

# Practice Interview

## Objective

\*The partner assignment aims to provide participants with the opportunity to practice coding in an interview context. You will analyze your partner's Assignment 1. Moreover, code reviews are common practice in a software development team. This assignment should give you a taste of the code review process.\*

## Group Size

Each group should have 2 people. You will be assigned a partner

## Part 1:

You and your partner must share each other's Assignment 1 submission.

## Part 2:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

```
In [ ]: # Your answer here
# Involved creating a recursive or looping system to first construct a binary tree using a breadth-first search traversal. Then, the task was to traverse each node, pl
```

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

```
In [ ]: # Your answer here

arr1 = [0,0,1,1, 2, 3, 5, 6, 7,7]
root1 = tree_builder(arr1, 0, len(arr1)) # Build the tree
duplicate_value1 = find_duplicate_closest_to_root(root1) # Find the duplicate closest to root
print("Duplicate value closest to root:", duplicate_value1) # Duplicate value closest to root: 0
```

- Copy the solution your partner wrote.

```
In [ ]: # Definition for a binary tree node.
class TreeNode(object):
    def __init__(self, val = 0, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right

def tree_builder (arr : list[int], i : int, n: int) -> TreeNode:
    if not arr:
        return None
    root = None
    if i < n and arr[i] is not None:
        root = TreeNode(arr[i])
        root.left = tree_builder(arr,2*i+1,n)
        root.right = tree_builder(arr, 2*i+2,n)
    return root

# Your answer here
# 1 Create the set and Stack
def find_duplicate_closest_to_root(root: TreeNode) -> int:
    if root is None:
        return -1

    seen = set()
    stack = [root]
# 2 Add the node to the stack, then check that node in the seen, if present return node value
    while stack:
        node = stack.pop()

        if node.val in seen:
            return node.val
        else:
            seen.add(node.val)

# 3 pushed node right to stack then push left, because stack is last in first out it look at left node first and then the right.
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)

# 4 node in stack will get taken out as they are added to the seen set unless it is present in the set.
    return -1

arr1 = [1, 2, 2, 3, 5, 6, 7]
root1 = tree_builder(arr1, 0, len(arr1))
duplicate_value1 = find_duplicate_closest_to_root(root1)
print("Duplicate value closest to root:", duplicate_value1)

arr2 = [1, 10, 2, 3, 10, 12, 12]
root2 = tree_builder(arr2, 0, len(arr2))
duplicate_value2 = find_duplicate_closest_to_root(root2)
print("Duplicate value closest to root:", duplicate_value2)

arr3 = [10, 9, 8, 7]
root3 = tree_builder(arr3, 0, len(arr3))
duplicate_value3 = find_duplicate_closest_to_root(root3)
print("Duplicate value closest to root:", duplicate_value3)

arr4 = [10, 9, 1, 7,8,2,11,12,13,15,16,18,19,20,20]
root4 = tree_builder(arr4, 0, len(arr4))
duplicate_value4 = find_duplicate_closest_to_root(root4)
print("Duplicate value closest to root:", duplicate_value4)

arr5 = []
root5 = tree_builder(arr5, 0, len(arr5))
duplicate_value5 = find_duplicate_closest_to_root(root5)
print("Duplicate value closest to root:", duplicate_value5)
```

Duplicate value closest to root: 2  
Duplicate value closest to root: 10  
Duplicate value closest to root: -1  
Duplicate value closest to root: 20  
Duplicate value closest to root: -1

- Explain why their solution works in your own words.

```
In [ ]: # As nodes are added to the stack (starting from the root), they are also added to a set.
# I then add the right node and its corresponding left sibling to the stack.
# The left node is added to the set, and both its right and left children are added to the stack.
# Assuming these children are either unique or None, they will be removed from the stack after popping the root's right node and appending its children.
# The helper function traverses each node in a top-down manner (breadth-first search) and adds it to the set.
```

- Explain the problem's time and space complexity in your own words.

```
In [ ]: # This code pushes all left-side nodes onto the stack, then traverses all nodes once from the right.
# This results in a time complexity of O(n).
# In the worst-case scenario where there are no duplicates, all values will be stored in a set, leading to a space complexity of O(n).
```

- Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

```
In [ ]: # 1. TreeNode Class Definition
# The TreeNode class is properly defined and doesn't need any changes.
# 2. Tree Builder Function
# The function tree_builder is well-implemented but can be simplified slightly. The explicit check if not arr is unnecessary since the base case if i >= n already handles it.
# Return None at the end is missing.
# 3. find_duplicate_closest_to_root Function
# The logic is correct. However, comments can be improved for better understanding.
# An additional check can be added to ensure the function handles invalid input gracefully.
# 4. Testing Code
# The testing code is correct but can be wrapped in a function for reusability.
```

## Part 3:

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

## Reflection

```
In [ ]: # In this assignment, I focused on reviewing and refining a solution designed to build a binary tree from an array
# and identify the first duplicate value closest to the root. The process began with a detailed examination of the provided code,
# which involved understanding its logic, structure, and potential shortcomings. I ensured the TreeNode class and tree-building
# logic were correctly implemented and then analyzed the find_duplicate_closest_to_root function for efficiency and clarity.

# The critique emphasized improving comments for better comprehension, handling edge cases, and simplifying code where possible.
# For instance, unnecessary checks could be removed from the tree_builder function, and a missing return statement
# should be added to ensure proper functionality.



# Presenting the review involved outlining the logical flow and the rationale behind each suggested adjustment.
# This approach helped ensure the revised code was not only correct but also easy to understand and maintain.
# Overall, this exercise reinforced the importance of clear documentation, edge case handling, and code readability in software development.
```

## Evaluation Criteria

We are looking for the similar points as Assignment 1

- Problem is accurately stated
- New example is correct and easily understandable
- Correctness, time, and space complexity of the coding solution
- Clarity in explaining why the solution works, its time and space complexity
- Quality of critique of your partner's assignment, if necessary

## Submission Information

 Please review our [Assignment Submission Guide](#)  for detailed instructions on how to format, branch, and submit your work. Following these guidelines is crucial for your submissions to be evaluated correctly.

### Submission Parameters:

- Submission Due Date: HH:MM AM/PM – DD/MM/YYYY
- The branch name for your repo should be: assignment-2
- What to submit for this assignment:
  - This Jupyter Notebook (assignment\_2.ipynb) should be populated and should be the only change in your pull request.
- What the pull request link should look like for this assignment: [https://github.com/<your\\_github\\_username>/algorithms\\_and\\_data\\_structures/pull/<pr\\_id>](https://github.com/<your_github_username>/algorithms_and_data_structures/pull/<pr_id>)
  - Open a private window in your browser. Copy and paste the link to your pull request into the address bar. Make sure you can see your pull request properly. This helps the technical facilitator and learning support staff review your submission easily.

Checklist:

- ☐ Created a branch with the correct naming convention.
- ☐ Ensured that the repository is public.
- ☐ Reviewed the PR description guidelines and adhered to them.
- ☐ Verify that the link is accessible in a private browser window.

If you encounter any difficulties or have questions, please don't hesitate to reach out to our team via our Slack at #cohort-3-help . Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.