```
<script src="https://cdn.anychart.com/releases/8.12.0/js/anychart-core.min.js"></script
<script src="https://cdn.anychart.com/releases/8.12.0/js/anychart-graph.m
</script> <script src="https://cdn.anychart.com/releases/8.12.0/js/anychart
</script> <link rel="stylesheet" type="text/css"
href="https://cdn.anychart.com/releases/8.12.0/css/anychart-ui.min.css?
hcode=a0c21fc77e1449cc86299c5faa067dc4"/> <link rel="stylesheet" type="text/css"
href="https://cdn.anychart.com/releases/8.12.0/fonts/css/anychart-font.min.css"/>
```

Tutorial 5: Binary Search Tree

CAB301 - Algorithms and Complexity

School of Computer Science, Faculty of Science

Agenda

1. **Lecture Recap**: Binary Tree and Binary Search Tree
    - Binary Tree **Traversal**
    - Binary Search Tree Operations:
        - **Search**
        - **Insertion**
        - **Deletion**
2. **Tutorial Questions** + **Q&A**

Binary Tree

<script src="./themes/chart.js"></script> <div class="flexbox"> <div style="flex: 0.5">

Each node has at most **two children**: left and right.

</div> <div id="container" style="flex: 0.5; border: 1px solid black;"></div> </div>

Binary Search Tree

<div class="flexbox"> <div style="flex: 0.5">

In a **Binary Search Tree (BST)**, for each node:

- All nodes in the left subtree have **smaller** values.
- All nodes in the right subtree have **greater** values.

Any operation must maintain the BST property.

</div> <div id="b-tree-container" style="flex: 0.5; border: 1px solid black;"></div> </div>

Binary Tree Traversal

<div class="flexbox"> <div style="flex: 0.5">

In-order: Left, Root, Right

<button id="in-order">Run</button>

Pre-order: Root, Left, Right

<button id="pre-order">Run</button>

Post-order: Left, Right, Root

<button id="post-order">Run</button>

<input id="traversal-slider" type="range" min="1" max="5" value="1" step="1" style="width: 100%; margin-top: 10px;"> </div> <div id="b-tree-container-2" style="flex: 0.5; border: 1px solid black;"></div> </div>

Search Operation

<div class="flexbox"> <div style="flex: 0.5">

Search is similar to **Binary Search**:

<div style="font-size: 20px">

```
ALGORITHM Search($K$, $root$)
    if $root \neq null$
        if $root.item = K$
            return $true$
        else if $root.item > K$
            return Search($K$, $root.left$)
        else
            return Search($K$, $root.right$)
```

Insertion Operation

<div class="flexbox"> <div style="flex: 0.5">

Insertion is similar to **Search**:

<div style="font-size: 14px">

```
ALGORITHM Insert($K$, $root$)
    if $root = null$
        $ptr \leftarrow$ new Node; $ptr.item \leftarrow K$; $root \leftarrow ptr$
    else
        if $root.item > K$
            if $root.left = null$
                $ptr \leftarrow$ new Node
                $ptr.item \leftarrow K$
```

Deletion Operation

<div class="flexbox"> <div style="flex: 0.5">

Deletion is more complex:

- **Case 1**: Node has no children (leaf node). Simply remove it.
- **Case 2**: Node has one child (left or right). Replace it with the child.
- **Case 3**: Node has two children (left and right)

<div style="display: flex; margin-bottom: -30px;"> <input id="delete-value" type="number" value="0" style="width: 100%; height: 100%"> <button id="delete-button">Delete</button> </div> <input id="delete-slider" type="range" min="1" max="5" value="1" step="1" style="width: 100%; margin-top: 10px;"> </div> <div id="b-tree-container-5" style="flex: