Question 1)

Max\_here = 0

Begin = 0

Temp = 0

End = 0;

Max\_all = 0;

For all int x in arr{

If(max\_here < 0)

Max\_here = x;

Temp = i

Else

Max\_here += x;

If(max\_here > max\_all)

Max\_all = max\_here

Begin = temp

End = I;

The above pseudocode goes through the array only one time, and performs the necessary comparisons as it goes through. The complexity is therefore O(n).

Question 2)

2.1: Sequence c {925, 202, 911, 240, 912, 245, 363} fails because, assuming all numbers in the sequence had to be inserted according to the same insertion algorithm for a binary tree, number 912 would have been inserted to the right of number 911, as opposed to the left where it ends up in the tree. A subsequent search for this number in the tree would fail because 912 is not in its correct spot.

2.2: a)

Function (Node n)

//Left right center

//Recursively call function

If(n.left exists)

LeftMaxed = Function(n.left)//returns node with current maximum on left

leftSum = addAll(n.left)//sums all nodes in left

Else

LeftMaxed = null;

leftSum = 0;

If(n.right exists)

Rightmaxed = Function(n.right)//same, but for right

rightSum = addAll(n.right)//sums all nodes in right

Else

RightMaxed = null;

rightSum = 0;

if(LeftMaxed.value > RightMaxed.value)//implement null as less than

if(leftSum+rightSum+n.value < LeftMaxed.value)

return LeftMaxed;

else

return n

else

if(leftSum+rightSum+n.value < RightMaxed.value)

return RightMaxed

else

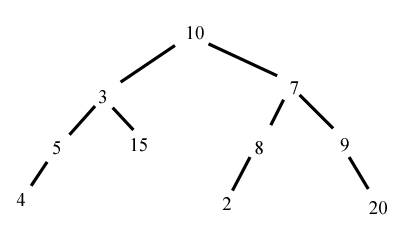
return n

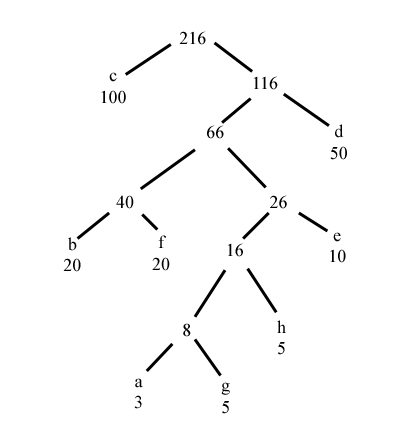
2.2 b) Assuming addAll in the above has a complexity of T(n) = 2T(n/2) + O(1), since it divides each run into 2 parts, worst case in half, as well as a simple addition statement on each call

Function consists of 2 addAll calls per Function call, and two recursive calls to itself,

So the complexity should be F(n) = 2T(n) + 2F(n/2) + O(1), where T(n) is above

2.3 a) True, two traversal types should be able to sufficiently construct a binary tree

b)

Question 3)

3.1: The Huffman tree is as follows

Resulting encodings are therefore:

A -> 010111

B -> 0111

C -> 1

D -> 00

E -> 0100

F -> 0110

G -> 010110

H -> 01010

The program attached is written in Java, but DOES NOT WORK. I ran out of time to finish the code, and seems to have an error with getting Huffman values correctly after a certain point.

Question 4)

16.1-2 : The approach described in the algorithm is a greedy algorithm because it takes the best options by deciding the best ones as the latest starting ones, which is similar to choosing the first ones to finish, but in reverse order. The algorithm is optimal because each attempt leaves the most space in front of the selected options.

16.1-3 :

An example of selecting a task with the least duration is that that task could interrupt two slightly longer choices, such as:

Task 1: starts at 3, ends at 7

Task 2: starts at 6.5, ends at 7.5

Task 3: starts at 7.25, ends at 9

Task 2 is the shortest choice, but interferes with Task 1 and 3, the only other tasks listed

Always selecting one that overlaps the least tasks could lead to picking choices that start by not blocking any tasks, but lead to some smaller tasks being blocked by larger ones if the placement means the small ones interfere more.

Selecting earliest start time could pick a choice that continues all day and blocks all possible choices.