Assignment 10

1. Given a Tree T, write a pseudo code algorithm findDeepestNodes(T), that returns a Sequence of pairs (v, d) where v is an internal node of tree T and d is the depth
   of v in T. The function must return all internal nodes that are at the maximum depth (no other nodes). What is the time complexity of your algorithm?

```
Algorithm findDeepestNodes(T):
    maxDepth ← -1
    result ← new empty Sequence

    dfs(T.root(), 0)

    return result

Algorithm dfs(T, node, depth, maxDepth):

    if depth > maxDepth then
        maxDepth ← depth
        result ← new empty Sequence
        result.add((node, depth))
    else if depth = maxDepth then
        result.add((node, depth))

    for each child in T.children(node) do
        dfs(child, depth + 1)
```

C-4.25 Bob has a set A of n nuts and a set B of n bolts, such that each nut in A has a unique matching bolt in B. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only kind of comparison that Bob can make is to take a nut-bolt pair (a,b), such that a is from A and b is from B, and test it to see if the threads are larger, smaller or a perfect match with the threads of b. Describe an efficient algorithm for Bob to match up all of his nuts and bolts. What is the running time of this algorithm, in terms of nut-bolt tests that Bob must make?

1. Pick a random **pivot nut** `n`.
2. Use `n` to **partition the bolts** into:

- Smaller than `n`
- Matching `n` (store this bolt)
- Larger than `n`

3. Now use the **matched bolt** to partition the nuts in the same way:
    - Smaller than bolt
    - Matching bolt (i.e., pivot nut again)
    - Larger than bolt
4. **Recursively match** the subarrays of smaller and larger nuts/bolts.

```
Algorithm matchNutsAndBolts(nuts, bolts, low, high):
    if low >= high:
        return

    pivotNut ← nuts.atRank(low)

    pivotIndex ← partition(bolts, low, high, pivotNut)

    partition(nuts, low, high, bolts.atRank(pivotIndex))

    matchNutsAndBolts(nuts, bolts, low, pivotIndex - 1)
    matchNutsAndBolts(nuts, bolts, pivotIndex + 1, high)

Function partition(array, low, high, pivot):
    i ← low
```

```
for j from low to high:
    if compare(array.atRank(j), pivot) < 0:
        swap(array.atRank(i), array.atRank(j))
        i ← i + 1
    else if compare(array.atRank(j]), pivot) = 0:
        swap(array.atRank(j), array.atRank(high))
        j ← j - 1  // check swapped item
swap(array.atRank(i), array(high))
return i
```