Assignment 6

a) Define in pseudo-code an algorithm to calculate the height of a tree. Hint: it needs to be recursive.

```
Algorithm height(T)
    return heightHelper(T, T.root())

Algorithm heightHelper(T, p)
    if T.isExternal(p) Then
        return 0
    else
      lefth ← heightHelper(T, T.leftChild(p))
      righth ← heightHelper(T, T.rightChild(p))
        return MAX(lefth, righth) + 1
```

b) Using the Tree.js, implement your algorithm in a JavaScript function.

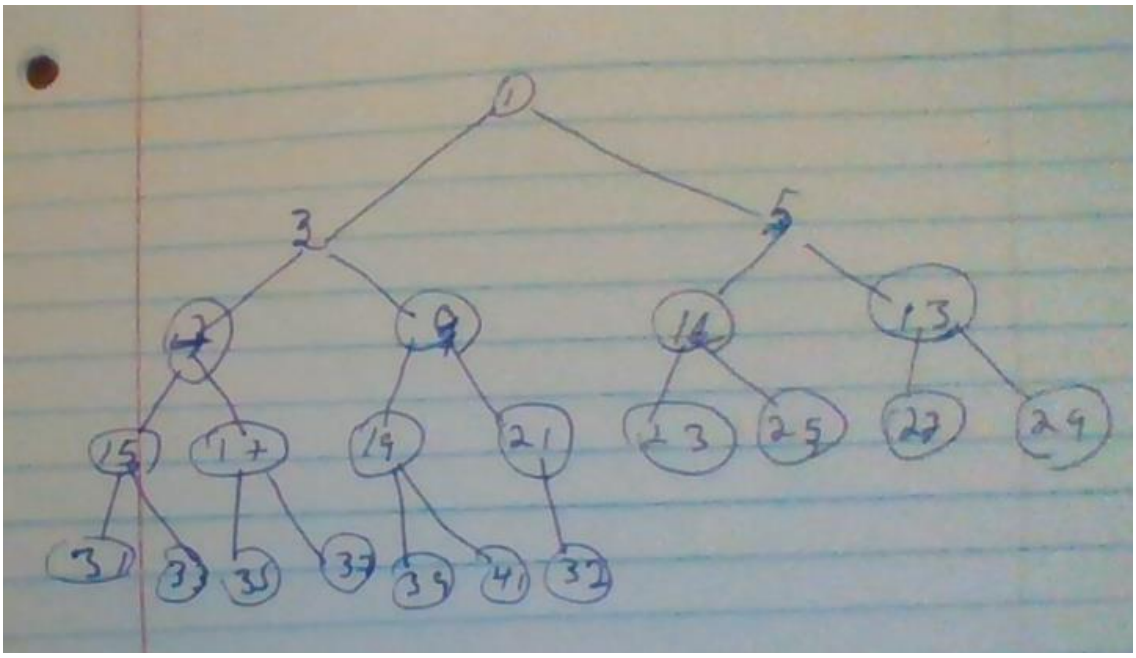
```
function heightOfTree(T) {
    return heightOfTreeHelper(T, T.root())
}
function heightOfTreeHelper(T, p) {
    if (T.isExternal(p)) {
        return 0;
    } else {
        let X = heightOfTreeHelper(T, T.leftChild(p))
        let y = heightOfTreeHelper(T, T.rightChild(p))
    }
    return Math.max(x, y) + 1
}
```

(c) Implement your algorithm using the EulerTour class provided in Tree.js.

2. Suppose a binary tree T is implemented using an array S, as described in the notes. If n items are stored in S in sorted order, starting with index 1, is the tree T a heap? Justify your answer.

Yes, because heap is a sorted binary tree. And the children are greater than parent;

R-2-18 Draw an example of a heap whose keys are all the odd numbers from 1 to 49 (with no duplicates), such that the insertion of an item with key 32 would cause up-heap bubbling to proceed all the way up to a child of the root (replacing that child key of the root with 32).



It is not cause up-heap. Because the parente of 32 is less than 32

C-2.32 Let *T* be a heap storing n keys. Design an efficient recursive pseudo-code algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (note that x is not necessarily in T). For example, given the heap at the bottom slide 35 of the notes and query key x=7, the algorithm should return 7, 5, 4, 6. Note that the keys do not need to be reported in any particular order. Ideally, your algorithm should run in O(k) time, where k is the number of keys reported. Hint: Stop searching beyond a node when the key is greater than x.

Algorithm: findKeys(T,k)

S ← new sequence

Return findKeysHelper(T, k, T.root( ), S)

Alforithm: findKeysHelper(T, k, p, S)

If T.isExternal(p)

Return

If p.element <=k

S.insertItem(p.ele)

findKeysHelper(T, k, T.leftChild(p), S)

findKeysHelper(T, k, T.rigthChild(p), S)