OBSERVATIONS ON THE TRAVERSALS OF TREES

C. BRIMMER, S. DEIS, B. GROENDYKE, K. KAMMAN, AND Z. YANG

Under the direction of Prof. L. E. Smogor

Table of Contents

1. Introduction

The goal of our study was to find relationships between rooted trees and the scores produced by the transversals that are used to process them. According to *The International Dictionary of Applied Mathematics*, a tree is "a connected non-oriented graph which does not contain cycles. A tree has no multiple edges and no loops...they are the simplest possible, connected graphs." The objects which the tree connects are called

Date: Fall 2008.

* We wish to acknowledge Graham Oster for the original questions he posed about tree traversals.

vertices. A rooted tree is a finite set V (of elements called vertices) and a set of ordered pairs, E, of elements in V such that:

(1) There is one and only one element of V which is not the second element of any pair (called the root).
(2) For every vertex v in the set V, where v is not the root, there is one and only one v_1 in the set V, $v \neq v_1$ such that (v_1, v) in the set E.

Any vertex of a non-rooted tree can be designated as the root; therefore, the root must always be made clear. From now on, when we say "tree," we mean "rooted tree." The nodes that are located the farthest away from the designated root are referred to as leaves. The leaves also are vertices that never appear as the first elements in a pair.

The vertices of a tree are also commonly referred to as nodes. The different nodes of a tree are referred to as parents, children, and siblings. A parent is a node that has at least one other node branched out from it and this out-branching node is called the child. A sibling nodes are two nodes that are on the same level of a tree. A level of nodes are all the nodes an equal number of vertices from the root of the tree. A subtree is all the nodes down to the leaves that root out of a certain node. A branch is a subtree and the node that is its parent.

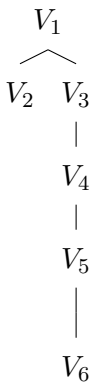
A traversal is a voyage through a tree that starts at the root and visits and processes every node of the tree. To visit a node means to travel to it while moving through the tree. To process a node means to add the node to the list of nodes that the traversal is collecting. All traversals start their voyage by visiting the branch of the tree that is the farthest to left of the root and move as far down the tree as possible, then they back track through the tree in order to reach all the nodes at least once (nodes are reached multiple times through back tracking). This process is called a depth-first search. We used three different traversals when analyzing out trees. They are pre-order, in-order, and post-order. They are defined below:

(1) The pre-order traversal processes the parents of the tree before processing their children.
(2) The in-order traversal processes the child left of the parent, the parent, and then the right child of the parent.
(3) The post-order traversal processes both of the children before visiting the parent.

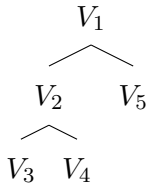
We are able to keep track of how a traversal scores nodes while processing the tree via a method that is best described by imaging that a bell sits at each node. These bells can each only ring once and can only be rung when processing the node, not when visiting. The traversal which rings the most bells (creates the most noise) wins. If two or three traversals process a node at the same time the sound of the ringing is divided among them, which allows for fractional scores. In other words, the score of a traversal method is equal to the sum of the number of nodes that a given traversal processes prior to the other traversal methods. S_{pr} signifies the pre-order score, S_{in} signifies the in-order score, and S_{po} signifies the post-order score. The equations for each traversal method add up to equal m, where m is the number of nodes.

During our study of trees we tended to focus on trees that were linear, di-branching, and/or binary. A few standard definitions and examples are helpful here:

(1) A linear tree is a tree where each non-root parent only has one child and the root may have either one or two children.
(2) A di-branching tree is a tree where the root has two children and the left and right subtrees are linear. An example of a di-branching tree is given below:



(3) A binary tree is a tree where each parent has no more than two children. An example of a binary tree is given below:



2. Five Nodes Can Create Twelve Different Linear Trees

There are 12 different linear, 5-branch trees.

A linear tree with 5 nodes must have 2 terminal nodes and 3 interior nodes. We see that we can change the geometry of the tree by altering the orientation of the interior nodes. The tree must remain linear, so we keep the roots at the ends of the tree. We can create 3 possible orientations. The first orientation has 1 node, or descendant, branched to the left, a center node, and 3 nodes branching to the right. The second orientation has 2 nodes on the left side, a center node, and 2 nodes branching to the right. The third and final orientation has 3 nodes branched to the left, a center node, and 1 node branching to the right. Each of these orientations has 4 possible geometric variations, which means 4 unique trees for each orientation. By adding these sets of 4 unique trees together, we arrive at a total of 12 different linear trees with 5 branches.

We shall prove thise by visualizing each set of trees. Imagine the first orientation with 1 descendant on the left side and 3 descendants on the right. The second descendant on the right can branch either to the left or to the right. Furthermore, the third descendant on the right can branch to the left or the right. These combinations create four unique possibilities: left-left, left-right, right-left, and right-right.

Now imagine the second orientation. It too has four unique possibilities that arise. While the second descendant on the left branches to the left, the second descendant on the right can branch either to the right or left. These are two options. The other two are as such: while the second descendant on the left branches to the right, the second descendant on the right can branch either to the right or left.

Finally, imagine the third orientation. The second descendant on the left can branch either to the left or to the right, and the third descendant on the left can also branch either to the left or to the right. These combinations yield the final four unique possibilities: left-left, left-right, right-left, and right-right.

Note that the combinations from the first and third vertices are not the same, but rather mirror-images of each other. As stated previously, the three sets of 4 unique trees add up to a total of 12 unique, linear, 5-branched trees.

3. Scores for Left-Sided Linear Di-Branching Trees

By testing the scores of the three different traversals (pre-order, in-order, and post-order) for all kinds of linear di-branching trees, we discovered that there was a consistent pattern in the increase of the scores for left-sided linear di-branching trees. A left-sided linear di-branching tree is a linear di-branching tree where there is only one right child from the root and at least a left grandchild from the root.

We found that when the number of nodes in a left-sided linear di-branching tree increases from an odd number of nodes to an even number of nodes, the total score for each traversal increases by one-third. This pattern occurs because when there is an odd number of nodes, the left subtree has an even number of nodes on it. Pre-order collects nodes from the top of the subtree and heads down, while in-order and post-order collect nodes together from the bottom up. Due to the even number of nodes on the subtree, all three traversals meet between the two middle nodes. When the next node is added to bring the total number of nodes to an even number, there is now an odd number of nodes on the left subtree. This change causes all three traversals to meet at the middle left node at the same time and split the score from this node three ways. If one imagines the newly added node to now be the middle node where they all meet, then it is apparent that the traversals collect the same number of nodes as they did before plus a third of the newly added node. Therefore, the total score for each traversal increase by a third when the number of nodes increases from an odd amount to an even amount.

We also found that when the number of nodes increases from an even number of nodes to an odd number of nodes, S_{pr} increases by two-thirds and the S_{in} and S_{po} increase by one-sixth. S_{pr} score increases by two-thirds because the one node that all three of the

traversals share when there is an even number of nodes, comes to belong entirely to pre-order when the number of nodes increases to an odd number. This gain for pre-order leaves one-third of the newly added node for in-order and post-order. Since, those traversals move up the left subtree of the tree together they will reach the remaining one-third at the same time and split it in half; therefore, they each gain one-sixth of the node.

We then developed these observations into equations that will provide the scores for each traversal based on the number of nodes. We defined the number of nodes by the variable m.

Consider a left-sided linear di-branching trees with m nodes, where m is an even number. Pre-order wins half of the number of nodes because it is the sole traversal that starts at the top of the tree and works down the subtrees plus a third of the node that all the traversals reach at the same time when there is an even number of nodes. However, pre-order will always lose the child on the right hand side to post-order because it collect parents first while post-order collects all the children first; therefore, it is one less node that pre-order will win. Therefore, the equation for the pre-order score when the number of nodes is even is:

$$S_{pr} = m/2 - 2/3$$

In-order wins half the amount of nodes that pre-order wins, excluding the node which they all win reach at the same time, because it shares its winnings with post-order. Therefore, the equation for the in-order score when there is an even number of nodes is: $S_{in} = 1/2(m/2 - 1) + 1/3$, which can be reduced to:

$$S_{in} = m/4 - 1/6$$

Post-order wins all the nodes that in-order wins plus the right-side child. Therefore, the equation for the in-order score when there is an even number of nodes is:

$$S_{po} = m/4 + 5/6.$$

Consider now a left-sided linear di-branching trees with m nodes where m is an odd number. Pre-order wins half of the nodes on the left subtree, which is all the nodes but one. Therefore, the equation for the S_{pr} when there is an odd number of nodes is: $S_{pr} = (m - 1)/2$. In-order wins a quarter of the nodes on the left handed subtree because it moves with and shares all of its nodes with post-order. Therefore, the equation for the in-order score when there is an odd number of nodes is: $S_{in} = (m - 1)/4$. Post-order wins a quarter of all the nodes on left side of the tree plus the node on the right side of the tree. Therefore, the equation for the post-order score when there is an odd number of nodes is: $S_{po} = (m - 1)/4 + 1$.

4. Right-Sided Linear Di-Branching Score with an Odd Number of Nodes

The number of nodes is m, and m is an odd number. When we list the three traversals with different m, we can find some patterns in the order:

TABLE 1. Processing for Pre-, In-, and Post-Orders in Right-sided Linear Di-Branching Trees with Odd Amount of Nodes

Order	Round#1	Round#2	Round#3	Round#4	Round#m
Pre	V_1	V_2	V_3	V_4	... V_m
In	V_2	V_1	V_3	V_4	... V_m
Post	V_2	V_m	V_{m-1}	V_{m-2}	... V_1

(1) The first vertex, V_1 ,of the pre-order always scores 1, and the first vertices of the in- and post-order, V_2 ,both score $\frac{1}{2}$.
(2) Half of the rest of the vertices starting from the second in the post-order all score 1. Therefore, the score of the post-order can be expressed as $\frac{m-1}{2} + \frac{1}{2}$, which is $\frac{m}{2}$.
(3) Except for the first vertex, the pre- and in-order have the same score. We know that the total score of the three orders is m, so the total score of the pre- and in-orders starting from the second letter should be the total score minus the score of the first column and the score of the post-order except the first vertex, which is $m - 2 - \frac{m-1}{2}$. Divide it by 2 and plus the score we mentioned in rule one. We have: the pre-order score is $\frac{m}{4} + \frac{1}{4}$ and the in-order is $\frac{m}{4} - \frac{1}{4}$.

Thus, in this case:

$$S_{pre} = \frac{m}{4} + \frac{1}{4}$$

$$S_{in} = \frac{m}{4} - \frac{1}{4}$$

$$S_{post} = m - 2 - \frac{m-1}{2}$$

5. Formulas for Pre-, In-, and Post-Orders with an Even Amount of Nodes

In this explanation, we are concerned with a special kind of linear dibranching, where the tree branches to the right and has one left descendant. This type of tree with is called linear right dibranching.

For a linear right dibranching tree with m nodes, where m is an even number, we denote the scores of pre-order, in-order, and post-order as S_{pre}, S_{in}, and S_{post}, respectively. We observe that, for such trees, where $m \geq 4$,

$$S_{pre} = \frac{3m - 4}{12}$$

$$S_{in} = \frac{3m - 10}{12}$$

$$S_{post} = \frac{3m-1}{6}$$

In a tree with root V_1, and a terminal node V_m, disregard root V_1 and left child V_2: these two vertices fall within the first two rounds of scoring. The first two rounds of scoring remain constant in **all** of the linear right dibranching trees, such that pre-order receives 1 point, in-order receives 0.5 points, and post-order receives 1.5 points (the sum of the first two rounds is 3 points):

TABLE 2. Processing for Pre-, In-, and Post-Orders in Right Linear Dibranching Trees with Even Amount of Nodes

Order	Round#1	Round#2	Round#3	Round#4	Round#m
Pre	V_1	V_2	V_3	V_4	... V_m
In	V_2	V_1	V_3	V_4	... V_m
Post	V_2	V_m	V_{m-1}	V_{m-2}	... V_1

We now turn our attention to the right branch, where the total number of nodes is $m-2$. The 2 in the expression $m-2$ comes from disregarding V_1 and V_2: they are excluded from the right branch. Based on the method of traversals, post-order will move to the bottom of the tree, starting at V_m. Post-Order will have a complete win of all the nodes below the median node. Pre- and in-order share a path, which starts at V_3. Pre- and in-orders split the winnings of the nodes above the median node. Thus, when m is even, all three methods of traversals meet and share the median node.

Imagine a tree where $m = 8$. When $m - 2 = 6$, then post-order receives $7/3$ points exclusively along the right branch while pre- and in-orders each receive $4/3$ points exclusively along the right branch. Now imagine a tree where $m = 10$. When $m - 2 = 8$, then post-order receives $10/3$ points exclusively along the right branch while pre- and in-orders each receive $11/6$ points. Now, imagine a tree where $m = 12$. Where $m - 2 = 10$, then post-order receives $13/3$ points exclusively along the right branch while pre- and in- orders each receive $7/3$ points exclusively along the right branch.

If we observe the pattern of scoring, we can see that as $m - 2$ increases by 2, the post-order score increases by 1: post-order will win half of the nodes along right branch $(m - 2)$ and thus, be rewarded with half of the added nodes. A close inspection of the post-order scores shows us that the post-order scores (along the right branch) are given by

$$\frac{m-2}{2} - \frac{2}{3}$$

Now, add 1.5 points to account for the post-order points in the first two rounds. If we simplify this formula, we arrive at

$$S_{post} = \frac{3m-1}{6}$$

Because there are no left children off the right branch for in-order to favor, pre-order and in-order follow the same processing along the right branch. Thus, they split the points along the right branch and will have identical scores. Remember, the total number of points equals the total number of nodes. Thus, the pre- and in-order scores can be found by subtracting the post-order score from the total number of points left (again, excluding the first two rounds where 3 points were rewarded). Then, divide this calculation by 2 because pre- and in-order split the difference evenly. We arrive at a base formula

$$(m-3) - \frac{\frac{m-2}{2} - \frac{2}{3}}{2}$$

Remember that we excluded the points in the first two rounds. We add 1 point to the pre-order formula to account for the pre-order points in the first two rounds. Also, add 0.5 points to the in-order score to account for the in-order points in the first two rounds. if we simplify both formulas, we arrive at the following:

$$S_{pre} = \frac{3m-4}{12}$$

$$S_{in} = \frac{3m-10}{12}$$

6. Scores for a Left-Sided Linear Di-Branching Tree With Two Right Children

Imagine a left-sided, linear dibranching tree with two right-side children. This means that the tree has a constant two descendants to the right side of the root, and any number of descendants from the left side of the root. For such a tree, the number of vertices shall be noted by the letter m. The vertices are labeled as follows:
$V_1, V_2, V_3, ..., V_m$. When we work in terms of m, we denote the vertices close to V_m as V_{m-1}, V_{m-2}, etc. For example, if $m = 8$, we can write V_7 as V_{m-1}.
When we traverse the tree, we show the scores of pre-, in-, and post-order as S_{pr}, S_{in}, and S_{po}, respectively.

The orders of traversal are:

TABLE 3. Traversal orders

S_{pr}	V_1	V_2	V_3	...	...	...	V_{m-1}	V_m
S_{in}	V_{m-2}	V_{m-3}	V_{m-4}	...	V_2	V_1	V_{m-1}	V_m
S_{po}	V_{m-2}	V_{m-3}	V_{m-4}	...	V_2	V_m	V_{m-1}	V_1

In all cases, we see that pre-order will win the first vertex V_1 for itself and receive 1 point, but we disregard this point here and count it as part of the formula later. Meanwhile, in- and post-order share $V_m - 2$ and each receive $\frac{1}{2}$ point. The other constant that we see is that a three-way tie among all three traversals at $V_m - 1$, where each method receives $\frac{1}{3}$ point.

Therefore, pre-order has a constant $\frac{4}{3}$ points, in-order has a constant $\frac{5}{6}$ points, and post-order has a constant $\frac{5}{6}$ points.

When m is an odd number, pre-order will win the points above the left median for itself. This is defined as

$$\frac{m-3}{2}.$$

Because they follow the same path in the beginning, in-order and post-order will share the points below the left median. We define this as

$$\frac{m-3}{4}.$$

The three points all tie at the left median, so pre-, in-, and post-order all receive $\frac{1}{3}$ point from this spot. When added to the constants, we have the following formulas after simplifying:

$$S_{pr} = \frac{m-3}{2} + \frac{2}{3}$$

$$S_{in} := \frac{m-3}{4} + \frac{2}{3}$$

$$S_{po} = \frac{m-3}{4} + \frac{5}{3}$$

When m is an even number, pre-order will win the top half of the left vertices. We define this as

$$\frac{m-1}{2}.$$

In- and post-order will share the bottom half of the left vertices. We define this as

$$\frac{m-2}{4}.$$

When the constants are added from the first processing round, and the formulas are simplified, we obtain the following:

$$S_{pr} = \frac{m}{2} - \frac{2}{3}$$

$$S_{in} = \frac{m-2}{4} + \frac{1}{3}$$

$$S_{po} = \frac{m-2}{4} + \frac{4}{3}$$

7. Scores for a Right-Sided Linear Di-Branching Tree With Two Left-Children

We are concerned with a Right-sided Linear Di-branching Tree with two children on the left and any number of children on the right. Suppose m vertices: $V_1, V_2, ..., V_m$.
We are asked to give the fomula for S_{pr}, S_{in}, S_{po}.

Table 4. The list of vertices for each order

Pre	V_1	V_2	V_3	V_4	... V_{m-2}	V_{m-1}	V_m
In	V_3	V_2	V_1	V_4	... V_{m-2}	V_{m-1}	V_m
Post	V_3	V_2	V_m	V_{m-1}	... V_5	V_4	V_1

When m is even, we observe that the pre-order will always have V_1 as the first vertex, scoring 1 and the in- and post-order will share V_3, scoring 1/2 for each of them. The three orders will share V_2 as their second vertex, scoring 1/3. Therefore, with the first two vertices, the pre-order will score 4/3, the in- and post-order both 5/6. For the rest of the vertices, half of them can make 1 point to the post-order, which in total is $(m-2)/2$. Except the first two vertices in the traversal, the pre- and in-order will have the same score. We know the sum of the scores of the three traversals is m. When m minus the score of the post-order and the score of the first two vertices of the pre- and in-order, the remaining is $m - 3 - (m-2)/2$, so the scores of the pre- and in-order except the first two vertices are both half of the remaining, which is $(m-3)/2 - (m-2)/4$. Therefore, by adding together, we have the score for all three traversals when m is even:

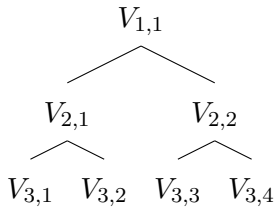
$$S_{pr} = m/4 + 1/3$$
$$S_{in} = m/4 - 1/6$$
$$S_{po} = m/2 - 1/6$$

When m is odd, for the same reason as mentioned above, with the first two vertices, the pre-order will score 4/3, the in- and post-order both 5/6. For the rest of the vertices, except for the middle one, half of them can make 1 point to the post-order, which in total is $(m-3)/2$. The middle vertex will be shared by all three traversals, so the post-order can only score 1/3 from it. Except the first two vertices in the order, the pre- and in-order will have the same score. We know the sum of the scores of the three orders is m. When m minus the score of the post-order and the score of the first two vertices of the pre- and in-order, the remaining is $m-3-(m-3)/2-1/3$, so the scores of the pre- and in-order except the first two vertices are both half of the remaining, which is $(m-3)/2-(m-2)/4-1/6$. Therefore, by adding together, we have the score for all three traversals when m is odd:

$$S_{pr} = m/4 + 5/12$$
$$S_{in} = m/4 - 1/12$$
$$S_{po} = m/2 - 1/3$$

8. Systematic Naming of Nodes in Binary Trees

To label nodes in complete binary trees, we will use **specific node-naming**. A node V is represented by $V_{n,p}$, where n is the level of the node and p is the node's position along that level. The position p must be such that $1 \leq p \leq 2^{n-1}$.
Observe a 3-level complete binary tree, for example. The root will be labeled $V_{1,1}$ because this node lies on level 1 ($n = 1$) and is in position 1 ($p = 1$) along the level. The left child of $V_{1,1}$ lies on level 2 and is in position 1 along the 2nd level and should be named $V_{2,1}$. Thus, the root's right child, which lies in position 2 along level 2, should be named $V_{2,2}$. The children of $V_{2,1}$ and $V_{2,2}$ then lie on level 3, so each of the four children have a value $n = 3$. Because the right child or $V_{2,1}$ lies in the first position along level 3, it is labeled $V_{3,1}$.
A tree with *specific node-naming* labeling:



(Note that for the left child of a vertex, p is odd and for the right child of a vertex, p is even.)
Using specific node-naming, we an predict the name of the parent of the child $V_{n,p}$. However, in calculating the name of the parent, we run into problems, since the arithmetic required to find the position of $V_{n,p}$'s parent depends on whether p is even or

odd. Thus, to find the position x of the parent, we use the equation

$$x = \left[\frac{p}{2} + \frac{1}{2}\right]$$

where $[x]$ is the greatest integer less than or equal to x. For instance, when $p = 7$ or when $p = 8$, the parent is in position 4. The formula yields [4] or [4.5], respectively, and in both cases, the greatest integer will be 4. To find the parent's level y, adhere to the the equation $y = n - 1$, since the parent of a child lies on a level above the child. Following these two functions $[x]$ and $[y]$, we can name the parent of child $V_{n,p}$ such that

$$P = V_{n-1,[\frac{p}{2}+\frac{1}{2}]}$$

where P is the name of the parent.

9. The Traversal of a Complete Binary Tree is Composed of the Traversals of Its Subtrees

We can find that a subtree of a complete binary tree is also a complete binary tree, which means a complete binary tree is actually made up of many smaller complete binary trees. The smallest one of them can be found on the last two levels: one parent with children as leaves. We can call this kind of subtree the unit tree. If we consider two unit trees with the same parent, which can be called twin unit trees, as two nodes, they, together with the parent, will form a new unit tree. Go on the process, finally, we will find that the whole complete binary tree will eventually turn to one single unit tree.
A traversal in a complete binary tree of a certain order will approach the bottom of the tree, where the unit trees stay, and when the traversal reaches one unit tree, it will follow the order to process all nodes in that unit tree. The pre-order visits a parent first, then the left child, finally the right child; the in-order is the left child, the parent, and the right child; and the post-order goes through the two children from left to right before reaching the parent. After that, we see that unit tree as a new node. And when the traversal passed its twins unit tree, we see that unit tree as a node as well. Then a new unit tree emerges and the traversal would finish the new unit tree, before reaching out of it. In the order, if we circle all the vertices of a unit tree together as one vertex and the vertices of its twins as another vertex, there will be the same effect of ordered vertices of the new unit tree we mentioned. All the way up, the whole complete binary tree will become a unit tree with its vertices in the certain order in the traversal. So, we can see that the traversal of that complete binary tree is actually composed of all traversals of its subtrees.

10. Number of Nodes in the Left Subtree of a Complete Binary Tree

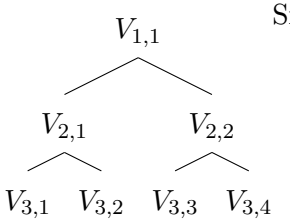
The number of nodes in a Complete Binary Tree can be expressed by the equation $2^n - 1$ where n represents the number of levels in the tree. The nodes that constitute the left subtree of a Complete Binary Tree are the left child of the root and all of its descendants, down to the leaves. The number of nodes in the left subtree is fewer than

half of the number of total nodes in the tree because the root is not included as a node of the subtree. The number of nodes in the left subtree can be represented by the equation

$$2^{n-1} - 1$$

11. Minimum Post Order Score For A Complete Binary Tree

A complete binary tree is a binary tree in which every parent node has two children nodes and if a node has children then its brother node must also have children. An example of a complete binary tree is given below:



Since the post-order traversal processes both the children nodes before it processes the parents, and the other two traversals process the parent nodes before both the children nodes have been processed, post-order processes all the leaves except for the left-most leaf first. Post-order shares the left-most leaf with the in-order traversal because they both start their journey through the tree at that leaf. The number of nodes of a tree is given by:

$$2^n - 1,$$

where n is the number of levels of the tree (a level of a tree is a horizontal line of nodes). This is true because if a two-level tree has three nodes, then adding a new level by requiring all the leave of the two-level tree to have two children causes the three-level tree to have four more nodes than than the two-level tree. The result of this addition is equal to the results of $2^3 - 1$ nodes.
The number of leaves is given by:

$$2^{n-1}$$

This is true because if a two-level tree has two leaves, then adding a new level by requiring all the leaves of the two-level tree to have two children causes the three-level tree to have four leaves. The result of this arithmetic is to the result of 2^{3-1}. Based on the earlier stated observation of how the post-order traversal processes a tree compared with the other two traversals, the post-order traversal must win all the leaves except for half of one. Therefore a bound on the minimum score of for the post-order traversal for a complete binary tree is:

$$S_{po} \geq 2^{n-1} - .5$$

12. Processing Order for Complete Binary Trees

When a complete binary tree of n levels and $2^n - 1$ nodes is traversed,and the processing order is taken note of, certain patterns are detected. When we choose to look at only the far left and right branches of each level, the patterns are easiest to notice. Consider complete binary trees with 4 and 5 levels, respectively.

For a 4-level tree, we look at the following nodes: $V_{1,1}$, $V_{2,1}$, $V_{3,1}$, $V_{4,1}$, $V_{2,2}$, $V_{3,4}$, and $V_{4,8}$. This 4-level complete binary tree has a total of 15 nodes. However, we are only concerned with the 7 nodes listed. We want to look at the order in which these nodes are processed when we traverse the tree, where 1 means the node is the first node to be processed, and 15 means the node is the last node to be processed. We process the tree and determine the processing order for each method. The following table lists this

TABLE 5. Processing order for the far left and right branches of each level

	$V_{1,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-2,2}$	$V_{n-1,4}$	$V_{n,8}$
Pre	1	2	3	4	9	13	15
In	8	4	2	1	12	14	15
Post	15	7	3	1	14	13	12

Based on the order of processing, we look for a way to predict the processing order. We want to find a method other than just the numbers of the processing order, like $1, 4, 12$, etc. We look for a way to express any given node's place in the processing order in terms of n, the number of levels in the tree. This way we can see if processing order follows some pattern or if it is unique for trees with different n values. With the exception of the first 3 nodes processed, which are listed as $1, 2$, and 3, we put the processing order in terms of n and make the following table, which again is simply another way to look at processing order:

TABLE 6. Processing order for the far left and right branches of each level in terms of n

	$V_{1,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-2,2}$	$V_{n-1,4}$	$V_{n,8}$
Pre	1	2	3	n	2^{n-1}+1	2^n-3	2^n-1
In	2^{n-1}	n	2	1	$\frac{3}{4}\ 2^n$	2^n-2	2^n-1
Post	2^n-1	$n+3$	3	1	2^n-2	2^n-3	2^n-n

We now repeat the above process, but this time we look at a 5-level complete binary tree. This tree has 31 nodes. For a 5-level tree, we look at the following nodes:

$V_{1,1}$, $V_{n-3,1}$, $V_{n-2,1}$, $V_{n-1,1}$, $V_{n,1}$, $V_{n-3,2}$, $V_{n-2,4}$, $V_{n-1,8}$, and $V_{n,16}$.
Just as last time, we are only looking at the far left and far right nodes for each level. This time, the processing order will range from 1 to 31, as the following table of processing order shows:

TABLE 7. Processing order for the far left and right branches of each level

	$V_{1,1}$	$V_{n-3,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-3,2}$	$V_{n-2,4}$	$V_{n-1,8}$	$V_{n,16}$
Pre	1	2	3	4	5	17	25	29	31
In	16	8	4	2	1	24	28	30	31
Post	31	15	7	3	1	30	29	28	27

Again we put the processing order from the table above in terms of n. However, this time we exclude the first 4 nodes processed; these are simply listed as $1, 2, 3$, and 4. The processing order becomes much more complicated this time when we put it in terms of n. Table 7 lists the processing order in terms of n:

TABLE 8. Processing order for the far left and right branches of each level in terms of n

	$V_{1,1}$	$V_{n-3,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-3,2}$	$V_{n-2,4}$	$V_{n-1,8}$	$V_{n,16}$
Pre	1	2	3	4	n	2^{n-1}+1	$\frac{3}{4}\ 2^n$+1	2^n-3	2^n-1
In	2^{n-1}	2^{n-2}	4	2	1	$\frac{3}{4}\ 2^n + 1$	$\frac{7}{8}\ 2^n$	2^n-2	2^n-1
Post	2^n-1	2^{n-1}-1	2^{n-2}-1	3	1	2^n-2	2^n-3	$\frac{3}{4}\ 2^n$+2	2^n-n

We compare the processing orders, in terms of n, for the complete binary trees of 4 and 5 levels. Examination shows that some of the nodes on both trees have the same value for processing order. The nodes that have the same value on both trees are listed in Table 8:

These nodes are the root and the left and right extremes of the lowest level. Based on this data, we can say that for a complete binary tree with n levels, where n is any integer

TABLE 9. Instances of same processing order

	$V_{1,1}$	$V_{n,1}$	$V_{n,2^n}$
Pre	1	n	2^n-1
In	2^{n-1}	1	2^n-1
Post	2^n-1	1	2^n-n

greater than or equal to 2, the root node and the farthest left and right nodes on the lowest level have a predictable processing order.
For pre-order, the root node is always the first node processed; the farthest left node on the lowest level is equal to the number of levels in the tree; and the farthest right node is always the last node processed, which is the quantity 2^{n-1}.
For in-order, the root node is always equal to 2 raised to the exponent of $n-1$, (which is the number of levels minus 1); the farthest left node on the lowest level is always the first node processed; and the farthest right node on the lowest level is always the last node processed, equal to the number of nodes in the tree.
For post-order, the root node is always the last node processed, equal to the number of nodes in the tree; the farthest left node on the lowest level is always the first node processed; and the farthest right node on the lowest level is always equal to the quantity 2 raised to the exponent n, minus n.
All other nodes have processing orders unique to 2^{n-1}; they do not have constant processing orders like the three nodes mentioned above.

13. Across Traversal Method

The across traversal method is a traversal method defined by its lateral processing of a row. The across method first processes the left-most descendant on the bottom level of a tree. It then processes every node along the bottom row from left to right. When all the bottom nodes are processed, the across method then moves to the next level up and processes all nodes on this level. It continues this pattern until V_1 is processed. We denote the score of the across method as S_{ac}. The following tables list S_{pr}, S_{in}, S_{po}, and S_{ac} for various types of binary trees.

TABLE 10. Linear Tree, Left-Branching

	2 nodes	3 nodes	4 nodes	5 nodes	6 nodes	7 nodes
S_{pr}	1	$1\frac{1}{4}$	2	$2\frac{1}{4}$	3	$3\frac{1}{4}$
S_{in}	$\frac{1}{3}$	$\frac{7}{12}$	$\frac{2}{3}$	$\frac{11}{12}$	1	$1\frac{1}{4}$
S_{po}	$\frac{1}{3}$	$\frac{7}{12}$	$\frac{2}{3}$	$\frac{11}{12}$	1	$1\frac{1}{4}$
S_{ac}	$\frac{1}{3}$	$\frac{7}{12}$	$\frac{2}{3}$	$\frac{11}{12}$	1	$1\frac{1}{4}$

TABLE 11. Linear Tree, Right-Branching

	2 nodes	3 nodes	4 nodes	5 nodes	6 nodes	7 nodes
S_{pr}	$\frac{1}{2}$	$\frac{3}{4}$	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$
S_{in}	$\frac{1}{2}$	$\frac{3}{4}$	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$
S_{po}	$\frac{1}{2}$	$\frac{3}{4}$	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$
S_{ac}	$\frac{1}{2}$	$\frac{3}{4}$	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$

TABLE 12. Linear Dibranching Tree

	3 nodes	5 nodes	7 nodes	9 nodes	11 nodes	13 nodes	15 nodes
S_{pr}	1	$1\frac{7}{12}$	$2\frac{1}{2}$	$2\frac{5}{6}$	$3\frac{5}{6}$	$4\frac{1}{2}$	5
S_{in}	$\frac{1}{3}$	$\frac{11}{12}$	$1\frac{1}{3}$	$1\frac{2}{3}$	$2\frac{1}{6}$	$2\frac{5}{6}$	$2\frac{5}{6}$
S_{po}	$\frac{5}{6}$	$\frac{11}{12}$	$\frac{5}{6}$	$1\frac{1}{6}$	$1\frac{1}{3}$	$1\frac{1}{3}$	$1\frac{5}{6}$
S_{ac}	$\frac{5}{6}$	$1\frac{7}{12}$	$2\frac{1}{3}$	$3\frac{1}{3}$	$3\frac{2}{3}$	$4\frac{1}{3}$	$5\frac{1}{3}$

As the data tables show, the closer a binary tree is to being a complete binary tree, the more effective the across traversal method becomes. Across method never fully wins a binary tree with only one side of branching, i.e. the right- and left-branching linear trees. When a binary tree with an equal number of descendants on each side has 15 or more nodes, the across method wins. Across method always wins a complete binary tree, regardless of the number of descendants.

TABLE 13. Complete Binary Tree

	3 nodes	7 nodes	15 nodes	31 nodes
S_{pr}	1	$2\frac{1}{2}$	$3\frac{2}{3}$	7
S_{in}	$\frac{1}{3}$	1	2	$2\frac{5}{6}$
S_{po}	$\frac{5}{6}$	$\frac{5}{6}$	$1\frac{1}{2}$	$4\frac{1}{3}$
S_{ac}	$\frac{5}{6}$	$2\frac{5}{6}$	$7\frac{5}{6}$	$16\frac{5}{6}$

14. Addition of the Variable of Time

We first introduced the variable of time to our exploration of trees in order to make the race between traversals more realistic. We first, introduced time by assigning each traversal a time for how long it takes to process nodes and left traveling time as instantaneous. We made observations by studying twelve different complete binary trees, half of which had three nodes and half of which had seven nodes. Also, we assigned the three different traversals the processing rates of one second per node, two seconds per node, or three seconds per node. In none of the trees that we studied did two or all of the traversals have the same processing rates. We found that the traversal with the fastest processing rate always gained the highest score by a large margin, that the traversal with the second fastest processing time always gained the second highest score unless it is a seven node tree and the pre-order traversal moves at the lowest traversal time. In a seven node complete binary tree where the pre-order tree is processing at the slowest rate, it can still gain a node and obtain the second highest score because the first node that it processes is so far away from the nodes that the in-order and post-order traversals process first.

We next added the variable of time by assigning a rate at which the traversals travel from node to node, while leaving the processing time of nodes instantaneous. We tested a complete binary tree with three nodes where each traversal moved at a rate of one node per second. The pre-order traversal obtained the highest score because it was able to immediately process its first node because it processes the root first and because it processed the other two nodes at the same time as the in-order and post-order traversals.

15. Processing Patterns for Linear Dibranching Trees

We are concerned strictly with linear dibranching trees which have the same number of nodes on the left side as they do on the right side, and only two nodes on each level. For example, a tree with three levels that has root $V_{1,1}$ and two descending levels to the left and right.

To look for patterns, we separate our trees into two categories: those with an even number of levels, and those with an odd number of levels. We will first consider trees with an even number of levels.
The best way to look for a pattern is to compare several examples, so we shall look at the processing of trees with 2, 4, and 6 levels for any correlation. We do not look at the score, but rather the order of processing to see if we can consistently predict scores.

TABLE 14. 2-Level Tree

Pr	$V_{1,1}$	$V_{2,1}$	$V_{2,2}$
In	$V_{2,1}$	$V_{1,1}$	$V_{2,2}$
Po	$V_{2,1}$	$V_{2,2}$	$V_{1,1}$

TABLE 15. 4-Level Tree

Pr	$V_{1,1}$	$V_{2,1}$	$V_{3,1}$	$V_{4,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$
In	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$
Po	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,1}$

TABLE 16. 6-Level Tree

Pr	$V_{1,1}$	$V_{2,1}$	$V_{3,1}$	$V_{4,1}$	$V_{5,1}$	$V_{6,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$	$V_{5,2}$	$V_{6,2}$
In	$V_{6,1}$	$V_{5,1}$	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$	$V_{5,2}$	$V_{6,2}$
Po	$V_{6,1}$	$V_{5,1}$	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{6,2}$	$V_{5,2}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,1}$

From this processing information, we can make observations and predictions. We see that pre-order wins the top half of the nodes on the left side. If there are n levels on this tree, we define this as levels 1 through $\frac{n}{2}$. Post-order wins the bottom half of the nodes on the

right side. We write this as levels $\frac{n}{2}+1$ to n. There are also two cases for ties. Pre- and in-order will tie for the top half of right side nodes, not including node $V_{1,1}$. We write this as the levels 1 to $\frac{n}{2}$, not including level 1. In- and post-order will tie the bottom half of the left side nodes. We write this as $\frac{n}{2}+1$ to n.
We now place these criteria in a table to organize our findings.

TABLE 17. Processing predictions, n is even

	Left Side	Right Side
S_{pr}	Levels 1 to $\frac{n}{2}$	Tie levels above $\frac{n}{2}$, exclude level 1
S_{in}	Tie levels $\frac{n}{2}+1$ to n	Tie levels above $\frac{n}{2}$, exclude level 1
S_{po}	Tie levels $\frac{n}{2}+1$ to n	Levels $\frac{n}{2}+1$ to n

We now take the same approach toward trees with an odd number of levels. Again, we observe patterns in a couple of sample processings, in this case the processings of trees with 3 and 5 levels.

TABLE 18. 3-Level Tree

Pr	$V_{1,1}$	$V_{2,1}$	$V_{3,1}$	$V_{2,2}$	$V_{3,2}$
In	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{2,2}$	$V_{3,2}$
Po	$V_{3,1}$	$V_{2,1}$	$V_{3,2}$	$V_{2,2}$	$V_{1,1}$

TABLE 19. 5-Level Tree

Pr	$V_{1,1}$	$V_{2,1}$	$V_{3,1}$	$V_{4,1}$	$V_{5,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$	$V_{5,2}$
In	$V_{5,1}$	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$	$V_{5,2}$
Po	$V_{5,1}$	$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{5,2}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,1}$

Again, from this processing information we can make observations and predictions, though they are more complex. Pre-order will win the left side nodes above the middle.

We define this as levels 1 to $\frac{n}{2} - 0.5$. Post-order will win the left side nodes below the middle. We define this as levels $\frac{n}{2} + 1.5$ to n. There are both two- and three-way ties. Pre- and in-order will tie for the right side levels between, not including, level 1 and the middle. We write this as levels 2 to $\frac{n}{2} - 0.5$. In- and post-order will tie for the bottom half of the left side levels below the middle point. We write this as the levels between $\frac{n}{2} + 1.5$ and n. Finally, there is a three-way tie between all processing orders at the middle nodes of the tree on both the left and right sides. We write this as $\frac{n}{2} + 0.5$, left and right. Again, we place these criteria in a table, to help organize our findings.

TABLE 20. Processing predictions, n is odd

	Left Side	Right Side	Left and Right Sides
S_{pr}	Levels 1 to $\frac{n}{2} - 0.5$	Tie for levels 2 to $\frac{n}{2} - 0.5$	Three way tie for level $\frac{n}{2} + 0.5$
S_{in}	Tie for levels $\frac{n}{2} + 1.5$ to n	Tie for levels 2 to $\frac{n}{2} - 0.5$	Three way tie for level $\frac{n}{2} + 0.5$
S_{po}	Tie for levels $\frac{n}{2} + 1.5$ to n	Levels $\frac{n}{2} + 1.5$ to n	Three way tie for level $\frac{n}{2} + 0.5$

Of what use is this finding? We can use these predictions to determine the scores of a linear dibranching tree with the same number of levels on each side, and also to determine the specific nodes each processing order will win, without ever having to write out the processing order. Simply plug in your number of levels for n, evaluate the formulas, and add them together.

16. Processing Order for Complete Binary Trees

When a complete binary tree of n levels and $2^n - 1$ nodes is traversed,and the processing order is taken note of, certain patterns are detected. When we choose to look at only the far left and right branches of each level, the patterns are easiest to notice. Consider complete binary trees with 4 and 5 levels, respectively.
For a 4-level tree, we look at the following nodes: $V_{1,1}$, $V_{2,1}$, $V_{3,1}$, $V_{4,1}$, $V_{2,2}$, $V_{3,4}$, and $V_{4,8}$. This 4-level complete binary tree has a total of 15 nodes. However, we are only concerned with the 7 nodes listed. We want to look at the order in which these nodes are processed when we traverse the tree, where 1 means the node is the first node to be processed, and 15 means the node is the last node to be processed. We process the tree and determine the processing order for each method. The following table lists this processing order for the far left and right branches of each level:

TABLE 21. Order of processing, 4-level tree

	$V_{1,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-2,2}$	$V_{n-1,4}$	$V_{n,8}$
Pre	1	2	3	4	9	13	15
In	8	4	2	1	12	14	15
Post	15	7	3	1	14	13	12

Based on the order of processing, we look for a way to predict the processing order. We want to find a method other than just the numbers of the processing order, like $1, 4, 12$, etc. We look for a way to express any given node's place in the processing order in terms of n, the number of levels in the tree. This way we can see if processing order follows some pattern or if it is unique for trees with different n values. With the exception of the first 3 nodes processed, which are listed as $1, 2$, and 3, we put the processing order in terms of n and make the following table, which again is simply another way to look at processing order:

TABLE 22. Order of processing, 4-level tree, in terms of n

	$V_{1,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-2,2}$	$V_{n-1,4}$	$V_{n,8}$
Pre	1	2	3	n	2^{n-1}+1	2^n-3	2^n-1
In	2^{n-1}	n	2	1	$\frac{3}{4}\ 2^n$	2^n-2	2^n-1
Post	2^n-1	$n+3$	3	1	2^n-2	2^n-3	2^n-n

We now repeat the above process, but this time we look at a 5-level complete binary tree. This tree has 31 nodes. For a 5-level tree, we look at the following nodes:
$V_{1,1}$, $V_{n-3,1}$, $V_{n-2,1}$, $V_{n-1,1}$, $V_{n,1}$, $V_{n-3,2}$, $V_{n-2,4}$, $V_{n-1,8}$, and $V_{n,16}$.
Just as last time, we are only looking at the far left and far right nodes for each level. This time, the processing order will range from 1 to 31, as the following table of processing order shows:

Again we put the processing order from the table above in terms of n. However, this time we exclude the first 4 nodes processed; these are simply listed as $1, 2, 3$, and 4. The processing order becomes much more complicated this time when we put it in terms of n. A fraction in front of another integer denotes a multiplication of just the fraction and the

TABLE 23. Order of processing, 5-level tree

	$V_{1,1}$	$V_{n-3,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-3,2}$	$V_{n-2,4}$	$V_{n-1,8}$	$V_{n,16}$
Pre	1	2	3	4	5	17	25	29	31
In	16	8	4	2	1	24	28	30	31
Post	31	15	7	3	1	30	29	28	27

integer, it does not include any parts that are being added or subtracted. The following table lists the processing order in terms of n:

TABLE 24. Order of processing, 5-level tree, in terms of n

	$V_{1,1}$	$V_{n-3,1}$	$V_{n-2,1}$	$V_{n-1,1}$	$V_{n,1}$	$V_{n-3,2}$	$V_{n-2,4}$	$V_{n-1,8}$	$V_{n,16}$
Pre	1	2	3	4	n	2^{n-1}+1	$\frac{3}{4}\ 2^n$+1	2^n-3	2^n-1
In	2^{n-1}	2^{n-2}	4	2	1	$\frac{3}{4}\ 2^n + 1$	$\frac{7}{8}\ 2^n$	2^n-2	2^n-1
Post	2^n-1	2^{n-1}-1	2^{n-2}-1	3	1	2^n-2	2^n-3	$\frac{3}{4}\ 2^n$+2	2^n-n

We compare the processing orders, in terms of n, for the complete binary trees of 4 and 5 levels. Examination shows that some of the nodes on both trees have the same value for processing order. The nodes that have the same value on both trees are listed in the following table:

TABLE 25. Instances of same processing order for both trees

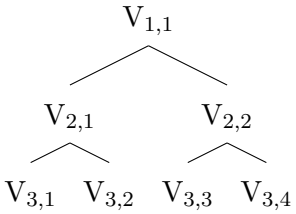
	$V_{1,1}$	$V_{n,1}$	$V_{n,2^n}$
Pre	1	n	2^n-1
In	2^{n-1}	1	2^n-1
Post	2^n-1	1	2^n-n

These nodes are the root and the left and right extremes of the lowest level. Based on this data, we can say that for a complete binary tree with n levels, where n is any integer

