1. (10 pts) Given the red-black trees T_1 and T_2, which contain m and n elements respectively, we want to determine whether they have some particular key in common. Assume an adversarial sequence that placed the m and n items into the two trees.

   (a) Suppose our algorithm traverses each node of T_1 using an in-order traversal and checks if the key corresponding to the current node traversed exists in T_2. Express the asymptotic running time of this procedure, in terms of m and n.

   (b) Now suppose our algorithm first allocates a new hash table H_1 of size m (assume H_1 uses a uniform hash function) and then inserts every key in T_1 into H_1 during a traversal of T_1. Then, we traverse the tree T_2 and search for whether the key of each node traversed already exists in H_1. Give the asymptotic running time of this algorithm in the average case. Justify your answer.

2. (30 pts) Voldemort is writing a secret message to his lieutenants and wants to prevent it from being understood by mere Muggles. He decides to use Huffman encoding to encode the message. Magically, the symbol frequencies of the message are given by the *Lucas numbers*, a famous sequence of integers discovered by the same person who discovered the *Fibonacci numbers*. The nth Lucas number is defined as $L_n = L_{n-1} + L_{n-2}$ for $n > 1$ with base cases $L_0 = 2$ and $L_1 = 1$.

   (a) For an alphabet of $\Sigma = \{a, b, c, d, e, f, g, h\}$ with frequencies given by the first $|\Sigma|$ Lucas numbers, give an optimal Huffman code and the corresponding encoding tree for Voldemort to use.

   (b) Recall that in the Huffman algorithm, we may specify a simple convention that determines the way a pair of dequeued symbols are arranged in the coding tree relative to their parent. How many optimal Huffman codes could you have provided to Voldemort for the set of frequencies in part (2a)? Justify your answer.

   Hint: (i) symmetries and tie-breaking, and (ii) not every optimal code tree can be produced by a simple convention for Huffman.

   (c) Generalize your answer to (2a) and give the structure of an optimal code when the frequencies are the first n Lucas numbers.

3. (30 pts total) Draco Malfoy is struggling with the problem of making change for n cents using the smallest number of coins. Malfoy has coin values of $v_1 > v_2 > \cdots > v_r$ for r coins types, where each coin's value v_i is a positive integer, and where v_1 is the most valuable coin. His goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized.

(a) A greedy algorithm for making change is the **cashier's algorithm**, which all young wizards learn. Malfoy writes the following pseudocode on the whiteboard to illustrate it, where n is the amount of money to make change for and v is a vector of the coin denominations:

```
wizardChange(n,v) :
   d[1 .. v.len()] = 0  // initial histogram of coin types in solution
   while n > 0 {
      k = r
      while ( v[k] > n and k >= 0 ) { k-- }
      if k==0 { return 'no solution' }
      else { d[k]++ }
   }
   return d
```

Hermione snorts and says Malfoy's code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

(b) Sometimes the goblins at Gringotts Wizarding Bank run out of coins, and make change using whatever is left on hand. Identify a set of U.S. coin denominations for which the greedy algorithm does not yield an optimal solution. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include a penny so that there is a solution for every value of n.)

(c) On the advice of computer scientists, Gringotts has announced that they will be changing all wizard coin denominations into a new set of coins denominated in powers of c, i.e., denominations of $c^0, c^1, \ldots, c^\ell$ for some integers $c > 1$ and $\ell \geq 1$. (This will be done by a spell that will magically transmute old coins into new coins, before your very eyes.) Prove that the cashier's algorithm will always yield an optimal solution in this case.

Hint: consider the special case of $c = 2$.

4. (30 pts) A good hash function $h(x)$ behaves in practice very close to the uniform hashing assumption analyzed in class, but is a deterministic function. That is, $h(x) = k$ each time x is used as an argument to $h()$. Designing good hash functions is hard, and a bad hash function can cause a hash table to quickly exit the sparse loading regime by overloading some buckets and under loading others. Good hash functions often rely on beautiful and complicated insights from number theory, and have deep connections to pseudorandom number generators and cryptographic functions. In practice, most hash functions are moderate to poor approximations of uniform hashing.

Consider the following hash function. Let U be the universe of strings composed of the characters from the alphabet $\Sigma = [\texttt{A}, \ldots, \texttt{Z}]$, and let the function $f(x_i)$ return the index of a letter $x_i \in \Sigma$, e.g., $f(\texttt{A}) = 1$ and $f(\texttt{Z}) = 26$. Finally, for an m-character string $x \in \Sigma^m$, define $h(x) = ([\sum_{i=1}^{m} f(x_i)] \mod \ell)$, where ℓ is the number of buckets in the hash table. That is, our hash function sums up the index values of the characters of a string x and maps that value onto one of the ℓ buckets.

(a) The following list contains US Census derived last names:
`http://www2.census.gov/topics/genealogy/1990surnames/dist.all.last`
Using these names as input strings, first choose a uniformly random 50% of these name strings and then hash them using $h(x)$.

Produce a histogram showing the corresponding distribution of hash locations when $\ell = 200$. Label the axes of your figure. Brief description what the figure shows about $h(x)$; justify your results in terms of the behavior of $h(x)$. Do not forget to append your code.

Hint: the raw file includes information other than the name strings, which will need to be removed; and, think about how you can count hash locations without building or using a real hash table.

(b) Enumerate at least 4 reasons why $h(x)$ is a bad hash function relative to the ideal behavior of uniform hashing.

(c) (10 pts extra credit) Produce a plot showing how the length of the longest chain (were we to use chaining for resolving collisions) grows as a function of the number n of these strings that we hash into a table with $\ell = 200$ buckets. Comment on this trend using the language of the asymptotic growth of functions, worst-case scenarios, and loading factors of hash tables.

5. (20 pts extra credit) Let A and B be arrays of integers. Each array contains n elements, and each array is in sorted order (ascending). A and B do not share any elements in common. Give a $O(\lg n)$-time algorithm which finds the median of $A \cup B$ and prove that it is correct. This algorithm will thus find the median of the $2n$ elements that would result from putting A and B together into one array. (Note: define the median to be the average of the two middle values of a list with an even number of elements.)