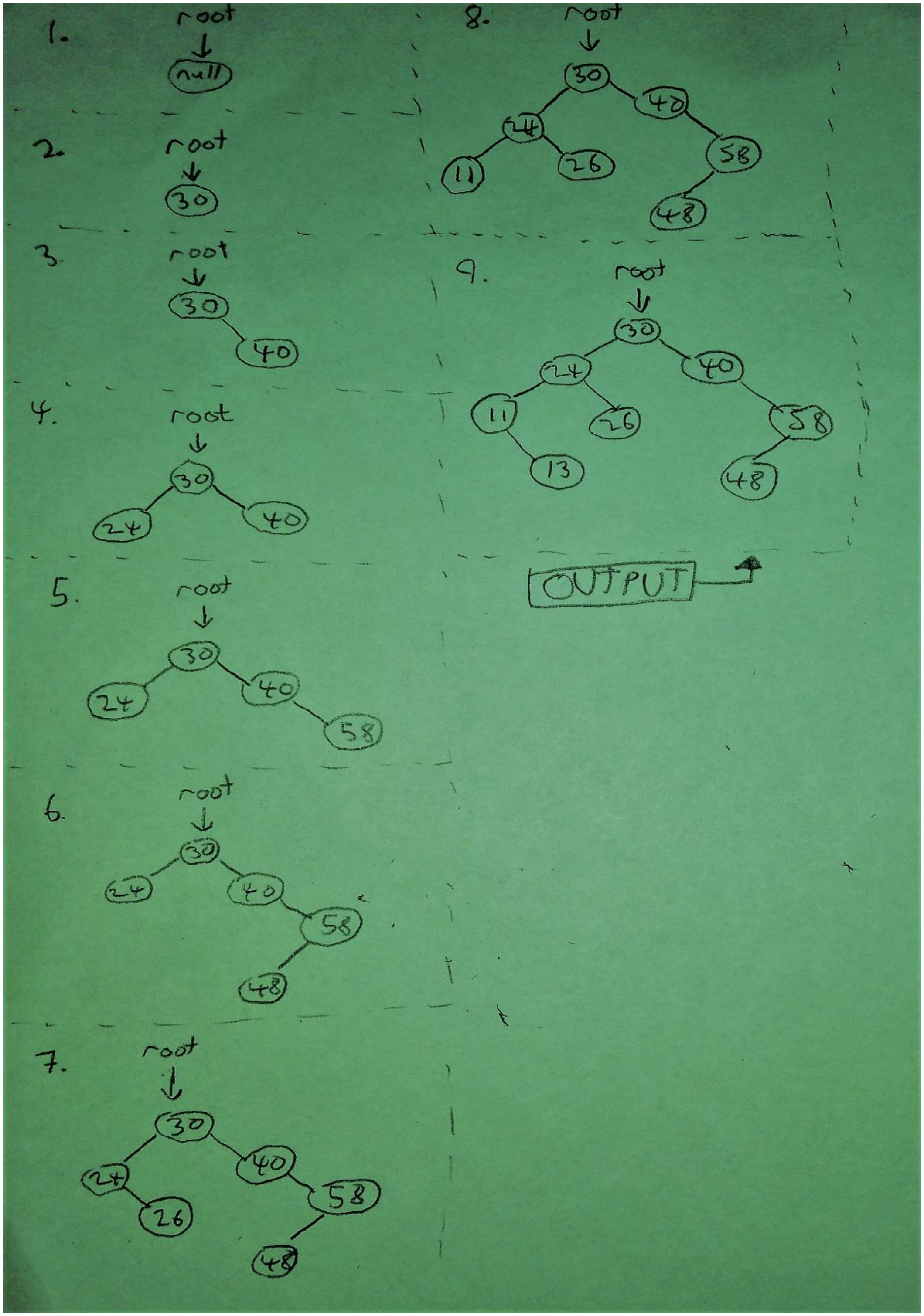
CSC 225 Assignment 3,
Julian Rocha, V00870460
A01, B02

1.