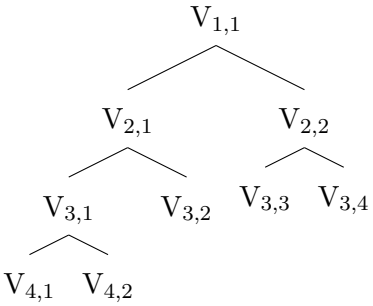
greater than or equal to 2, the root node and the farthest left and right nodes on the lowest level have a predictable processing order.
For pre-order, the root node is always the first node processed; the farthest left node on the lowest level is equal to the number of levels in the tree; and the farthest right node is always the last node processed, which is the quantity 2 raised to the exponent n, minus 1 (this is the total number of nodes in the tree).
For in-order, the root node is always equal to 2 raised to the exponent of $n-1$, (which is the number of levels minus 1); the farthest left node on the lowest level is always the first node processed; and the farthest right node on the lowest level is always the last node processed, equal to the number of nodes in the tree.
For post-order, the root node is always the last node processed, equal to the number of nodes in the tree; the farthest left node on the lowest level is always the first node processed; and the farthest right node on the lowest level is always equal to the quantity 2 raised to the exponent n, minus n.
All other nodes have processing orders unique to the number of levels in the tree; they do not have constant processing orders like the three nodes mentioned above.

17. Adding Two Children to a Complete Binary Tree

The position of two new leaves in a complete binary tree will impact the how many levels above the new leaves changes in what traversal wins each node will be seen. Let's begin with a complete binary tree with n= 3 levels, like the one below.

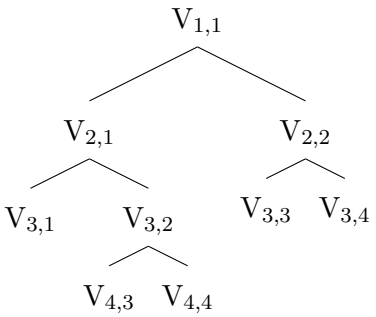


When two children are added to the left-most leaf of a complete binary tree where n=3, a tree like the one below is created.



The impact of adding these two new leaves is that S_{pr} and S_{po} each increase by one-half point. Changes in what traversal order wins each node will be seen in the level immediately above the new leaves and two levels above the new leaves. When two new

children are added to the left-most leaf, the changes will be observed two levels above the new children (n=2 in this example). However, when the new children are placed on a leaf of the original other than the left most one (like the one below) the changes observed will only impact the level that is immediately above the new children (n=3 in this example).



The difference in observed changes is a result of the fact that pre-order is now able to reach more nodes along the left-most branch of the tree before post-order because there are more nodes along the left-most branch of the tree. In the case of adding the two new children to a leaf that is not the left-most leaf, the left branch of the tree is not any longer. Due to the fact that the left branch is not any longer, pre-order and post-order continue to win the nodes along the left branch in the same fashion that they did in the original tree where n=3.

18. The Change of the Position of a Vertex of a Complete Tree in the Pre-order

We are concerned with how the position of a vertex in a complete tree will change when more levels are added.

We will use the vertices $V_{3,1}, V_{3,2}, V_{3,3}, V_{3,4}$ as examples. When there are only 3 levels, the position of $V_{3,1}$ in the pre-order is the 3^{rd}, $V_{3,2}$ the 4^{th}, $V_{3,3}$ the 6^{th}, $V_{3,4}$ the 7^{th}. One level is added. Then $V_{3,1}$ will still be the 3^{rd}, but $V_{3,2}$ will become the 6^{th}, $V_{3,3}$ the 10^{th}, $V_{3,4}$ the 13^{th}. However, when more levels are added, listing the whole traversal is not a wise choice. We need to find a general fomula for the positions of these four vertices.

Because the whole tree is said to be a complete tree, when levels are added, what are added beneath the four vertices $V_{3,1}, V_{3,2}, V_{3,3}, V_{3,4}$ appear to be identical. In fact, every one of the four will lead out two little identical complete trees (with the same number of level). And the number of level of these little complete trees would be $n-3$, where n is the level number of the whole tree. We know that a complete tree with n levels will have 2^n-1 vertices, so the vertex number of the little complete tree coming from the four vertices will be $2^{n-3}-1$.

For $V_{3,1}$, the pre-order will process it before any vertex of those little complete trees added, so the position of $V_{3,1}$ in the pre-order will never change from the 3^{rd}.

For $V_{3,2}$, the pre-order will process the vertices in the two little complete trees from $V_{3,1}$ before processing it. So the position of $V_{3,2}$ will be put backward by the number of the

vertices in the two little complete trees from $V_{3,1}$, which makes its position the $4 + 2(2^{n-3} - 1)^{th}$.
For $V_{3,3}$, the pre-order will process the vertices in the two little complete trees from $V_{3,1}$ and the two little complete trees from $V_{3,2}$ before processing it. So the position of $V_{3,3}$ will be put backward by the number of the vertices in the two little complete trees from $V_{3,1}$ and the two little complete trees from $V_{3,2}$, which makes its position the $6 + 4(2^{n-3} - 1)^{th}$.
For $V_{3,4}$, the pre-order will process the vertices in the two little complete trees from $V_{3,1}$, $V_{3,2}$ and $V_{3,3}$ before processing it. So the position of $V_{3,4}$ will be put backward by the number of the vertices in those little complete trees, which makes its position the $7 + 6(2^{n-3} - 1)^{th}$.

19. Further Examinations

Due to the fact that we have yet to find a tree where in-order traversal obtains the highest score, we would like to try to create a tree where in-order wins. If we are unable to do so, then we would like to prove that the in-order traversal can never beat the pre-order or post-order traversals.

We would also like to develop equations for the exact scores of the traversals based on the number of levels of a complete binary tree. And once we have found ways to determine the scores for certain types of trees, we would like to be able to predict the scores for any binary tree. We would also like to develop a method so that we can predict when the ration of a win is as large as possible for any binary tree.

The largest complete binary tree that we have calculated scores for is one with four levels. We would like to examine complete binary trees with more levels than four levels in order to see if we can determine whether or not the pre-order traversal always wins all the nodes in the first half of the levels.

In order to apply our findings of trees to real-world situations, we would like to continue our exploration with the added variable of time. If we combine our two current ideas of assigning rates to how long it takes the traversals to process nodes and how long it takes the traversals to travel from one node to the next, we will create a system where the competition between how nodes process trees is realistic to the real world. We would also like to find which ratios of rates are need to create a large difference between score like that which was observed where the rate of processing varied and ratio of rates that would create a three-way tie between the traversals.

We would like to continue our study of developing equations which explain where a node will appear in each order based on the nodes placement in a tree. Our final goal for this study would be to develop an equation that will tell us this information for the kth leaf in a tree with n levels.