Jingheng Huan

Problem 1

Algorithm 1 FindIndex(A, n)

```
function FindIndex(A, n)
    low ← 1
    high ← n
    while low ≤ high do
        mid ← ⌊(low + high)/2⌋
        if A[mid] = mid then
            return mid
        else if A[mid] < mid then
            low ← mid + 1
        else
            high ← mid − 1
        end if
    end while
    return No such index exists
end function
```

Explanation of Running Time This algorithm achieves a worst-case running time of $O(\log n)$ by employing a binary search approach that consistently halves the search interval with each iteration. In the worst case, the desired index i (where $A[i] = i$) does not exist or is located at one end of the array, making this algorithm explore all possible sub-arrays by repeatedly dividing the search space. At every step, it compares the middle element A[mid] with its index mid to determine which half of the array may contain the desired index. This systematic reduction of the search space by half in each step ensures that the number of necessary operations grows logarithmically relative to the size of the input array, thereby maintaining an $O(\log n)$ running time in the worst-case scenario.

Problem 2

(a) Greedy Algorithm

Algorithm 2 MaxParty

Require: Tree T with root node r
Ensure: Maximum number of employees invited without inviting any two adjacent ones

```
1: function MaxParty(node)
2:     if node is null then
3:         return (0, false)
4:     end if
5:     total ← 0
6:     childrenSelected ← [ ]
7:     for each child in node.children do
8:         (childTotal, childSelected) ← MaxParty(child)
9:         total ← total + childTotal
10:        append childSelected to childrenSelected
11:    end for
12:    if none of childrenSelected are true then
13:        total ← total + 1
14:        return (total, true)
15:    else
16:        return (total, false)
17:    end if
18: end function
```

Perform a **post-order traversal** of the tree, processing all children of a node before the node itself. For each node during traversal: **Include** the node in the party if none of its immediate children are included. **Exclude** the node if at least one of its immediate children is included. Maintain a set *Selected* to keep track of invited employees. The function `MaxParty` returns a tuple containing the total number of selected employees and a boolean indicating whether the current node is selected.

(b) Worst-Case Asymptotic Runtime

The algorithm performs a single post-order traversal of the tree, visiting each node exactly once and performing a constant amount of work at each node. Time Complexity is O(n), where n is the number of nodes (employees) in the tree.

(c) Proof of Correctness Using Induction

Base Case

Consider a tree with a single node (the root). The algorithm will include this node in *Selected*, which is the correct maximum independent set since there are no other nodes to consider.

Inductive Step

Assume that for all trees with fewer than n nodes, the algorithm correctly finds the maximum independent set. Consider a tree T with n nodes. During the post-order traversal, for each node v, the algorithm makes a decision based on whether any of its immediate children are included in the *Selected* set.

1. **If v is included in the *Selected* set:** None of its immediate children are included, ensuring no two adjacent nodes are selected. By the inductive hypothesis, the subtrees rooted at each child are optimally solved. Therefore, the inclusion of v contributes exactly one to the total, and the rest of the subtree contributes optimally.

2. **If v is excluded from the *Selected* set:** At least one of its immediate children is included. By the inductive hypothesis, the subtrees rooted at each child are optimally solved, allowing for the maximum number of selected nodes without violating the adjacency constraint.

In both scenarios, the algorithm makes a choice that maximizes the number of selected nodes without violating the adjacency constraint. Thus, by induction, the algorithm correctly computes the maximum independent set for any tree.

Collaborators: None