2. Assumptions:
   Worst case of a comparison based sorting algorithm cannot run faster than O(n log n). In our past assignments, we have shown that log(n!) is θ(n log n)

   Proof by contradiction:
   Suppose we take an array and construct a BST from the array which takes time O(k) where O(k) $<$ O(n log n).
   Now suppose we do an in-order traversal of the tree, which we know takes time O(n) (since every node must be visited once).
   If we were to construct a new array from the in-order traversal, we would have a sorted version of the original array.
   The entire process would have taken time
   $O(k) + O(n) < O(n \log n)$.
   From our prior knowledge, however, we know that we cannot perform comparison-based sorting on n items in a worst case faster than O(n log n).
   Therefore O(k) cannot be less than $O(n \log n) = O(\log n!)$

3. Algorithm:

   Since we are dealing with an AVL tree, the height h = log n.

   - If node is empty, stop searching down and return 0
   - If node value < k1 (value too small), recursively call right child
   - If node value > k2 (value too large, recursively call left child
   - Else node value is in range therefore: increment 1 to return value and recursively call both left and right children

   Implementation:

```
int countAllinRange (int k1, int k2) {
        return recursiveMethod (treeRoot, k1, k2)
}
int recursiveMethod (Node n, int k1, int k2) {
        if (n == null) return 0
        if (n.value < k1) return recursiveMethod (n.rightChild, k1, k2)
        if (n.value > k2) return recursiveMethod (n.leftChild, k1, k2)
        else return 1 + recursiveMethod (n.leftChild, k1 k2) + recursiveMethod(n.rightChild, k1, k2)
}
```

4. h(i) = (2i + 5) (mod 11)

Key	Item
0	11
1	39
2	20
3	5
4	16
5	44
6	88
7	12
8	23
9	13
10	94

5. $h(k; i) = [h1(k) + ih2(k)] \pmod t$
   where k is the item key, i is the probe number (starts at 0), and t is the table size

   Since this is double hashing, we know that the probe sequence will consist of equally sized increments, h2(k), added to the original hash h1(k). The sequence stops either once an open spot is found or once h1(k) gets probed (the sequence performs a loop). If we wish to know the max slots the sequence probes, we wish to know how many slots are examined in a full table before we return to h1(k).

   If we assume t is even and h2(k) is even then there are two cases to look at, when h1(k) is even and when h1(k) is odd.

   **Please note some modular arithmetic is used below, namely:

   - $a = b \pmod m \Leftrightarrow a - b = m * k$ for some integer k
   - Any even integer x can be expressed as 2(k) for some integer k
   - Any odd integer y can be expressed as 2(k) + 1 for some integer k

   <u>Case 1:</u> h1(k) even
   $h\ (k; i) = [2(x) + 2i(y)] \pmod{2(z)}$ for some integers x, y and z
   $= 2(a) \pmod{2(z)}$ for some integer $a = x + i(y)$
   $h\ (k; i) - 2(a) = 2(z)(p)$ for some integer p
   $h\ (k; i) = 2[z(p) + a]$

$h(k; i) = 2[b]$ for some integer $b = z(p) + a$
Therefore, the possible probes will always be even ($2b$), meaning at most only half the slots in t will be visited.

<u>Case 2:</u> $h1(k)$ is odd
$h(k; i) = [1+2(x) + 2i(y)] \pmod{2(z)}$ for some integers x, y, z
$= [1 + 2(a)] \pmod{2(z)}$ for some integer $a = x + i(y)$
$h(k; i) - [2(a) + 1] = 2(z)(p)$ for some integer p
$h(k; i) = 2[z(p) + a] + 1$
$h(k; i) = 2[b] + 1$ for some integer $b = z(p) + a$
Therefore, the possible probes will always be odd ($2b+ 1$), meaning at most only half the slots in t will be visited.