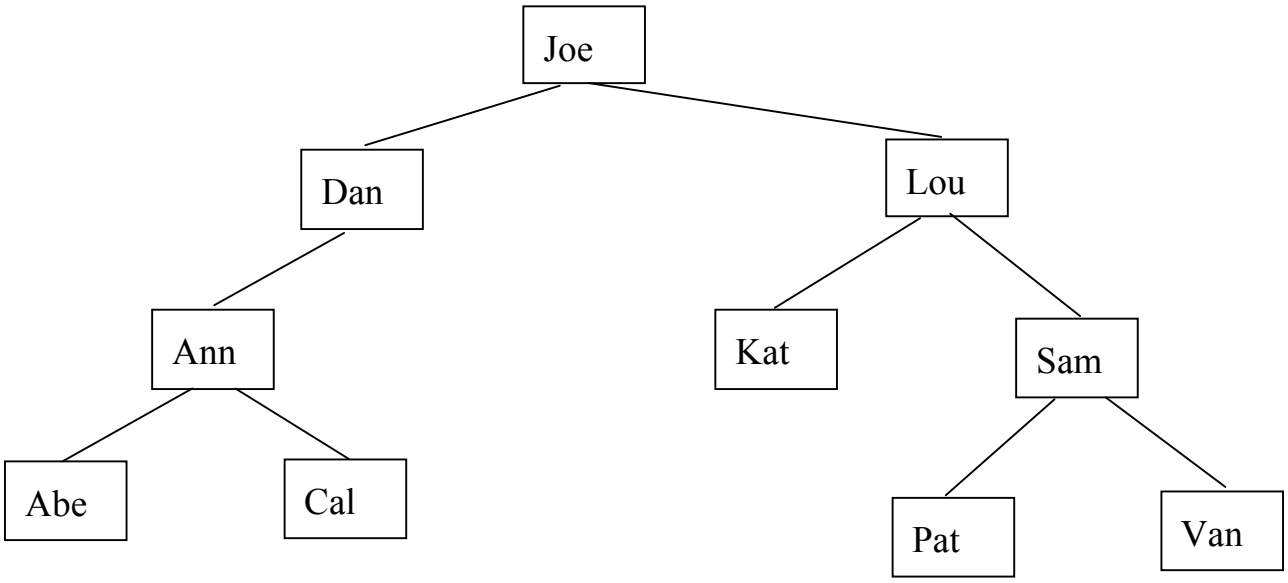
Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and NetId at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	6 pts.	
Problem 2 & 3	8 pts.	
Problem 4	10 pts.	
Problem 5	5 pts.	
Problem 6	10 pts.	
Problem 7	8 pts.	
Problem 8	15 pts.	
TOTAL	62 pts.	

Problem 1 [6 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 *Ned*

Part B. lookup *Pat*

Part C. lookup *Joe*

Problem 2 [2 pts.]

Give the big-O worst case time for checking if an array of size n is already sorted.

Problem 3 [6 pts.]

In the lecture on algorithm analysis, we discussed the big-O of doing each of the following methods on the `Sequence` class below, assuming we used an array to represent the sequence. Here, **give the big-O worst case time for each of the following methods, assuming the class uses a `LinkedList` representation internally.** Write your answer to the left of each method (we did the first one for you).

```
      // a Sequence of integers
      // precondition for methods that take a loc parameter:
      //   0 <= loc < numVals()
      public class Sequence {
         LinkedList<Integer> seqList;

O(1)     Sequence() { . . . }      // creates an empty sequence

         int getValAt(int loc) { . . . }
                                   // get the value at position loc in the sequence

         boolean contains(int val) { . . . }
                                      // does the sequence contain the value val?

         void removeValAt(int loc) { . . . }  // remove the value at position loc

         void insertAtEnd(int val) { . . . }
                                           // insert val at the end of the sequence

         void insertInFront(int val) { . . . }
                                     // insert val at the beginning of the sequence

         int numVals() { . . . }  // number of values in the sequence

      }
```

Problem 4 [10 pts.]

