Name: _____

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

CS 455 Midterm Exam 2
Spring 2016 [Bono]

Apr. 5, 2016

There are 7 problems on the exam, with 57 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. There is also a separate one-page double-sided code handout. If you need additional space to write any answers, pages 9 and 10 are left blank for that purpose. If you use those pages for answers you just need to direct us to look there. *Do not detach any pages from this exam.*

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 username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [4 points]

The contract that a class's `equals` and `hashCode` methods have to satisfy for a class to be used as the key type for a `HashMap` is:

if _____ then _____

Suppose `k1` and `k2` are two objects of the key type for a `HashMap`. Complete the above statement by filling in the blanks above, choosing from the following numbers 1 through 6.

(1) `k1.hashCode() == k2.hashCode()`

(2) `k1.hashCode() != k2.hashCode()`

(3) `k1.hashCode() < k2.hashCode()`

(4) `k1.equals(k2)`

(5) `!k1.equals(k2)`

(6) `k1.compareTo(k2) < 0`

Problem 2 [4 points]

Consider a plain old binary search tree (that does not maintain the balance property). Recall the way an insert is done into such a tree is to search for the element (using binary search on the tree), and then insert the value at the spot in the tree where the search failed.

Suppose we build such a tree out of ***n*** elements that are already in order. (E.g., start from an empty tree, and then insert "Ann", then "Bob", then "Cat", then "Dan", then "Ed", then "Fred", then "Gary", then "Hal", etc.)

Part A. What's the height of this tree as a function of *n*? (The height is the longest path from the root to a leaf, counting path length as the number of nodes on the path.)

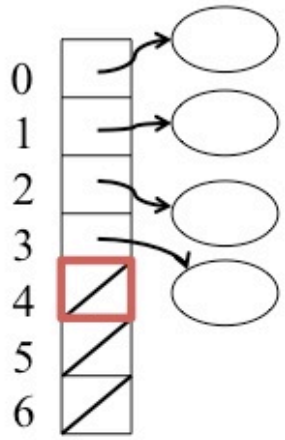
Part B. What's the total worst case big-O time for building this whole tree as a function of *n*?

Problem 3 [2 points]

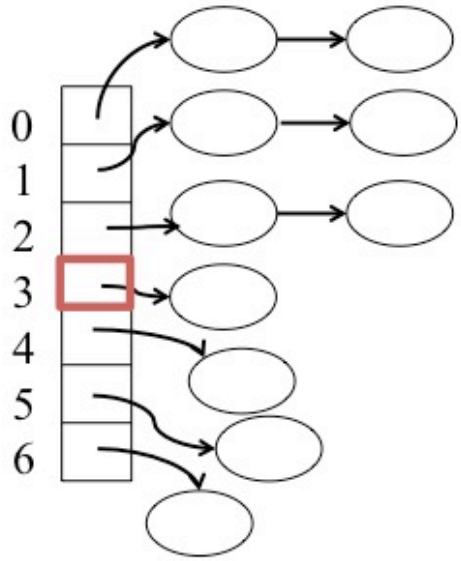
Name the Java Collection class we have discussed this semester that most closely matches the functionality of the `Names` class we discussed earlier in the semester. The `Names` class stores a bunch of `Strings` with no duplicates and has methods: insert(name), remove(name), lookup(name), numNames() and printNames() (the last one prints all the names in alphabetical order).

Problem 4 [5 points]

Consider a hash table scheme where we fill the table (i.e., array) up starting from the beginning (like a partially filled array), and then only add a second element to a chain once all the hash buckets all have one element. This way, entries will get evenly distributed throughout the table. To make this fast, the instance variable `nextSmallLoc` will keep track of the next location to use – you may assume it gets initialized to zero when we first create the table; it gets updated in the hash function below. Assume `HASH_SIZE` is the size of the hash table array. Here are some examples of hash table contents, and the corresponding value for `nextSmallLoc`.



`nextSmallLoc = 4`
`HASH_SIZE = 7`



`nextSmallLoc = 3`

Here's the hash function for this scheme:

```
public int hash(String key) {
   int hashValue = nextSmallLoc;
   nextSmallLoc = (nextSmallLoc + 1) % HASH_SIZE;
                                  // next time we'll choose the next hash bucket
   return hashValue;
}
```

Part A [1]. Suppose the current value of `nextSmallLoc` is `(HASH_SIZE - 1)`. What value will `nextSmallLoc` have after the next call to `hash`?

Part B [4]. The above would not make a good hash function. Why not? (Be specific about what the problem is.)

Problem 5 [10 points]

Consider the following partially written method, `searchFrom`, to do a recursive maze search as we did in assignment 3. This is a recursive helper method for the public search method of the `Maze` class. Note that `searchFrom` *returns* the path found (it does not store it in an instance variable) and if no path can be found from the given location it returns `null`.

Complete the code so it creates and returns the path correctly; i.e., after your code additions it will do what the comment says it does. (Note: the partial code is on the next page so you have sufficient space to write your answer.)

