The easiest way to prove that this algorithm runs in O(n) time is to understand what the algorithm is doing. Given a node, the algorithm checks the left and right child to see if the child exists on either side. The algorithm then recursively checks the children of the left and right child until it finds a null node.

Given that each node in the tree has a parent that is also in the tree, this algorithm will visit each of the n nodes in the tree, as well as find all of the null nodes. By definition, a binary tree with O(n) nodes can have no more than O(n) null nodes. Thus, the running time of the algorithm is O(n) + O(n) = O(n).

Also, by definition, a postorder traversal processes all nodes of a tree by recursively processing all subtrees, then finally processing the root. Thus, since there are n verticies in the graph and “processing” each (or visiting) takes O(1) time, the whole postorder traversal must take O(n) time.

2. [20 points] We define an AVL binary search tree to be a tree with the binary search tree property where, for each node in the tree, the height of its children differs by no more than 1. For this problem, assume we have a team of biologists that keep information about DNA sequences in an AVL binary search tree using the specific weight (an integer) of the structure as the key. The biologists routinely ask questions of the type, “Are there any structures in the tree with specific weight between a and b, inclusive?” and they hope to get an answer as soon as possible. Design an efficient algorithm that, given integers a and b, returns true if there exists a key x in the tree such that a ≤ x ≤ b, and false if no such key exists in the tree. Describe your algorithm in pseudocode and English. What is the time complexity of your algorithm? Explain.

We first note that AVL trees are nothing more than self-balancing binary search trees. Thus we can assume that traversal from the root node to a leaf node will take no more than O(lgn) time.

Using this, what we are going to do is traverse the tree, checking each node we pass as we go. If the node has a value between a and b, we can return true because that is the criteria. If the value is less than a, we know a value between a and b cannot exist in the left subtree since all values in the left subtree are of value less than a as well. Thus we can check the right subtree. Likewise, if the value is greater than b, we know a value between a and b cannot exist in the right subtree since all values in the right subtree are of value greater than b as well. In this case we check the left subtree.

We can do this iteratively until we either reach a value between a and b (in which case we return true) or we reach a root node. If we reach a root node we know that there is no value between a and b in the AVL tree, and thus we return false. Each check at a node takes O(1) time, and as we discussed earlier, traversing an AVL tree takes O(lgn) time, thus the algorithm described takes O(lgn) time.

psuedocode:

item\_exist\_between(a, b, root): // give values for a, b, and the tree root  
 node = root // initialize the active node to root  
 while node != null: // iterate until we hit a leaf node  
 if node.value >= a && node.value <= b: // if value is between a and b return true  
 return true  
 else if node.value < a: // if value less than a, check right subtree  
 node = node.right  
 else: // if value greater than b, check left subtree  
 node = node. left  
 return false // we hit a leaf node, return false

3. Suppose you are consulting for a company that manufactures computer equipment and ships it to distributors all over the country. For each of the next n weeks, they have a projected supplysi of equipment (measured in pounds) that has to be shipped by an air freight carrier. Each week’s supply can be carried by one of two air freight companies, A or B.

• Company A charges at a fixed rate r per pound (so it costs rsi to ship a week’s supply si).

• Company B makes contracts for a fixed amount c per week, independent of weight. However, contracts with company B must be made in blocks of four consecutive weeks at a time.

A schedule for the computer company is a choice of air freight company (A or B) for each of the n weeks with the restriction that company B, whenever it is chosen, must be chosen for blocks of four contiguous weeks in time. The cost of the schedule is the total amount paid to companies A and B, according to the description above.

Give a polynomial-time algorithm that takes a sequence of supply values s1, ..., sn and returns a schedule of minimum cost. For example, suppose r = 1, c = 10, and the sequence of values is

11, 9, 9, 12, 11, 12, 12, 9, 9, 11.

Then the optimal schedule would be to chose company A for the first three weeks, company B for the next block of four contiguous weeks, and then company A for the final three weeks.

In an optimal solution, we wither use company A or B for the ith week. For week i, if we choose company A, we pay r\*si and behave optimally from week i-1 to week i. If we choose company b, then we pay 4\*c, and we behave optimally from week i-4 to week i.

Along this line, we can set up a recurrence such that

???

5. [20 points] Suppose you are acting as a consultant for the Port Authority of a small Pacific Rim nation. They are currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port. Here is a basic sort of problem they face.

A ship arrives with n containers of weight w1, w2, ..., wn. Standing on the deck is a set of trucks, each of which can hold K units of weight. (You may assume that K and wi are integers.) You can stack multiple containers in each truck, subject to the weight restrictions of K. The goal is to minimize the number of trucks that are needed to carry all the containers. This problem is NP-complete.

A greedy algorithm you might use for this is the following. Start with an empty truck and begin piling containers 1,2,3,... onto it until you get to a container that would overflow the weight limit. (These containers might not be sorted by weight.) Now declare this truck “loaded” and send it off. Then continue the process with a fresh truck. By considering trucks one at a time, this algorithm may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

(a) [10 points] Give an example of a set of weights and a value for K where this algorithm does not use the minimum number of trucks.

(b) [10 points] Show that the number of trucks used by this algorithm is within a factor of two of the minimum possible number for any set of weights and any value of K.

a) Assume a value K = 10 and a set of weights [1, 4, 6, 5, 4]

Using the greedy algorithm described, we can pile the following containers on trucks 1..n:

1: [1, 4] (the third container with a weight of 6 would overweigh the truck with max weight 10)  
2: [6] (the fourth container with a weight of 5 would overweigh the truck)  
3: [5, 4] (the final two containers could fit on the third truck

Whereas the optimal solution would be two trucks with container weights as follows:

1: [1, 4, 5]  
2: [6, 4]

This is an example of where a set of weights and a value K where the algorithm does not use the minimum number of trucks.

b) Note that the total weight (Wt) is the sum of all of the container weights. In a perfect world, given a maximum truck load of K, we could unload the cargo in Wt/K trips (given that the container weights cooperate). This will provide us with a minimum bound for the number of trucks T that need to be used. Stated algebraically:

Next we consider the load of two consecutive trucks produced by the greedy algorithm. We note that by definition, the sum of the capacity of two consecutive trucks must exceed K, otherwise they would both be loaded onto the same truck. In pairing off each truck, it is obvious that at least one of each pair is at least half filled. It is possible (in the case that we have an odd number of trucks) that the last truck will also be less than half filled. Given a number of trucks T, we know that at least (T-1/2) are more than half filled. Thus we know that:

Rearranging we get the number of trucks used by the algorithm to be :

Looking at the Wt/K term make clear that the maximum number of trucks used by the greedy algorithm is within a factor of 2 of the optimal solution.