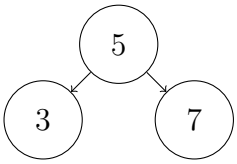
COSC 202, Spring 2024 Exam 2 Practice

The solutions are not intended to be detailed. On the exam you need to provide answers in sufficient detail to get full credit. If you are uncertain what counts as sufficient detail for any particular question, please ask your instructor.

1. Consider the following recursive function called mystery and the Binary Search Tree:

```
mystery(root, low, high)
if root is None then
    return 0
end if
count = 0
if root.key ≥ low then
    count += mystery(root.left, low, high)
end if
if root.key ≤ high then
    count += mystery(root.right, low, high)
end if
if low ≤ root.key ≤ high then
    count += 1
end if
return count
```



What will be the return value of mystery(root, 0, 3)?

Solution: mystery(root, 0, 3) = 1.

2. Given a sorted binary array, we want to efficiently count the total number of 1's in it. For instance

   - Input: nums[ ] = [0, 0, 0, 0, 1, 1, 1]
   - Output: The total number of 1's present is 3

Below is some pseudocode for a recursive algorithm that solves this task.

```
 class Main
{
    // Function to find the total number of 1's
    public static int count(int[] nums, int left, int right)
    {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        if (nums[right] == 0) {
            return 0;
        }

        if (nums[left] == 1) {
            return (right - left + 1);
        }

        int mid = (left + right) / 2;
        return count(nums, left, mid) + count(nums, mid + 1, right);
    }

    public static void main(String[] args)
    {
        int[] nums = { 0, 0, 0, 0, 1, 1, 1 };

        System.out.println("The total number of 1's present is "
                                    + count(nums, 0, nums.length - 1));
    }
}
```

(a) Justify the correctness of this algorithm. In your response make sure to justify each of the base cases, as well as the recursive step.

Solution:

1. Case 1: If list is empty, there cannot be any 1s.
2. Case 2: If the last element is 0, there cannot be any 1s since the list is sorted.
3. Case 3: If the first element is 1 then all elements are 1 because it is sorted.
4. Split the list into two parts. The sum of the number of 1s in the two parts will be equal to the total number of 1s in the list.

(b) Write a recurrence relation for this algorithm and use this to derive the algorithm's time complexity.

Solution:

1. T(n) = T(n/2) + c
2. Time complexity: O(log n)
3. Note even though there are two recursive calls, it is never possible for both branches to keep dividing recursively. One of the branches will end immediately: if the end of first half is 1, then start of second half is necessarily 1. Therefore the second branch will hit the base case immediately. Similarly, if the start of second half is 0, then end of first half is 0. Therefore the first branch will hit the base case immediately.

(c) Let's assume there was only one base case instead of three, where statements **if (nums[right] == 0)** and **if (nums[left] == 1)** did not exist. The code would not return the correct value, of course. It would only return zero. Can you write down the recurrence relation for this code and its subsequent time complexity?

Solution:

1. T(n) = 2T(n/2) + c
2. Time complexity: O(n)
3. Note you can use two methods to solve the time complexity. Master Theorem or direct substitution.
   Master Theorem: a = 2, b = 2, k = 0. $a > b^k$. runtime is $O(n^{\log_2 2})$ or $O(n)$.
   Direct substitution:

$$\left\{\begin{array}{l} 1^{st} : T(n) = 2T(n/2) + c \\ 2^{nd} : T(n) = 2\{2T(n/4) + c/2\} + c \\ 3^{rd} : T(n) = 2\{2\{2T(n/8) + c/4\} + c/2\} + c \\ \vdots \\ l^{th} : T(n) = 2^l T(n/2^l) + l \times c \\ \vdots \\ k : T(n) = 2^k T(n/2^k) + k \times c \end{array}\right\}$$

   Where each line represents $T(n)$ after re-applying the substitution on the right side of the equation repeatedly. Note that every time we apply the substitution, the size of the input n gets smaller and smaller. Observing the pattern, we can guess the expression at any given level l.
   Finally, the input reaches 0 (or null) where the algorithm hits its base case. For easier approximation, we can assume that the input reaches size

of 1 at the last level. We call this level k, where $T(n/2^k)$ becomes $T(1)$. Therefore, $n = 2^k$ or $k = \log_2 n$. Expression will then simplify to

$$T(n) = 2^{\log_2 n} \times T(1) + (\log_2 n) \times c$$

Simplifying gives: $T(n) = n \times 1 + c \log_2 n = O(n)$.

3. Given a BST and a key, design a recursive algorithm that returns the rank of the key — i.e., the number of nodes in the tree less than the key.

Solution:

```
public int rank(Key key)
  {  return rank(key, root);  }

 private int rank(Key key, Node x)
 {  // Return number of keys less than x.key
    // in the subtree rooted at x.
     if (x == null) return 0;
     int cmp = key.compareTo(x.key);
     if      (cmp < 0) return rank(key, x.left);
     else if (cmp > 0) return 1 + size(x.left) + rank(key, x.right);
     else              return size(x.left);
}
```

4. Below is some pseudocode for LSD sorting with one semantic error.

```
sorted_alphabet = {'a': 1, 'b': 2, 'c':3, 'd':4, 'e':5, 'f':6 ... 'z':26}
R = sorted_alphabet.size() # size of alphabet: 26
aux = new array[lst.length]
for j in [0, 1 ... W-1]: #W is max length of words
    count = new int[R+1]
    for i in [0, 1 ... lst.length-1]:
        ind = sorted_alphabet[lst[j][i]]
        count[ind] +=1
    for i in [0, 1 ... R-1]:
        count[r+1] += count[r]
    for i in [0, 1 ... lst.length-1]:
        aux[count[lst[j][i]]] = lst[j]
        count[lst[i]] +=1
    for i in [0, 1 ... lst.length-1]:
        lst[i] = aux[i]
```

(a) (4 points) How would the erroneous code above sort the following list of strings [“cap”, “bee”, “hen”, “cat”, “ant”]? *Hint: it might be helpful to identify the error before trying to trace the entire code.*

Solution: It would sort by the last letter. [“bee”, “hen”, “cap”, “cat”, “ant”]. “cat” is before “ant” because the second letter of “cat” comes before the second letter of “ant”.

(b) (3 points) How would you fix the error?

Solution: Change the first line so we are iterating backwards, instead of forwards. (for j in [0, 1 ... W-1]).