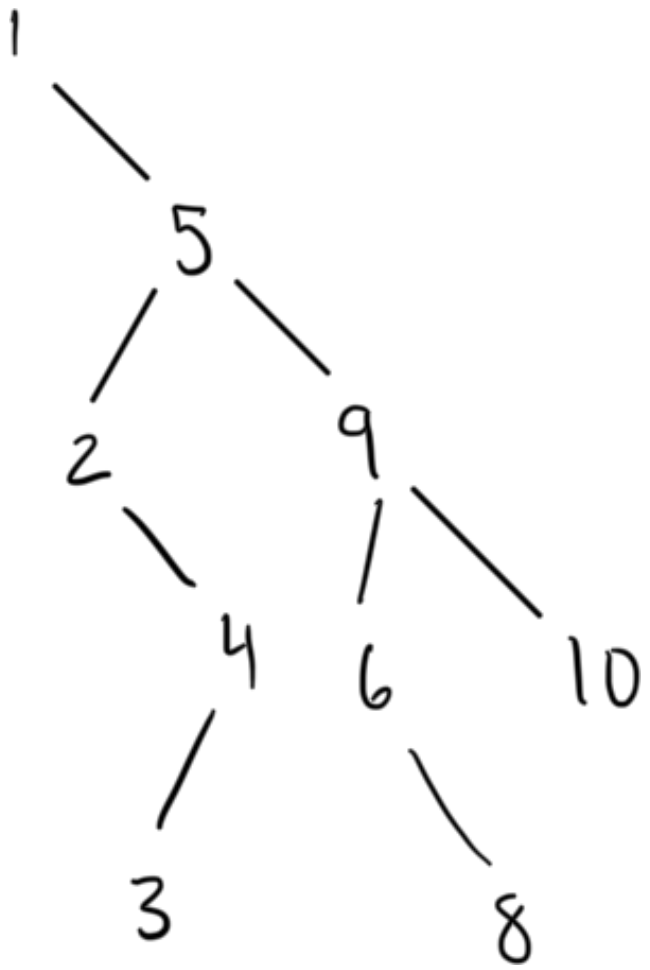
Week 8 — Binary Search Tree

1.



2.

$log_2(n)$
$log_2(1000) = 9.97$

So about 10 steps to search a value in a well-balanced BST with 1000 elements

3.

```
int greatestVal(Node* root){
    if (root == nullptr) throw runtime_error("Empty tree"); // base case 1 - tree is empty so throw error
    if (root->right == nullptr) return root->val; // base case 2 - reached rightmost node (max value)
    return greatestVal(root->right); // recursively keep going right
}
```

4.

```cpp
#include <iostream>
#include <vector>
using namespace std;

struct Node{
    int val;        // value stored in the node
    Node* left;     // pointer to the left child
    Node* right;    // pointer to the right child

    // constructor to initialize a node
    Node(int val): val(val), left(nullptr), right(nullptr){}
};

// insert a new value into the BST
Node* insert (Node* root, int x){
    if (!root) return new Node(x);      // if tree is empty, create a new node
    if (x < root->val){                 // if value is smaller, go to the left
        root->left = insert(root->left, x);
    } else {                            // if value is larger or equal, go to the right
        root->right = insert(root->right, x);
    }
    return root;      // return the root pointer
}

// print values in sorted order
void printTree(Node* root){
    if (!root) return;  // base case: nothing to print
    printTree(root->left);      // left subtree
    cout << root->val << " ";   // print current node value
    printTree(root->right);     // right subtree
}

int main(){
    vector<int> nums = {1, 5, 9, 2, 4, 10, 6, 3, 8};    // list of nums
    Node* root = nullptr;   // new node initialized to null

    // insert all numbers into the BST
    for(int x : nums){
        root = insert(root, x);
    }

    cout << "Tree" << endl;
    printTree(root);

    return 0;

}
```

Video Link: