COMP 2804 Assignment 2 Sample Solutions

March 12, 2021

1 Question 1

A binary tree is:

- either one single node
- or a node whose left subtree is a binary tree and whose right subtree is a binary tree.

Prove that any binary tree with n leaves has exactly 2n - 1 nodes.

Proof by induction:
Base case: n = 1
This binary tree has a single node ($n = 1$ leaves $\rightarrow 2 \cdot 1 - 1 = 1$ node) and it is a binary tree by definition.

Inductive Step:
Assume that any binary tree with k leaves has exactly 2k-1 nodes (Inductive hypothesis).
Now we need to prove that any binary tree with k+1 leaves has exactly
$2(k+1) - 1 = 2k + 2 - 1 = 2k + 1$ nodes.

In order for any binary tree to have 1 more leaf, the number of nodes in the tree needs to increase by 2.
Let's start from any binary tree with k+1 leaves, choose any two leaves that share the same parent node and delete both leaves from the tree.
The number of nodes of the resulting binary tree decreased by 2, however the number of leaves decreased by 1 as the two deleted leaf nodes share the same parent node which makes their parent node a leaf node.
As a result, we have a binary tree with K leaves and using our inductive hypothesis this binary tree has exactly 2k-1 nodes. Therefore, to reconstruct our original binary tree with k+1 leaves back from the binary tree with k leaves and 2k-1 nodes, we re-attach the two deleted leaf nodes to their parent node, increasing the number of nodes in the binary tree by 2 and the number of leaves by 1.

2 Question 2

Algorithm 1: Non-recursive Gossip Algorithm

K = n;
while $K > 4$ **do**
  P_{K-1} calls P_K;
  $K \leftarrow K - 1$
end
P_1 calls P_2;
P_3 calls P_4;
P_1 calls P_3;
P_2 calls P_4;
$K \leftarrow 5$;
while $K \leq n$ **do**
  P_{K-1} calls P_K;
  $K \leftarrow K + 1$
end

The first while loop produces the same calls as the first recursive call in the recursive algorithm. The following 4 calls after the first while loop replicates the phone calls produced by the base case. The second while loop produces the same calls as the second recursive call in the recursive algorithm.

3 Question 3

The algorithm is correct. After the algorithm terminates all the people know all the pieces of gossip.
In step 1 of the algorithm, P1 collects all the pieces of gossip from each P_i where $i > 1$.
By the end of step 1, P1 knows all the pieces of gossip.
In step 2 of the algorithm, P1 passes the collected pieces of gossip to each P_i where $i > 1$.
Therefore, the algorithm is correct.

The number of calls made in this algorithm:
In step 1, n-1 phone calls are made where P1 calls all the other n-1 people.
In step 2, n-1 phone calls are made where P1 calls all the other n-1 people.
In total, the algorithm makes $(n-1) + (n-1) = 2n - 2$

The number of calls made by the algorithm we have seen in class equals $2n - 4$. Since the algorithm proposed in this question makes more calls than the algorithm proposed in class, this indicates that this algorithm is not optimal as it makes more calls.

If we measure optimality/complexity using big O notation rather than the exact number of calls.
The algorithm proposed in this question has complexity $\mathcal{O}(n)$.
The algorithm we have seen in class, which is we know is optimal, has complexity $\mathcal{O}(n)$.
Therefore, the algorithm proposed in this question is optimal.

4 Question 4

Prove that for all $n \geq 0$,

$$a_n = f_{n+1} - 1 \tag{1}$$

Proof by induction.
Base case: n = 0.

$$a_0 = f_{0+1} - 1 = 1 - 1 = 0 \tag{2}$$

Base case: n = 1

$$a_1 = f_{1+1} - 1 = f_2 - 1 = 1 - 1 = 0 \tag{3}$$

Both cases are correct as no addition operations are performed when calculating f_0 and f_1 since these are the base cases.

Before we move to the inductive step, we note the following useful observation:

$$a_n = \begin{cases} 0 & if n \in \{0, 1\} \\ a_{n-1} + a_{n-2} + 1 & if n > 1 \end{cases}$$

The base cases of this recurrence relation are true because for f_0 and f_1 we do not do any addition at all. Therefore, the number of addition operations is equal to zero.
As for the case of $n > 1$, since the addition operation is only performed at the else-case of the fibonacci algorithm, the number of addition operations needed to calculate f_n is the number of addition operations needed for each of the two recursive calls to calculate f_{n-1} and f_{n-2} respectively. Then one more addition operation is needed to add up f_{n-1} and f_{n-2}.

Inductive step:
Assume that the following is true for all $k \geq 1$

$$a_k = f_{k+1} - 1 \text{ (Inductive Hypothesis)} \tag{4}$$

Prove that

$$a_{k+1} = f_{k+2} - 1 \tag{5}$$

From the recurrence relation we derived for a_n, we can write the following:

$$a_{k+1} = a_k + a_{k-1} + 1 \tag{6}$$

From our inductive hypothesis we have the following:

$$a_k = f_{k+1} - 1 \tag{7}$$

$$a_{k-1} = f_k - 1 \tag{8}$$

Therefore, we can write the following

$$\begin{aligned} a_{k+1} &= a_k + a_{k-1} + 1 \\ &= f_{k+1} - 1 + f_k - 1 + 1 \\ &= f_{k+1} + f_k - 1 \\ &= f_{k+2} - 1 \end{aligned} \tag{9}$$

Therefore, Proof is complete.

The algorithm is not efficient in terms of the total number of operations carried out. Without you having to give the actual such number, can you pin-point exactly where the inefficiency results from ?

The inefficiency results from calculating the same Fibonacci number multiple times. For example, in order to calculate f_n, we need to calculate recursively $f_{(n-1)}$ and $f_{(n-2)}$.
$f_{(n-1)}$: makes two recursive calls for $f_{(n-2)}$ and $f_{(n-3)}$.
Therefore, $f_{(n-2)}$ is going to be called twice and the same applies for smaller values of n where the recursive function will be called several times for each value.

In order to resolve this issue, we can store the calculated values in a hash table and modify the code such that before doing any recursive calls we check whether the corresponding Fibonacci number has been calculated by another branch of the recursion tree. Only if it has not been calculated before, we calculate it recursively, otherwise, we simply query the value from the hash table.

5 Question 5

Line 1 performs one comparison operation.
Line 2 performs one comparison operation.
Line 1 and 2 are executed once in each iteration of the loop. Therefore, each iteration of the loop makes a total of 2 comparisons.

The loop iterates a total of (n-1) times. Therefore, the total number of comparisons is: $2 \cdot (n-1)$

6 Question 6

For $n \geq 1$:

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \tag{10}$$

We use proof by induction.
Base case: n=1

$$\begin{aligned} \text{LHS of equation } &= 1 \\ \text{RHS of equation} = \frac{1(1+1)}{2} &= \frac{2}{2} = 1 \end{aligned} \tag{11}$$

Inductive step:
Assume for some value $k > 1$ the following:

$$1 + 2 + ... + k = \frac{k(k+1)}{2} \tag{12}$$

This is our inductive hypothesis.
Now we need to prove the following:

$$1 + 2 + ... + k + (k+1) = \frac{(k+1)(k+2)}{2} \tag{13}$$

Now we start from the LHS of the equation and attempt to reach the RHS as follows:

$$\begin{aligned} 1 + 2 + ... + k + (k+1) &= \frac{k(k+1)}{2} + (k+1) \text{ Using inductive hypothesis} \\ &= \frac{k(k+1) + 2(k+1)}{2} \\ &= \frac{(k+1)(k+2)}{2} \end{aligned} \tag{14}$$

The LHS is equal to the RHS, therefore proof is complete.

7 Question 7

For $n \geq 1$:

$$1^2 + 2^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6} \tag{15}$$

We use proof by induction.
Base case: n=1

$$\begin{aligned} \text{LHS of equation } &= 1^2 = 1 \\ \text{RHS of equation} = \frac{1(1+1)(2 \cdot 1 + 1)}{6} &= \frac{1 \cdot 2 \cdot 3}{6} = 1 \end{aligned} \tag{16}$$

Inductive step:
Assume for some value $k > 1$ the following:

$$1^2 + 2^2 + ... + k^2 = \frac{k(k+1)(2k+1)}{6} \tag{17}$$

This is our inductive hypothesis.
Now we need to prove the following:

$$1^2 + 2^2 + ... + k^2 + (k+1)^2 = \frac{(k+1)((k+1)+1)(2(k+1)+1)}{6} \tag{18}$$

First, we simplify the RHS of the equation as follows:

$$\begin{aligned} \frac{(k+1)((k+1)+1)(2(k+1)+1)}{6} &= \frac{(k+1)(k+2)(2k+3)}{6} \\ &= \frac{(k+1)(2k^2+7k+6)}{6} \end{aligned} \tag{19}$$

Now we start from the LHS of the equation and attempt to reach the simplified RHS as follows:

$$\begin{aligned} 1^2 + 2^2 + ... + k^2 + (k+1)^2 &= \frac{k(k+1)(2k+1)}{6} + (k+1)^2 \text{ Using inductive hypothesis} \\ &= \frac{k(k+1)(2k+1) + 6(k+1)^2}{6} \\ &= \frac{(k+1)(k(2k+1) + 6(k+1))}{6} \\ &= \frac{(k+1)(2k^2 + k + 6k + 6))}{6} \\ &= \frac{(k+1)(2k^2 + 7k + 6))}{6} \end{aligned} \tag{20}$$

The LHS is equal to the RHS, therefore proof is complete.

8 Question 8

Part a: How many 66-element subsets of S are there whose largest element is equal to k?
We define the following process:

1. Choose element k from S, there is one way to do that.
2. Choose 65 elements from the set {1,2,....,k-1} , there is $\binom{k-1}{65}$ ways of doing that.

Using product rule: there are $= 1 \cdot \binom{k-1}{65} = \binom{k-1}{65}$ many 66-element subsets of S are there whose largest element is equal to k.

Part b:
We will prove the following by showing that both sides are counting the same thing, therefore the two quantities must be equal.

$$\sum_{k=66}^{n} \binom{k-1}{65} = \binom{n}{66} \tag{21}$$

Let's define the following:
the number of ways to select a 66-element subsets of S whose largest value is k where $66 \leq k \leq n$.

To count that we can use the following two approaches:
Approach 1:
In this approach we determine at first the value of k, i.e. the largest value in the subset to be selected.
Then we select the remaining 65-elements of the subset.
The process is as follows:

1. For each possible value of k
   (a) Choose k from S and add it to the subset: there is 1 way of doing that.
   (b) Choose 65 elements from the set {1,2,3,....k-1} and add them to the subset: we can do this in $\binom{k-1}{65}$ ways

Since the subsets with different maximum values are clearly non-overlapping as the largest element in any given set is unique, we get the following:

$$\sum_{k=66}^{n} \binom{k-1}{65} \tag{22}$$

Approach 2:
In this approach, since $n \geq 66$ and $66 \leq k \leq n$. We can simply select any 66-element subset of S and choose the largest element of the selected subsets to be equal k.
Since we are choosing exactly 66 elements, it is guaranteed that the maximum element of the selected subset is within the valid range of values for k.
Therefore, the number of ways to do that is equal to $\binom{n}{66}$
Since we have shown the we can count the same thing using approaches 1 and 2, it is clear that the two quantities are equal.

9 Question 9

$$f(n) = \begin{cases} 1 & if n = 1 \\ 2 \cdot f(n-1) + 5 & if n > 1 \end{cases}$$

We plug in a few values in the recurrence relation and guess the following non-recursive solution.

$$f(n) = 3 \cdot 2^n - 5 \tag{23}$$

Proof by induction:
Base case: n =1

$$\begin{aligned} f(1) &= 3 \cdot 2^{(1)} - 5 \\ &= 6 - 5 \\ &= 1 \end{aligned} \tag{24}$$

Inductive step:
Assume $f(k-1) = 3 \cdot 2^{k-1} - 5$ is true for some $k > 1$ (inductive hypothesis).
Prove $f(k) = 3 \cdot 2^k - 5$
Proof:

$$\begin{aligned} f(k) &= 2 \cdot f(k-1) + 5 \\ &= 2 \cdot (3 \cdot 2^{k-1} - 5) + 5 \text{ by inductive hypothesis} \\ &= 2 \cdot 3 \cdot 2^{k-1} - 10 + 5 \\ &= 3 \cdot 2^k - 5 \end{aligned} \tag{25}$$

Proof is complete.

