Copilot

Here is a summary of part 1 of the current page:

- **Course Overview**: The ITMO Academy pilot course covers various topics in competitive programming, including Suffix Array, Segment Tree, Binary Search, Disjoint Sets Union, and Two Pointers Method.
- **Course Structure**: Each topic is divided into multiple steps, with each step containing theory and practice sections.
- **Recent Actions**: Displays recent activities of users enrolled in the course, such as which topics they have accessed or practiced.

Would you like more details on any specific section?

Copilot

Here is a summary of part 2 of the current page:

- **Hints and Solutions**: Users are discussing hints and solutions for various problems, including implementation steps and debugging tips.
- **Common Issues**: Frequent issues like Time Limit Exceeded (TLE) and Wrong Answer (WA) are mentioned, with users seeking help to resolve them.
- **Code Sharing**: Users share their code snippets and ask for feedback to identify errors and improve their solutions.
- **Problem-Specific Discussions**: Detailed discussions on specific problems, such as "Minimum Average Paths" and "Children Holiday," with users providing insights and strategies.

Feel free to ask if you need more details on any specific topic!

Copilot

Here are the key points from part 3 of the current page:

- **Binary Search Implementation**: The code snippet demonstrates a binary search algorithm to find the minimum value that satisfies certain conditions.
- **Common Issues**: Users discuss problems like wrong answers (WA) and time limit exceeded (TLE) errors in their code submissions.
- **Hints and Solutions**: Community members provide hints and solutions to help others debug and optimize their code.
- **Code Sharing**: Users share their code snippets and ask for feedback to resolve specific issues in their implementations.

Copilot

Certainly! Let's consolidate the information about binary search trees (BSTs) and provide example code in C, Java, and Python.

Binary Search Trees (BSTs)

A Binary Search Tree (BST) is a binary tree where the values of the left subtree are less than the root node, and the values of the right subtree are greater than the value of the root node. Here are some key properties of BSTs:

1. **Node Structure**:
    - Each node has at most two children: a left child and a right child.
    - The left subtree contains values less than the root, and the right subtree contains values greater than the root.
2. **Basic Operations**:
    - **Insertion**: Add a new node while maintaining the BST property.
    - **Searching**: Find a specific value in the tree.
    - **Deletion**: Remove a node while preserving the BST structure.

Now, let's see example code for BSTs in C, Java, and Python:

C (Binary Search Tree Implementation)

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int key;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->key = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);
    if (value < root->key)
        root->left = insert(root->left, value);
    else if (value > root->key)
        root->right = insert(root->right, value);
    return root;
}
```

```
// Other operations (search, delete, traversal) can be implemented similarly.
```

Java (Binary Search Tree Implementation)

```java
class TreeNode {
    int key;
    TreeNode left;
    TreeNode right;

    public TreeNode(int value) {
        key = value;
        left = right = null;
    }
}

class BinarySearchTree {
    TreeNode root;

    void insert(int value) {
        root = insertRec(root, value);
    }

    TreeNode insertRec(TreeNode root, int value) {
        if (root == null)
            return new TreeNode(value);
        if (value < root.key)
            root.left = insertRec(root.left, value);
        else if (value > root.key)
            root.right = insertRec(root.right, value);
        return root;
    }

    // Other operations (search, delete, traversal) can be implemented similarly.
}
```

Python (Binary Search Tree Implementation)

```python
class TreeNode:
    def __init__(self, value):
        self.key = value
        self.left = None
        self.right = None

def insert(root, value):
    if root is None:
        return TreeNode(value)
    if value < root.key:
        root.left = insert(root.left, value)
    elif value > root.key:
        root.right = insert(root.right, value)
    return root

# Other operations (search, delete, traversal) can be implemented similarly.
```

Feel free to adapt and extend these examples based on your specific requirements! 😊