CS200 - Problem Set 5

1. The floor and ceiling functions come up often in computer science. Their domain is the real numbers ($\mathbb{R}$) and their codomain is the integers ($\mathbb{Z}$). $\lfloor x \rfloor$ ("the floor of x") is the largest integer less than or equal to x. $\lceil x \rceil$ ("the ceiling of x") is the smallest integer greater than or equal to x. One reason floor and ceiling appear often in computer science is because computers are much better at dealing with integers than real numbers (why is that?).

   (a) What is $\lfloor -\sqrt{2} \rfloor$?

   (b) Is the ceiling function surjective? Explain why.

   (c) Is the floor function injective? Explain why.

   (d) Prove true or prove false: $\forall x \in \mathbb{R}, \lceil (\lfloor x \rfloor) \rceil = \lfloor x \rfloor$.

   (e) Prove true or prove false: $\forall x \in \mathbb{R}, \lfloor 2x \rfloor = 2\lfloor x \rfloor$.

2. Prove that Algorithm 1 for binary search is correct. Note that if A is sorted in increasing order and no integers are repeated, that means $i < j$ if and only if $A[i] < A[j]$. This is a complex proof, and you will probably not get all of the parts correct. Just try your best, and you'll learn from your mistakes. See last page for a hints.

   Input : (1) Array (list) A containing integers, where there are no repeated integers and the integers are sorted from smallest to largest, (2) an element V

   Output: True if there is an index g such that $A[g] = V$ and False otherwise.

   ```
   l=length of A;
   // Base Case
   if l = 1 and A[1] = V then
   |  return True;
   else
   |  return False;
   end
   // Recursive step
   mid = ⌊l/2⌋;
   if A[mid] < V then
   |  return BinarySearch(A[mid + 1 : l], V);
   else
   |  return BinarySearch(A[1 : mid], V);
   end
   ```

   Algorithm 1: `BinarySearch`(A, V)

3. Explain why we needed to use strong induction instead of regular induction in the proof of binary search. As an example, talk about what happens in the algorithm if the input array is size 5.

4. Suppose a procedure involves m tasks, where task i can be completed in n_i ways. (So task 1 can be completed in n_1 ways, task 2 can be completed in n_2 ways, etc.) Prove using the product rule from class that there are

$$\prod_{i=1}^{m} n_i \tag{1}$$

ways of completing the procedure. Note that if $(a_j, a_{j+1}, a_{j+2}, \ldots, a_k)$ is an ordered set of real numbers, then:

$$\prod_{i=k}^{j} a_i = a_k \times a_{k+1} \times \cdots \times a_{j-1} \times a_j. \tag{2}$$

Note: The version of the product rule we learned in class only applies for 2 tasks. In this problem, you should use that 2-task version as a starting point to prove this more general version.

5. How many surjective functions are there from set $A = \{1, 2, 3, \ldots, n\}$ to set $B = \{1, 2\}$? ($f : A \to B$ is surjective iff $\forall b \in B \, \exists a \in A, \; f(a) = b$.) See hints on final page.

6. Suppose you are making an app and would like each user to create a login ID. You decide the login ids will be strings of lowercase letters that are either 4 or 5 characters long, with no repeated letters. How many ids start with a or end with z? (You don't need to simplify your answer in any way.)

7. Let $G = (V, E)$ be a graph, and let $v \in V$. Please translate the following predicates into English or math (depending on which is given.) You may use V, E, G, and v in your translations if needed.

   (a) G is complete. (A complete graph is a graph where there is an edge between every pair of vertices.)

   (b) $\forall u \in V, |\{w : \{w, u\} \in E\}| < |\{w : \{w, v\} \in E\}|$.

8. How long did you spend on this homework?

Problem 2: Your induction variable n should be the length of the array A. Also, make sure your recursive call is to an array with size between 1 and k inclusive.

Problem 5: First count the total number of functions from A to B. Then count the number that are not surjective. Then take the difference to get the number that are surjective. To count the total number of functions from A to B, use the rule from problem 4. To do this, note that for each function, there is a choice of what to map the first element of A to, what to map the second element of A to, etc.