10 Question 10

$$f(n) = \begin{cases} a & n = 0 \\ b \cdot f(n-1) & \text{otherwise} \end{cases}$$

To show that the two versions are defining the same function we use proof by induction.
We would like to prove that:

$$f(n) = a \cdot b^n \tag{26}$$

is the non-recursive form of the following recusrive function:

$$f(n) = \begin{cases} a & n = 0 \\ b \cdot f(n-1) & \text{otherwise} \end{cases}$$

Base case: $(n = 0)$.

$$\begin{aligned} f(0) &= a \cdot b^0 \\ &= a \cdot 1 \\ &= a \end{aligned} \tag{27}$$

Inductive step:
Assume that

$$f(n-1) = a \cdot b^{(n-1)} \tag{28}$$

This is our inductive hypothesis. Now we prove that $f(n) = a \cdot b^n$

$$\begin{aligned} f(n) &= b \cdot f(n-1) \\ &= b \cdot (a \cdot b^{(n-1)}) \text{ Using our inductive hypothesis} \\ &= a \cdot b^n \end{aligned} \tag{29}$$

11 Question 11

To measure the complexity of MergeSortVar, we define T(n) to be the number of comparisons in MergeSortVar for a list l of length n where $n = 3^k$.
If the list has only one element, i.e. (n = 1, k =0) then no comparisons are needed to sort the list
Therefore, $T(1) = 0$.
If the list has more than one element then the number of comparisons needed to sort the list equals the following:

$$T(n) = 3T(\frac{n}{3}) + (\#\text{comparisons to merge } L1, L2 \& L3) \tag{30}$$

$\#$comparisons to merge $L1, L2\,\&L3 = n$ as stated in the question.
$\#$ of comparisons when calling MergeSortVar on $L_1 = \#$ of comparisons when calling MergeSortVar on $L_2 = \#$ of comparisons when calling MergeSortVar on L_3 Since all three lists have equal length $(\frac{n}{3})$.

Therefore, we have the following:

$$T(n) = 3T(\frac{n}{3}) + n \tag{31}$$

We plug in some values for $n > 0$:

$$\begin{aligned} T(3) &= 3T(1) + 3 = 3 \cdot 0 + 3 = 3 \\ T(9) &= 3T(3) + 9 = 9 + 9 = 18 \\ T(27) &= 3T(9) + 27 = 3 \cdot 18 + 27 = 81 \end{aligned} \tag{32}$$

We guess that the closed-form solution to the recurrence relation is:

$$T(3^k) = k \cdot 3^k \qquad T(n) = n \cdot log_3(n)$$

We prove that our guess is correct using induction:
Base case:

$$T(n = 3^0) = T(n = 1) = 1 \cdot log_3(1) = 0 \tag{33}$$

This is correct as when the list has only one element, no comparisons are needed to sort the list.

Inductive step:
Assume

$$T(m) = m \cdot log_3(m) \quad \forall j \text{ where } 1 \leq j \leq k-1 \text{ for a given k} \tag{34}$$

This is our inductive hypothesis.

Using our recursive definition we have the following:

$$\begin{aligned} T(n) &= 3T(\frac{n}{3}) + n \\ &= 3(\frac{n}{3}log_3(\frac{n}{3})) + n \quad \text{Using our inductive Hypothesis} \\ &= n \cdot log_3(\frac{n}{3})) + n \\ &= n \cdot (log_3(n) - log_3(3)) + n \\ &= n \cdot log_3(n) - n \cdot log_3(3) + n \\ &= n \cdot log_3(n) - n \cdot 1 + n \\ &= n \cdot log_3(n) \end{aligned} \tag{35}$$

Therefore, we have proved by induction that MergeSortVar has complexity $\mathcal{O}(nlog_3(n))$ where $n = 3^k$.

12 Question 12

Proof by induction.
Base case:

$$n = 0 \rightarrow 2^n = 2^0 = 1 \tag{36}$$

$$1 \in S \text{ (by definition)} \tag{37}$$

Inductive Step:
Assume that $2^k \in S$ is true for some value $k > 0$. (Inductive Hypothesis)
Prove that $2^{(k+1)} \in S$ is true.

$$\begin{aligned} 2^{(k+1)} &= 2^k.2^1 \\ &= 2.2^k \end{aligned} \tag{38}$$

We know from inductive hypothesis that $2^k \in S$. Therefore, by definition since $2^k \in S \rightarrow 2.2^k = 2^{(k+1)} \in S$.