Name: Laurentiu Pavel
Username: lpavel

Homework 3
CS2223

1. In order to solve this problem in linear time, I will use a dynamic programming technique. The main idea is to create another array best, such that best[i]=max(a[i] , best[i-1]+a[i]). The pseudocode:

```
Declare pos_min1=0, pos_min2=0 pos_max=0, max_sum = - infinity
For i=1 .. n
    If a[i] > best[i-1]+a[i]
        Pos_min2=i
        Best[i] = a[i]
    Else
        Best[i] = best[i-1] + a[i]

    If best[i] > max_sum
        Pos_min1 = pos_min2
        Max_sum = best[i]
        Pos_max = i
    Endif
Endfor
```

// the biggest element in stored on max_sum
// the min and max indexes are stored in pos_min1 and pos_max

2.1 In for the list to represent a binary tree, it is required that every node has to be either >= than all the following nodes or <= than all the following nodes.

a. Could be a binary tree. It respects the condition
b. Could be a binary tree. It respects the condition
c. Could not be a binary tree. 911 > 240 but 911 < 912
d. Could be a binary tree. It respects the condition
e. Could not be a binary tree. 347 > 299 and 347 < 621

2.2 a. Pseudocode

```
 copy the tree into an auxiliary one T
max=0;
procedure sum_subtree( Tree T )
        if T==NULL
                return 0;

        T->value = sum_subtree(T.left) + sum_subtree(T.right) + T.value;
        If max < T->value
                Max= T->value
                ID = T->ID

// the maximum sum of all values in subtree is in max
// the id of the node that defines the biggest sum is in ID
```

b. It always has the same complexity:

$T(n) = 2T(n/2) + O(1)$. From master theorem => $T(n) = O(n)$;

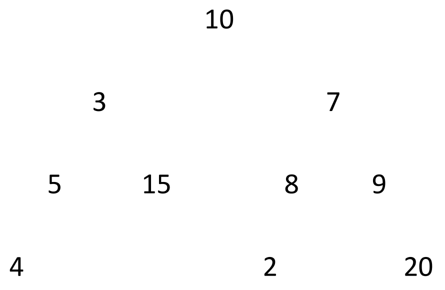
Even though the way it was written supposes that the tree is balanced, if it is not, the complexity is the same while it always goes through all the nodes a single time. Therefore, a more general formula for complexity is:

$T(n) = T(n-1) + O(1)$. Which is also=> $T(n) = O(n)$

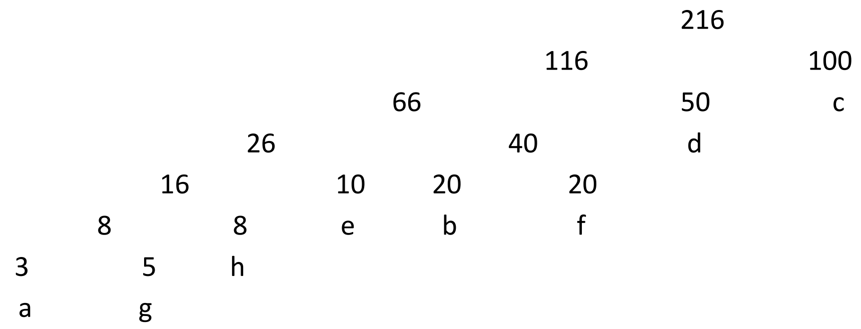
2.3 a) False – Cannot be constructed from pre-order and post-order

```
            b    is equivalent to  b           from pre and post perspective
       a                                 a
```

c) This can be constructed:



3.1



c – 1
d – 01
e – 0001
b – 0010
f – 0011
h – 00001
a – 000000
g – 000001

3.2 The program was written in C++ on the ccc server. Attached to these files there is a makefile. To compile and build the program, type make all in the bash. It will create the executable Huffman.exe.

To run the programs:

- for the encoder type in the bash: ./huffman.exe encode <file to encode> <encoded file>
- for the decoder type in the bash: ./huffman.exe decode <file to decode> <decoded file>

also, the makefile can be called with make clean and it will delete the sandbox, which is just stores the bits before compressing them into a byte.

4.

16.1-2 The algorithm is greedy because it was come up with a choice that always behaves the same, independent of how the other activities are distributed. However, the proof that yields the optimal solution is shown below:

Solution: Using the fact that always choosing from the beginning the activity that ends earliest gives the best result, flip the reverse table. In this case, based on symmetry, choosing the last activity means exactly the same thing as choosing the first one to end every time. While the one that starts from the beginning is true => the algorithm presented here is correct too.

16.1-3

In order to prove that they don’t work, it is enough to give counter-examples for each of them.

Choosing the one with the least duration: taking the 3 activities: A1 : (0 , 15) , A2 : (14, 18), A3 : (17-23), First it will be taken A2 because A1 and A2 overlap. Then A3 will not be taken. It will be only one thing spectating when A1 and A3 are 2 and represent the best choice