HIT220 - Group Assignment 3.2

Assignment Overview: This assignment assesses your understanding of search trees and sorting algorithms. You will implement specific algorithms, analyze their performance, and compile a detailed report on your findings.

1. Tasks and Deliverables:

1） Binary Search Tree (BST) (5 marks)

Tasks: Implement a Binary Search Tree with the following functionalities:

1. Insert the values: 50, 30, 70, 20, 40, 60, 80 sequentially.
2. Delete the value 70 from the BST.
3. Search for the value 20 in the BST.
4. After each operation, provide the tree structure and show the in-order traversal result.

Deliverables:

1. Python file (`bst.py`) with the implementation of the BST and its functions.
2. A brief report (Word or PDF) showing the tree structure and in-order traversal after each operation.

2） Sorting Algorithms (3 marks)

Tasks:

1. Sort the array `[64, 34, 25, 12, 22, 11, 90]` using the following algorithms:

   i. Insertion Sort
   ii. Bubble Sort
   iii. Selection Sort

2. Provide the time complexity for each algorithm, including best-case, worst-case, and average-case scenarios.
3. Provide the space complexity for each algorithm, detailing the additional space used.

Deliverables:

1. Python file (`sortings.py`) with implementations of Insertion Sort, Bubble Sort, and Selection Sort.
2. Report with a detailed time and space complexity analysis for each sorting algorithm.

3) Divide-and-Conquer Sorting (7 marks)

Tasks:

1. Implement and sort the array `[38, 27, 43, 3, 9, 82, 10]` using the following algorithms:

   i. Merge Sort
   ii. Quick Sort

2. Show intermediate steps of sorting for each algorithm.
3. Test all implemented sorting algorithms (Insertion, Bubble, Selection, Merge, Quick Sort) on an array of 1000 randomly generated integers.
4. Measure and record the time taken for each algorithm to sort the array.

Deliverables:

1. Python files (`divide_conquer_sorting.py` and `sorting_1000.py`) with implementations for tasks 1 and 3.
2. Report showing the array at each significant step of sorting.
3. Report comparing the performance of each sorting algorithm, including recorded times and observations.

2. Submission Instructions:

- Submit the following documents and code as **individual files; do not submit a zip file:**
  - A report (Word or PDF) containing the results for all three questions (make sure to clearly label each question).
  - Python files: `bst.py`, `sortings.py`, `divide_conquer_sorting.py`, `sorting_1000.py`.

3. Marking Rubric:

- Correctness of Implementations (40%)
- Clear Explanation and Accurate Analysis (30%)
- Performance and Comparison (20%)
- Clarity of Code and Reports (10%)