Part A

T(1) = 1
T(N) = 1 + T(N-1) for all N > 1

let N = 5

T(N) = 1 + T(N-1)
T(N-1) = 1 + T(N-2)
T(N-2) = 1 + T(N-3)
T(N-3) = 1 + T(N-4)

T(N) = 1 + T(N-1)
T(N) = 1 + 1 + T(N-2)
T(N) = 1 + 1 + 1 + T(N-3)
T(N) = 1 + 1 + 1 + 1 + T(N-4)

T(N) = 1 + 1 + 1 + ... + 1 + 1 (N terms)

Big-O runtime is:
O(N)

T(100) = 1
T(N) = 1 + T(N-1) for all N > 100

let N = 105

T(N) = 1 + T(N-1)
T(N-1) = 1 + T(N-2)
T(N-2) = 1 + T(N-3)
T(N-3) = 1 + T(N-4

T(N) = 1 + T(N-1)
T(N) = 1 + 1 + T(N-2)
T(N) = 1 + 1 + 1 + T(N-3)
T(N) = 1 + 1 + 1 + 1 + T(N-4)

T(N) = 1 + 1 + 1 + ... +1 + 1 (N terms)

Big-O runtime is:
O(N)

Part B

T(1) = 1
T(N) = N + T(N-1) for all N > 1
let N = 5

T(N) = N + T(N-1)
T(N-1) = N-1 + T(N-2)
T(N-2) = N-2 + T(N-3)
T(N-3) = N-3 + T(N-4)

T(N) = N + T(N-1)
T(N) = N + N-1 + T(N-2)
T(N) = N + N-1 + N-2 + T(N-3)
T(N) = N + N-1 + N-2 + N-3 + T(N-4)

sum:
T(N) = N + N-1 + N-2 + ... + 4 + 3 + 2 + 1 (N(N+1)/2 terms)

Big-O runtime is:
O(N^2)

T(1) = 1
T(N) = N/2 + T(N-1) for all N > 1

let N = 8
T(N) = N/2 + T(N/2)
T(N/2) = N/4 + T(N/4)
T(N/4) = N/8 + T(N/8)

T(N) = N/2 + T(N/2)
T(N) = N/2 + N/4 + T(N/4)
T(N) = N/2 + N/4 + N/8 + T(N/8)

T(N) = N/2 + N/4 + N/8 + … + 1(2N - 1 terms)

Big-O runtime is:
O(N)

Part C

In evaluating the recurrence relationships:

T(1) = 1
T(N) = 1 + T(N-1) for all N > 1

We notice that we keep adding 1 term every time we unroll the recurrence relationship:
T(N) = 1 + T(N/2)
T(N) = 1 + 1 + T(N/4)
T(N) = 1 + 1 + 1 + T(N/8)

Thus, the number of 1s we add is equal to to the number of times (H) we need to divide N by 2 so that the result is 1

N(0.5)^H = 1

which can be written as Log2(N) = H

So the number of operations that takes place is equal to Log2(N)

Part D

1. Is it possible that inserting a key and value into a BST could take a constant number of steps? If yes, describe the tree and what you could insert.
No. Interesting the key is always determined by the length of the tree.

2. Is it possible that inserting a key and value into a BST could take O(log2N) steps? If yes, describe the tree and what you could insert.
Yes. The tree has to be a balanced tree. The value of the key must be a value that would fit as a leaf in this tree (added to the bottom)

3. Is it possible that inserting a key and value into a BST could take O(N) steps? If yes, describe the tree and what you could insert.
Yes. The tree would be a tree in which every non-leaf node has exactly one child subtree: its left subtree.

4. Is it possible that inserting into a BST could take O(N2) steps? If yes, describe the tree and what you could insert.
No. The maximum number of operations possible is the number of nodes, since it is useless to check the same node twice.

Part E

1. How could you change the data structure so that you could use this knowledge to make your code to sometimes run more quickly?
Create a new instance variable (called recentSearch) the stores the item in the tree whose key have been most recently passed to get as an argument.
Every time get is called, the key in the argument will be compared to the key of recentSearch. If they are the same then the functions just returns recentSearch. Otherwise the function works normally.

2. Would this change the Big-O how quickly get runs?
Number of operations is only changed when the argument of get is equal to the key of recentSearch. In the case mentioned before, Big-O is a constant. Otherwise, Big-O is the same.

3. Why would you need to also change the code in the method remove to make this work?
Because remove looks for the given key like get does. So checking if the given key is the same as the value stored in recentSearch can significantly cut down the number of operations.