Some additional information about the problem and code given:

- the code uses additional helper methods which should be self-explanatory (they handle accessing the maze data and visited data). You may assume they are already implemented for you.
- Reminder: `MazeCoord` is a class used to represent a location in the maze. You will not need to use any MazeCoord methods to solve this problem.
- Selected `LinkedList` and `ListIterator` methods are shown on the code handout.
- `getNeighbors` (one of the helper methods used) will only return valid maze locations (i.e., nothing that goes out of bounds of a 2D maze array).

(problem continued on the next page)

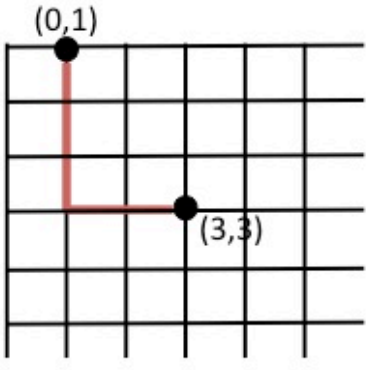
Problem 5 (cont.)

```
/**
   searchFrom returns a path from loc to the exit location, if there is one.
   @param loc the location in the maze to start from.
   @return the path found (i.e., a list of pairwise-adjacent non-wall maze
    coordinates, starting at loc and ending at the exit location),
    or null if there is no path from loc to the exit.
*/

private LinkedList<MazeCoord> searchFrom(MazeCoord loc) {




   if (hasWallAt(loc)) {  // loc is a wall




   }
   else if (alreadyVisited(loc)) { // loc has already been visited




   }
   else if (loc.equals(getExitLoc()) {  // loc is the exit




   }
   else {
      markAsVisited(loc);
// getNeighbhors returns a LL of the valid coordinates adjacent to loc.
      LinkedList<MazeCoord> neighbors = getNeighbors(loc);
      ListIterator<MazeCoord> neighborIter = neighbors.listIterator();
      while (neighborIter.hasNext()) {
         MazeCoord neighborLoc = neighborIter.next();
         LinkedList<MazeCoord> partialPath = searchFrom(neighborLoc);









      }


}
```

Problem 6 [15 points]

Suppose we want to improve the maze search so that it uses a heuristic (rule of thumb) to check locations closer to the exit before locations farther from the exit. The advantage is that since it will quit searching as soon as it finds any path, it may not ever have to explore from those farther locations. Of course, sometimes the only path to the exit will start by moving farther from the exit; we will still find such a path, we just won't explore in that direction first.

For this problem we'll use the "Manhattan distance" instead of the usual distance measure, since we have can only travel in 4 directions anyway. Manhattan distance measures the rectilinear distance between two points (named after distance walking blocks in Manhattan). E.g., suppose the current location is (3,3), and the exit location is (0, 1), then the Manhattan distance between these two points is (3-0) + (3-1) = 3 + 2 = 5. The neighboring point (3,4) is a distance of 6 units from the exit, but another neighboring point, (3,2), is only 4 from the exit. (See diagram below; maze locations are shown as intersections in the grid below.)



Furthermore, we'll assume the existence of the helper function `getNeighbors`, used in the previous problem, that returns a list of the neighbors of a location:

```
LinkedList<MazeCoord> neighbors = getNeighbors(loc);
```

So, to implement the heuristic, we would just need to sort this list by Manhattan distance from the exit before we do the recursive calls to search from the neighbor locations in the list.

(problem continued on the next page)

Problem 6 (cont.)

Implement the following function to reorder the neighbors: It's going to take the neighbors list and reorder it so that the locations closer to the exit appear before locations farther from the exit. Furthermore, you are required to implement it so it uses the Java `Collections.sort` method to reorder the elements. Solutions that don't use the sort method will receive little or no credit. You may not modify the `MazeCoord` class for this problem.

Hint: you will need to implement the following function, but you will also need additional code besides the function itself to make it work. The code handout has useful library information and MazeCoord methods.

```
// reorders the list, neighbors, so that maze locations that are closer to the exit
// come before maze locations that are farther from the exit (uses Manhattan dist.)
private static void reorderNeighbors(LinkedList<MazeCoord> neighbors, MazeCoord exit){
```

Problem 7 [17 points]

Write the static method `isPalindrome`, which takes a `String` and tells whether that string is a one-word palindrome. Your solution must use a `Stack` to help solve the problem (little to no credit will be given for a solution that doesn't use a `Stack`). For full credit, your solution is only allowed to traverse the String once. The string does not have to be a real English word, but it must read the same forwards and backwards to be considered a palindrome. Hint: unlike the solution discussed in class, this one will not involve recursion. See code handout for Stack interface.

Examples:

word	***return value of call to `isPalindrome(word)`***	
`"the"`	`false`	
`"tttt"`	`true`	
`"radar"`	`true`	
`"abccba"`	`true`	
`"abcbca"`	`false`	
`"abccbaa"`	`false`	
`"yay"`	`true`	
`"Yay"`	`false`	*(upper and lower case characters are not considered the same for this problem)*
`"aa"`	`true`	
`"a"`	`true`	
`""`	`true`	

```
// returns true iff the word given is a palindrome
public static boolean isPalindrome(String word) {
```

Extra space for answers or scratch work. (DO NOT detach this page.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.) (DO NOT detach this page.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.