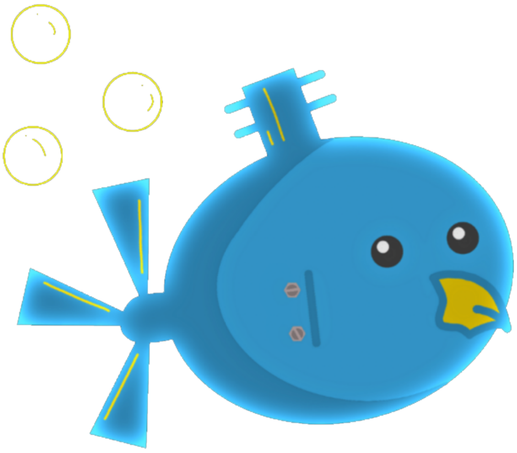
CALIFORNIA STATE UNIVERSITY, LOS ANGELES

Module Level Outcome: *Design, Analysis and Application of Algorithms*



ROBOSUB

Members
Thomas BENSON, David CAMACHO, Bailey CANHAM, Brandon CAO,
Roberto HERNANDEZ, Andrew HEUSSER, Hector MORA-SILVA,
Bart RANDO, Victor SOLIS

March 6, 2023

Contents

1 Question 5: Convert Sorted Array to Binary Search Tree

Problem: https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

1.1 Pseudocode

```
function SortedArrayToBST($nums$)
    if $nums$ is empty then
        return {null}
    end if
    $mid \leftarrow \lfloor \frac{len(nums)}{2} \rfloor$
    $root \leftarrow \{nums[mid]\}$
    $root.left \leftarrow$ SortedArrayToBST(nums[0:mid-1])
    $root.right \leftarrow$ SortedArrayToBST(nums[mid+1:len(nums)])
    return $root$
end function
```

1.2 Code

1.2.1 C

1.2.2 Python

```python
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
        def constructor(left, right):
            if left > right:
                return None

            midpoint = (left + right) // 2
            root = TreeNode(nums[midpoint])
            root.left = constructor(left, midpoint-1)
            root.right = constructor(midpoint+1, right)
            return root
        return constructor(0, len(nums)-1)
```

1.2.3 TypeScript

```typescript
    const sortedArrayToBST = (nums: number[]): TreeNode | null => {
    if(nums.length === 0) {
        return null;
    }
    const mid: number = Math.floor(nums.length / 2);
    const root: TreeNode = new TreeNode(nums[mid]);
    root.left = sortedArrayToBST(nums.slice(0, mid));
    root.right = sortedArrayToBST(nums.slice(mid + 1));
    return root;
};
```

2 Question 6: Binary Tree Preorder Traversal

Problem: https://leetcode.com/problems/binary-tree-preorder-traversal/

2.1 Pseudocode

```
function PREORDERTRAVERSAL(root)
    $stack \leftarrow \{\}$
    $result \leftarrow \{\}$
    if root is null then
        return result
    end if
    PUSH(stack, root)

    while stack is not empty do
        POP(stack, root)
        APPEND(result, root.val)
        if root.right is not null then
            PUSH(stack, root.right)
        end if
        if root.left is not null then
            PUSH(stack, root.left)
        end if
    end while
    return result
end function
```

```
function PREORDERTRAVERSAL(root)
    $result \leftarrow \{\}$
    PREORDER(root, result)
    return result
end function
function PREORDER(root, result)
    if root is null then
        return
    end if
    APPEND(result, root.val)
    PREORDER(root.left, result)
    PREORDER(root.right, result)
end function
```

2.2 Code

2.2.1 C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */


 void preorder(struct TreeNode* root, int* result, int* index) {
    if (root == NULL) return;

    // append
    result[*index] = root->val;
    (*index)++;

    preorder(root->left, result, index);
    preorder(root->right, result, index);
}

int* preorderTraversal(struct TreeNode* root, int* returnSize) {
    int* result = malloc(100 * sizeof(int));
    int index = 0;

    preorder(root, result, &index);
    *returnSize = index;

    return result;
}
```

2.2.2 Java

```
import java.util.*;

class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> ans = new ArrayList<Integer>();

        Stack<TreeNode> toVisit = new Stack<TreeNode>();

        if (root != null) {
            toVisit.push(root);
        }

        while(!toVisit.empty()) {
            TreeNode hold = new TreeNode();
            hold = toVisit.pop();

            ans.add(hold.val);

            if (hold.right != null) {
                toVisit.push(hold.right);
            }
            if (hold.left != null) {
                toVisit.push(hold.left);
            }
        }

        return ans;
    }
}
```

2.2.3 JavaScript

```
const preorderTraversal = (root) => {
    const result = [];
    const stack = [];
    if (root === null) {
        return result;
    }
    stack.push(root);
    while (stack.length > 0) {
        const node = stack.pop();
        result.push(node.val);
        if (node.right !== null) {
            stack.push(node.right);
        }
        if (node.left !== null) {
            stack.push(node.left);
        }
    }
    return result;
};
```