Consider the following recursive function for doing merge sort:

```
// Sorts the values in array a in increasing order
public static void mergesort(int[] a)
   if (a.length > 1) {
      int[] first = copyArr(a, 0, a.length / 2);
      int[] second = copyArr(a, a.length / 2, a.length);
      mergesort(first);
      mergesort(second);
      // *
      merge(first, second, a);
   }
}
```

Specification of the helper methods used above:

```
// Returns a copy of a piece of array orig.
// copies orig[startLoc] through orig[end-1] into a new array, which is returned.
// the length of the new array is (end - startLoc)
// pre: (0 <= startLoc < end) and (end <= orig.length)
private static int[] copyArr(int[] orig, int startLoc, int end) { … }

// Merges two sorted arrays, first and second, into an array, a
// pre: first.length + second.length <= a.length
private static void merge(int[] first, int[] second, int[] a) { … }
```

Suppose we call `mergesort` with the following array:

0	1	2	3	4	5	6	7	8	9
17	12	2	7	20	36	5	4	18	15

(top level shows array indices)

For the top-level call to `mergesort` on this data, show the contents of `first` and `second` at the point in the code marked above with an asterisk (*):

contents of array `first` *at point marked with **:*

contents of array `second` *at point marked with **:*

Problem 5 [5 points]

Suppose we have the `Term` class from assignment 2. This class overrode *neither* `equals` *nor* `hashCode`. However, we can still make a `HashSet` or `HashMap` using `Term`s, because `hashCode` and `equals` are inherited from `Object`. Consider the following code using `Term`:

```
Set<Term> termSet = new HashSet<Term>();
Term t = new Term(3, 4);
termSet.add(t);
Term target = new Term(3, 4);
```

For each of the following boolean expressions using the variables above, state whether it would come out true or false:

1. `t.equals(target)`
2. `t.hashCode() == target.hashCode()`
3. `t == target`
4. `termSet.contains(t)`
5. `termSet.contains(target)`

Problem 6 [10 points]

Consider the following function that uses the Java Queue interface:

```
public static void queueProb() {
   Queue<Integer> q = new LinkedList<Integer>();
   for (int i = 1; i <= 5; i++) {
      q.add(i);
      q.add(i * 2);
      System.out.print(q.remove() + " ");
   }
   System.out.println();
   // *
}
```

Part A. What is the output of the function?

Part B. Show the contents of the queue when the execution gets to the point labeled with an asterisk (i.e., contents after the loop). Use the space below, showing the front of the queue on the left (i.e., queue growing from left to right):

front →

Problem 7 [8 points]

Consider the following static method, `printSatisfying` that prints out all of the elements of a `LinkedList` of `Integer`s that satisfy a certain condition. The condition to be satisfied is a callback method `condition`, of the interface `ConditionTester`, also shown below (Note: this is analogous to the similar interface called `Predicate`, used in a recent lab).

```
// prints out the elements of list that satisfy the condition given in cond.
// The elements will all be on one line, space separated, followed by a newline.
public static void printSatisfying(LinkedList<Integer> list, ConditionTester cond)


interface ConditionTester {
   // test whether num satisfies some condition
   boolean condition(Integer num);
}
```

Write the method `printDivBy3`, which prints out all of the elements of the given list that are divisible by 3. Your solution must use the method `printSatisfying`, whose interface is given above, and include any additional code necessary to make it work. (You may assume `printSatisfying` and `printDivBy3` are part of the same class, and `printSatisfying` has already been written.)

Examples:

list	*output of call to* ***printDivBy3(list)***
(4 5)	<only prints a newline>
()	<only prints a newline>
(3 4 12 6 5 0)	3 12 6 0
(21)	21

```
// pre: the elements of list are non-negative
public static void printDivBy3(LinkedList<Integer> list) {
```

Problem 8 [15 points]

Write the function `copy`, which copies an `ArrayList` of `Integer`s into a `LinkedList` recursively.

Examples (`ArrayList` shown as comma-separated with brackets; `LinkedList` shown as space-separated with parentheses):

orig	*return value of call to* ***copy(orig)***
`[4,5]`	`(4 5)`
`[]`	`()`
`[3,4,3,3]`	`(3 4 3 3)`
`[2]`	`(2)`

Hint: you will need a helper method to do the actual recursion. There's more than one possible header for a helper method for `copy`, but one option is a method that has an extra parameter for an array index. The index will indicate the part of the `ArrayList` that should be copied (i.e. the part that starts at that index).

Little or no credit will be given for a solution that does not use recursion.

```
public static LinkedList<Integer> copy(ArrayList<Integer> orig) {
```