Name: Georges Hatem

CS590 Homework 2
Reinforcement Exercises

Due Date: February 6, 2022

Problem 2.5.2:

Since we are working with queue, we have to remember to start with front and rear values of -1 (f = -1 and r = -1). So, we will have the array A storing n>= 1 integers (with n known) and f = r = -1 as input in our Algorithm. For the output, we need to get the Array A in reverse order.

So, our job as stated in the problem explanation is to for loop into Array A and index each element into a queue and then remove the elements from the queue and put them in reverse order in the Array A.

My Algorithm pseudocode is as follows:

Algorithm reverseArray(A,n):
Input: An array A storing n>=1 integers; f = r = -1
Output: Make array A in reverse order

```
for i ← 0 to (n-1) do
    if(f = -1) then
        f ← 0
    r ← r + 1
    Q[r] ← A[i]

for i ← 0 to ((n-1)/2) do
    if(f = -1) then
      return (-1)
    else
      A[i] ← A[n-1-i]
      A[n-1-i] ← Q[f]
      f ← f + 1
      if(i = ((n-1)/2)) then
         f ← -1
```

r ← -1

return A

The running time of this Algorithm is O(n) and can be deduced in the following way:

1) The for loop has the following primitive operations. For (i = 0, i < n, i++)
   a. i = 0 is one primitive operations
   b. i < n has (n+1) primitive operations
   c. i++ has n primitive operations
2) f = -1 inside of the first for loop has n primitive operations
3) f = 0 has one primitive operation
4) In r = r + 1, 1 is being added to r and r is being assigned the new value of r. Therefore, 2*n primitive operations
5) In Q[r] = A[i], A[i] has been selected and Q[r] has been assigned equal to A[i]. Therefore, these are two primitive

operations happening n times. Therefore, 2n primitive operations

6) The 2nd for loop has the following primitive operations. For (i=0, i <= ((n-1)/2), i++)

a. i = 0 is one primitive operation

b. i <= ((n-1)/2) has 2*((n+3)/2) primitive operations. The 2 times the value is there because there is arithmetic operations and comparison

c. i++ has ((n+1)/2) primitive operations

7) f = -1 has ((n+1)/2) primitive operations

8) In A[i] = A[n-1-i], we are getting A[n-1-i] and assigning A[i] equal to A[n-1-i]. Therefore, we have 2*((n+1)/2) primitive operations

9) In A[n-1-i] = Q[f], we are getting Q[f] and assigning A[n-1-i] = Q[f]. Therefore, we have 2*((n+1)/2) primitive operations

10) In f = f + 1, we are adding 1 to f and assigning f = f + 1. Therefore, we have 2*((n+1)/2) primitive operations

11) The if statement has 2*((n+1)/2) primitive operations. The 2*value is over there because we are doing arithmetic operations and then comparison

12) f = -1 has one primitive operation

13) r = -1 has one primitive operation

14) return A has one primitive operation

So, from adding the above, we can come up with the following calculation and answer (7n+3 is after adding the results illustrated above for the 1st loop and 6n +12 is after adding the results illustrated above for the 2nd loop):

$$(7n + 3) + (6n + 12) = 13n + 15$$

From the answer above, you can see that the running time of this Algorithm is O(n)

Problem 2.5.5:

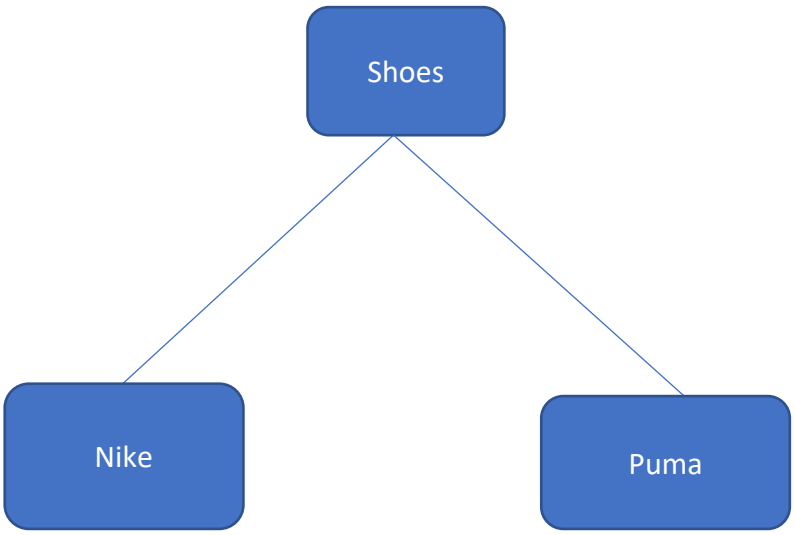
For the 1st question, it is not possible for preorder transversal of T to visit nodes in the same order as the postorder traversal of T. The reason behind that is the following:

Preorder Traversal: The Root Node is first visited, then the subtrees routed at its children are traversed recursively from left to right.

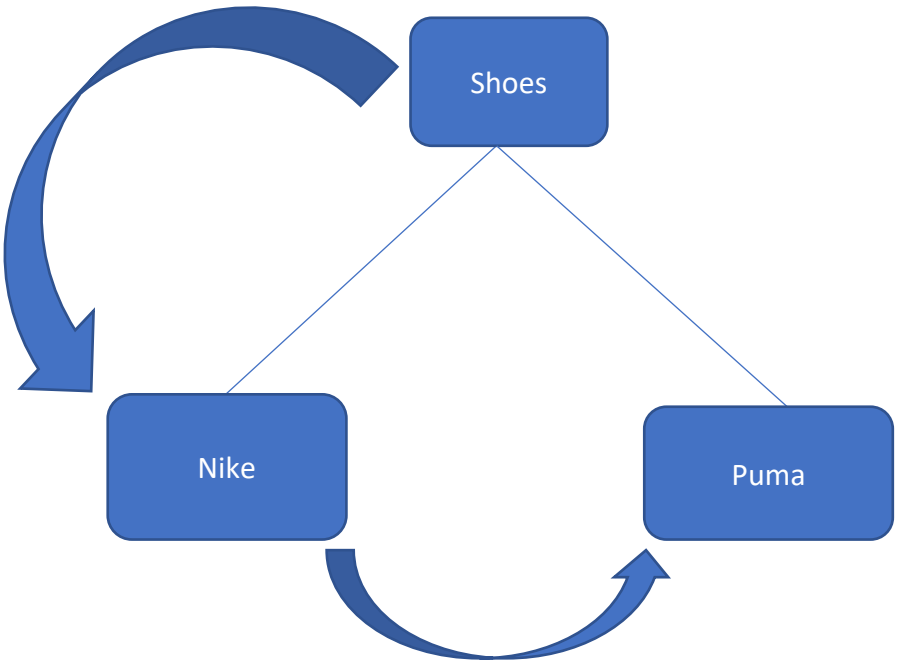
Postorder Traversal: The postorder traversal can be viewed as the opposite of the preorder traversal; the postorder traversal recursively traverses the subtrees routed at the children of the root first and then visits the root.

Let's give an example. Let's consider the following ordered tree with more than 1 node. Let's suppose that shoes is the tree root and under shoes come Children Nike and Puma. Of course, there are many other brands but for

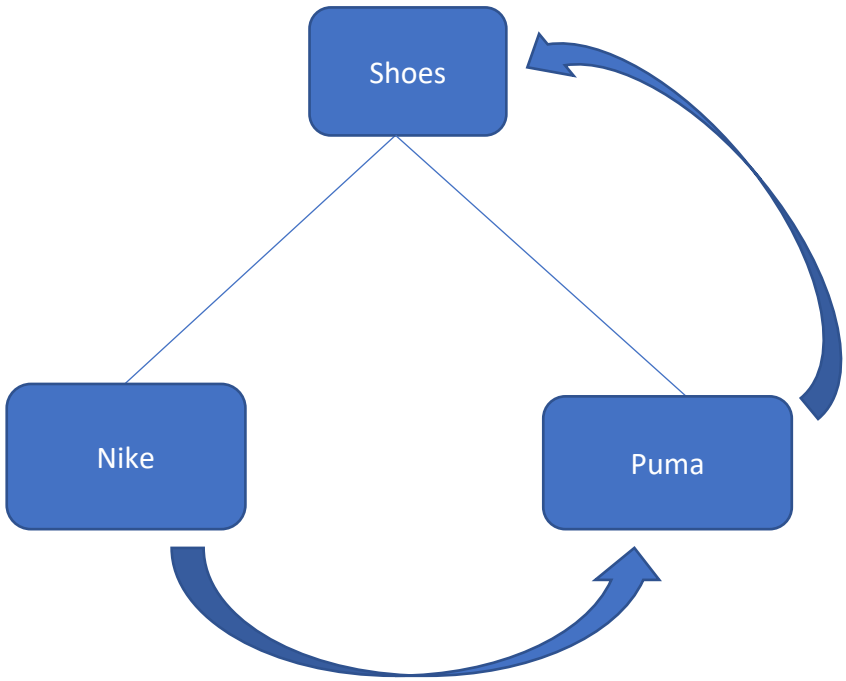
simplicity, let us consider only Nike and Puma as brands:



From the definition above for preorder traversal, the tree gets traversed in this way (for preserved traversal): Shoes, Nike, Puma



From the definition above for the postorder traversal, the tree gets traversed in this way (for postorder traversal): Nike, Puma, Shoes



As you can see from the explanation and analysis above, we can see that:

It is not possible for preorder transversal of T to visit nodes in the same order as the postorder traversal of T

Now, moving into the 2nd question, it is possible that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T. The reason behind that is the following:

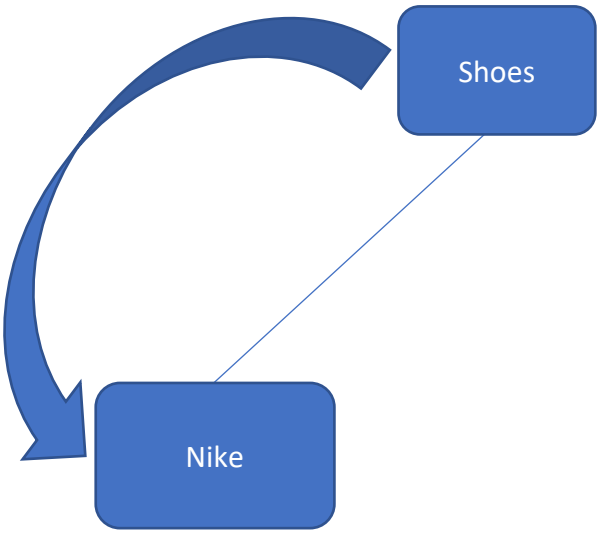
Preorder Traversal: The Root Node is first visited, then the subtrees routed at its children are traversed recursively from left to right.

Postorder Traversal: The postorder traversal can be viewed as the opposite of the preorder traversal; the postorder traversal recursively traverses the subtrees routed at the children of the root first and then visits the root.

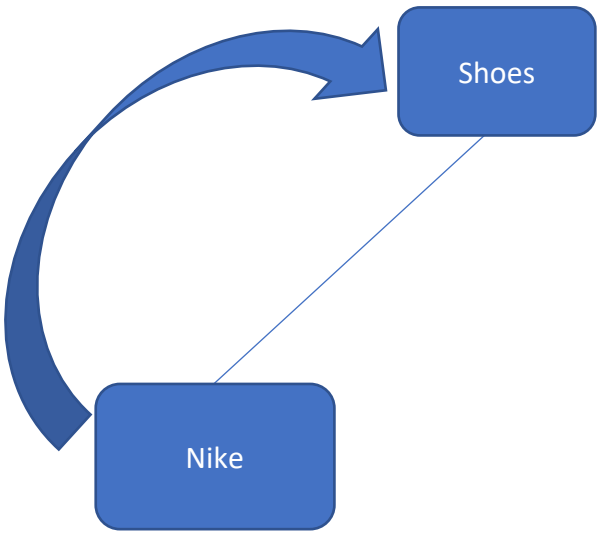
Since the postorder traversal can be viewed as the opposite of the preorder traversal, it is possible in some instances that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T.

Let’s take this example to illustrate. Let’s consider the following ordered tree with more than 1 node. Let’s suppose that shoes is the tree root and under shoes come only Children Nike. Of course, there are many other brands but for simplicity, let us consider only Nike as a shoe brand:

From the definition above for preorder traversal, the tree gets traversed in this way (for preserved traversal): Shoes, Nike



From the definition above for the postorder traversal, the tree gets traversed in this way (for postorder traversal): Nike, Shoes



As you can see from the explanation and analysis above, we can see that in some instances:

It is possible that the preorder traversal of T visits the nodes in the reverse order of the postorder traversal of T.