Exercise 2.2 (30pts):
Write a complete Java program that takes O(n) time complexity and takes as input a binary tree that returns true if the tree is a binary search tree; otherwise, it must return false. Explain why the algorithm takes O(n).

- The isBST algorithm within my studentbst.java file takes O(n) time because it visits each node once, and n is the number of nodes within our tree

---

t

Exercise 3. Hashtables and Hashfunctions (40pts)

Following our discussion in class about Hashtables provide the answers to the following questions: (you can either provide a word document or a hand-written document with the answers)

3.1: Define the following terms in your own words:

• Hashtable - data structure that maps keys to values (pointers to records) through the employment of associative arrays. Offers very fast insertion and searching.

• Hashfunction - maps a big number or string to a small integer, essentially creating the accessible key to the value within the array.

• Collision - occurs when hashfunction maps two keys to the same value or record. These must be resolved through collision resolution techniques such as chaining and open addressing

---

3.2: Research and provide the answer to the following question (please cite your resources)

• What is the difference between a Hashtable and a Hashmap?
    - The main difference between Hashtables and Hashmaps lies in synchronization support. While Hashtables support synchronization, hashmaps do not.
    - Another key difference would be that hashtables can not have any null keys or values (pointers and records). However, you are allowed to have one null key and a multitude of null values
• What is the desired running time for search, insert and delete in a Hashtable?
    - Desired running time for search, insert, and delete are O(1), or constant time. This is why hashtables are desirable, because they are so quick.

- Worst case run time would be O(n), when the hashtable has too many keys referring to the same value, or when the hash table hits its load balance and has to rehash

---

3.3: Provide two different ways to deal with collisions and briefly explain both of them.

- Chaining
    - In chaining, we solve the potential problem of collision by making each index its own linked list. In other words, we link all values that are accessed by the same key in a linked list, and put a pointer to the linked list within the hashtable
- Open Addressing
    - In open addressing, collisions are handled by placing the value in an empty slot, thus keeping all elements within the hashtable itself. There are many sub-techniques we can use to do this, such as linear probing, quadratic probing, and double hashing.

3.4: Similar to the fruits examples discussed in class, provide your own example that illustrates storing strings using a hashfunction into a hashtable with linear probing. (Your example should include collision scenarios)

Images Attached Below

4.4 Inserting Strings Into a Hashtable

Suzuki, Kawasaki, Honda, Yamaha, KTM

Kawasaki → Hash → 3

	Hash Table
0	
1	
2	
3	Kawasaki
4	

Suzuki → Hash → 0

0	Suzuki
1	
2	
3	Kawasaki
4	

Honda → Hash → 3
→ collision!
→ linear probing
→ put in next free one
→ Honda goes to 4

0	Suzuki
1	
2	
3	Kawasaki
4	Honda

Yamaha → hash → 1

KTM → hash → 2

	Hash Table
0	Suzuki
1	Yamaha
2	KTM
3	Kawasaki
4	Honda

Hash Table Full!

Exercise 5) Fill up the table below in Big O Notation (30pts)

Data Structure	Deletion	Insertion	Search
Sorted Array	O(n)	O(n)	O(log n)
Array	O(n)	O(n)	O(n)
LinkedList (deleting anywhere, searching anywhere)	O(1)	O(1)	O(n)
Binary Search Tree (average case)	O(log n)	O(log n)	O(log n)
Binary Search Tree (worst case)	O(n)	O(n)	O(n)
Hash Table	O(1)	O(1)	O(1)