Homework 3 Solutions
Due: Thursday 6 October 2022 by 11:59 PM

Instructions. Type your answers to the following questions and submit as a PDF on Gradescope by the due date and time listed above. (You may write your solutions by hand, but remember that it is at your own risk as illegible solutions will not receive credit.) Assign pages to questions when you submit. **For all questions, you must show your work and/or provide a justification for your answers.**

Note on Academic Dishonesty. Although you are allowed to discuss these problems with other people, your work must be entirely your own. It is considered academic dishonesty to read anyone else's solution to any of these problems or to share your solution with another student or to look for solutions to these questions online. See the syllabus for more information on academic dishonesty.

Grading. Some of these questions may be graded for completion and some for accuracy.

Question 1.
Consider the pseudocode below.

```
Input: L, a linked list (with a head pointer but no tail pointer) of size N
Output: ???
p := the head of L
pivot(L, p)

procedure pivot (List L, element p)
   ● if L is empty then let L1 and L2 be empty lists and return (L1, L2)
   ● let x be the head of L and remove x from L
   ● let (L1, L2) = pivot (L, p)
   ● if x <= p then add x to the front of L1 else add x to the front of L2
end pivot
```

(a) What does this algorithm do? Note that I want a brief statement of the task that is achieved (not a step-by-step rundown of the code.)
(b)* Write a recurrence relation that describes the runtime of this algorithm in terms of *N* and explain your answer. Then solve the recurrence relation to give the runtime of the algorithm.
(c) Explain what the possible list sizes are for the resulting lists.

*Make sure you consider both best and worst case, if they are different.

(a) The algorithm splits a linked list into to separate linked lists where the first list contains elements that are less than or equal to the head of the original list and the second list contains the elements that are greater than the head of the original list.
(b) Each time we do the call to pivot, we are dealing with a list that has been reduced by 1 (because we removed the head of the list). We are also only doing one recursive calle each time, and the rest of the work is constant. The base case is also constant.
So the recurrence relation would be $T(N) = T(N-1) + 1; T(1) = 1$. The runtime in big-Oh would be $O(N)$.
(c) All the elements could end up in one of the lists. This would happen if the first element was either the largest or smallest element in the original list. On the other hand, if the first element in the list is the median, then the elements would be split more or less evenly between the two lists.

Question 2.
(a) Write and analyze an algorithm that takes two sorted linked lists and merges them together into a single sorted linked list. Make sure your analysis considers all the "extreme" possible inputs and discusses best and worst case, if they are different. Give your runtimes in big-Oh notation. Assume that your lists have a head pointer but not a tail pointer. Note that your algorithm should work even if the lists are not of the same length. You can use N and M in your analysis to represent the sizes of each list.

```
Input: L1 (of size N) and L2 (of size M), sorted linked lists
Output: L3, a sorted linked list that is the combination of L1 and L2
return merge(L1, L2)

procedure merge (List L1, List L2)
   • if L1 is empty return L2
   • if L2 is empty return L1
   • Let x = head(L1).
   • Let y = head(L2).
   • if x <= y then
        ○ remove x from L1
        ○ L3 = merge(L1, L2)
        ○ add x to the front of L3
   • else
        ○ remove y from L2
        ○ L3 = merge(L1, L2)
        ○ add y to the front of L3
   • return L3
end merge
```

Analysis:

In the best case, one of the lists is empty, which means that we can just return the other list (it would be appended to the empty list 3, but that takes O(1) time with the tail pointer). So the runtime is O(1).
In the worst case, we have N + M elements that have to be copied over to the new list, so that would take O(N+M) time.

Question 3.
A full ternary tree is a tree whose internal nodes each have exactly three children.
(a) Write a recursive definition for a full ternary tree.
(b) Write a recursive definition for the number of internal nodes i(T) in a full ternary tree.
(c) Write a recursive definition for the number of external nodes e(T) in a full ternary tree.
(d) Write a recursive definition for the total number of nodes n(T) in a full ternary tree.

(a)
Basis Step. A tree with just one node is a full ternary tree (FTT).
Inductive Step. Let T1, T2, and T3 be FTTs. If the three are joined by making each of them the child of a new root, the resulting tree is also an FTT.

(b)
Basis Step. A full ternary tree (FTT) with just one node has 0 internal nodes, so i(T0) = 0.
Inductive Step. Let T1, T2, and T3 be FTTs. The three can be joined together as the three children of a new node to make a new FTT called T4.
Note that the number of internal nodes in each of the three original trees does not change, but we gain a new internal node (the root of T4). So
$i(T4) = i(T1) + i(T2) + i(T3) + 1.$

(c)
Basis Step. A full ternary tree (FTT) with just one node has 1 external node, so e(T0) = 1.
Inductive Step. Let T1, T2, and T3 be FTTs. The three can be joined together as the three children of a new node to make a new FTT called T4.
Note that the number of external nodes in each of the three original trees does not change, and the only new node (the root of T4) is internal. So
$e(T4) = e(T1) + e(T2) + e(T3).$

(d)
Basis Step. A full ternary tree (FTT) with just one node has 1 external node, so n(T0) = 1.
Inductive Step. Let T1, T2, and T3 be FTTs. The three can be joined together as the three children of a new node to make a new FTT called T4.
Note that the number of nodes in each of the three original trees does not change, and the only new node is the root of T4. So
$n(T4) = n(T1) + n(T2) + n(T3) + 1.$

Question 4.
Using your definitions from Question 3, make a conjecture about the relationship between the number of internal nodes and the number of external nodes in a full ternary tree. To do this, you should draw a few examples of full ternary trees and figure out a pattern. You should be able to come up with an equality relationship between the two. Once you have a good conjecture, prove it using structural induction.

Conjecture: For any FTT T, $e(T) = 2i(T) + 1$.
Proof by structural induction.
Basis Step. Let T0 be an FTT with just one node.
e(T0) = 1 and i(T0) = 0 and 1 = 2(0) + 1.
Inductive Step. Let T1, T2, and T3 be FTTs and let $T4 = T1 \cdot T2 \cdot T3$.
Assume as the IH that
$e(T1) = 2i(T1) + 1$, $e(T2) = 2i(T2) + 1$, and $e(T3) = 2i(T3) + 1$.
$e(T4) = e(T1) + e(T2) + e(T3)$
$= 2i(T1) + 1 + 2i(T2) + 1 + 2i(T3) + 1$ by the IH
$= 2(i(T1) + i(T2) + i(T3) + 1) + 1$
$= 2i(T4) + 1$

Question 5.
Using your definitions from Question 3, make a conjecture about the relationship between the total number of nodes and the number of internal nodes in a full ternary tree. To do this, you should draw a few examples of full ternary trees and figure out a pattern. You should be able to come up with an equality relationship between the two. Once you have a good conjecture, prove it using structural induction.

Conjecture: For any FTT T, $n(T) = 3i(T) + 1$.
Proof by structural induction.
Basis Step. Let T0 be an FTT with just one node.
n(T0) = 1 and i(T0) = 0 and 1 = 3(0) + 1.
Inductive Step. Let T1, T2, and T3 be FTTs and let $T4 = T1 \cdot T2 \cdot T3$.
Assume as the IH that
$n(T1) = 3i(T1) + 1$, $n(T2) = 3i(T2) + 1$, and $n(T3) = 3i(T3) + 1$.
$n(T4) = n(T1) + n(T2) + n(T3) + 1$
$= 3i(T1) + 1 + 3i(T2) + 1 + 3i(T3) + 1 + 1$ by the IH
$= 3(i(T1) + i(T2) + i(T3) + 1) + 1$
$= 3i(T4) + 1$