Problem 1-1. (Implementing a dictionary)

(a) Describe how a direct-address table can be used to implement a dictionary. What are some advantages and disadvantages to this approach?

(b) Describe how a hash table (with chaining) can be used to implement a dictionary. What are its advantages and disadvantages?

(c) Describe what open addressing is? What are its advantages and disadvantages?

Problem 1-2. (Hashing and Collisions)

(a) Assume we have a hash function h that hashes keys to m buckets and meets the criteria of simple uniform hashing. What is the probability that two hashed keys collide?

(b) When using h to hash n keys, how many collisions do we expect to have? Formally, the set of collisions is $\{(x,y) : x \neq y \text{ and } h(x) = h(y)\}$. So alternatively, what is the expected size of this set?

(c) How does the quantity above, the expected number of collisions, change as the load factor increases? Decreases?

(d) What happens to the expected number of collisions if we keep the load factor the same but increase n (i.e. scale n and m at the same rate)?

Problem 1-3. (Getting familiar with the binary-search-tree property)

(a) Suppose we have numbers between 1 and 1000 in a binary search tree and we want to search for the number 363. Which of the following sequences could **not** be the sequence of the nodes examined?

(i)	$2, 252, 401, 398, 330, 344, 397, 363$
(ii)	$924, 220, 911, 244, 898, 258, 362, 363$
(iii)	$925, 202, 911, 240, 912, 245, 363$
(iv)	$2, 399, 387, 219, 266, 382, 381, 278, 363$
(v)	$935, 278, 347, 621, 299, 392, 358, 363$

(b) Can you generalize your approach from above into an algorithm that outputs whether or not a sequence of keys is valid when searching for some key x? Write the pseudocode for this algorithm. (You can assume a successful search and thus the last key in the sequence should be the key you are searching for).

Problem 1-4. (Building a binary search tree)

(a) Construct a binary search tree by inserting the following keys into an empty tree T in the order given: $15, 10, 13, 17, 2, 5, 20, 16, 11$.

(b) Are there alternative permuatations of these keys that produce this same tree?

(c) Write out the order in which the keys would be printed when performing a *postorder tree walk, preorder tree walk, and inorder tree walk*.

(d) What would T look like after performing $Insert(T, 12); Delete(T, 10)$? What about if we had performed $Delete(T, 10); Insert(T, 12)$? In general, does the order in which we insert a key and delete a key not matter on the resulting tree?