|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Science and Computer Engineering** |  | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم و هندسة الحاسب** | |  |  |

**CCCS 314 – Design and Analysis of Algorithms**

**LAB 6 (Exercise 1-3)**

**Topics:**

1. **Transform and Conquer (Pre-sorting, AVL Trees, Multi-way Trees)**

**Total Marks: 2**

**Student Name: Bassam Alghamdi**

**Student ID: 2141362**

**Marks:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exercises | 1 | 2 | 3 | Total |
| Allocated | 1 | 0.5 | 0.5 | 2 |
| Obtained |  |  |  |  |
| **CLO, PLO** | 2.1, S1 | 2.1, S1 | 2.1, S1 |  |

**CLO** **Marks:**

|  |  |  |
| --- | --- | --- |
|  | CLO2.1, S1 | Total |
| Allocated | 2 | 2 |
| Obtained |  |  |

**Exercise 1: Pre-sorting**

Consider the problem of finding the distance between the two closest numbers in an array of n numbers. (The distance between two numbers x and y is computed as |x − y|.)

1. Design a presorting-based algorithm and implement it in your preferred language.

def closest\_distance\_pre\_sorting(arr):

arr.sort() # Sort the array in ascending order

n = len(arr)

min\_distance = float('inf') # Initialize minDistance to positive infinity

for i in range(1, n):

distance = abs(arr[i] - arr[i-1]) # Calculate the distance between current and previous elements

if distance < min\_distance:

min\_distance = distance

return min\_distance

1. Solve the above problem using brute-force approach. Implement in your preferred language.

def closest\_distance\_brute\_force(arr):

n = len(arr)

min\_distance = float('inf') # Initialize minDistance to positive infinity

for i in range(n):

for j in range(i+1, n):

distance = abs(arr[i] - arr[j]) # Calculate the distance between the current pair of numbers

if distance < min\_distance:

min\_distance = distance

return min\_distance

1. Compare the efficiency of the above two algorithms.

The pre-sorting algorithm has a time complexity of O(n log n) because it involves sorting the array before finding the closest distance. The sorting step takes O(n log n) time, and the subsequent iteration takes O(n) time.

On the other hand, the brute-force algorithm has a time complexity of O(n^2) because it checks all possible pairs of numbers in the array. The nested loop results in quadratic time complexity.

Therefore, the pre-sorting algorithm is more efficient than the brute-force algorithm for large input sizes.

**Exercise 2: AVL Trees**

Construct AVL trees by inserting the elements successively, starting with the empty tree.

1. 1, 2, 3, 4, 5, 6

Step 1: Insert 1

1

Step 2: Insert 2

2

/

1

Step 3: Insert 3

2

/ \

1 3

Step 4: Insert 4

2

/ \

1 3

\

4

Step 5: Insert 5

2

/ \

1 4

/ \

3 5

Step 6: Insert 6

4

/ \

2 5

/ \ \

1 3 6

1. 6, 5, 4, 3, 2, 1

Step 1: Insert 6

6

Step 2: Insert 5

6

/

5

Step 3: Insert 4

6

/

5

/

4

Step 4: Insert 3

5

/ \

4 6

/

3

Step 5: Insert 2

5

/ \

4 6

/

3

\

2

Step 6: Insert 1

5

/ \

4 6

/

3

\

2

\

1

1. 3, 6, 5, 1, 2, 4

Step 1: Insert 3

3

Step 2: Insert 6

3

\

6

Step 3: Insert 5

3

\

6

/

5

Step 4: Insert 1

3

\

6

/

5

/

1

Step 5: Insert 2

3

\

6

/

2

/ \

1 5

Step 6: Insert 4

3

\

6

/

2

/ \

1 5

\

4

**Exercise 3: Multi-way Trees**

A way to reduce the height of tree and ensure balance is to allow multiple children of nodes. In your class you learned 2-3 trees which allows up to 2 keys in a node, and the number of children is equal to the number of keys + 1. B-trees extend this concept to any arbitrary number of keys (usually number of keys is even and number of children (equal to number of keys+1) is odd).

Assume we want to design a 5-way B-Tree. This will mean that there can be maximum 4 keys in a node, and if the number of keys becomes 5, we can split it into two (the same way we split 2-3 tree when number of keys becomes 3).

Design a 5-way B-tree. Starting with an empty tree, insert the following keys in order.  
2, 3, 5, 7, 10, 50, 22, 44, 45, 55, 66, 68, 70, 17, 6, 21, 67

Step 1: Insert 2 (Root node)

2

Step 2: Insert 3 (Node 2 becomes full, split)

2, 3

Step 3: Insert 5

2, 3, 5

Step 4: Insert 7

2, 3, 5, 7

Step 5: Insert 10

2, 3, 5, 7, 10

Step 6: Insert 50 (Splitting root node)

5

/ \

2 7

/ \

3 10

\

50

Step 7: Insert 22

5

/ \

2 7

/ \

3 10

/ \

22 50

Step 8: Insert 44

5

/ \

2 7

/ \

3 10

/ \ \

22 44 50

Step 9: Insert 45

5

/ \

2 7

/ | | \

3 10 44 50

/ \

22 45

Step 10: Insert 55

5 44

/ \ / \

2 7 10 50

/ | | | | | \

3 55 45 22 68 70

Step 11: Insert 66

5 44

/ \ / \

2 7 10 50

/ | | | | | \

3 55 45 22 66 70

| |

68 67

Step 12: Insert 68 (Splitting leaf node)

5 44

/ \ / \

2 7 10 50

/ | | | | | \

3 55 45 22 66 68

| |

67 70

Step 13: Insert 70 (Splitting root node)

44 66

/ | | \

5 10 50 68

/ \ | | | | \

2 3 7 22 45 67 70

| |

55 66

Step 14: Insert 17

44 66

/ | | \

5 10 50 68

/ \ | | | | \

2 3 7 17 45 67 70

| |

55 66

Step 15: Insert 6

44 66

/ | | \

5 10 50 68

/ \ | | | | \

2 3 6 17 45 67 70

| |

55 66

Step 16: Insert 21

44 66

/ | | \

5 10 50 68

/ \ | | | | \

2 3 6 17 21 67 70

| |

55 66

Step 17: Insert 67 (Splitting leaf node)

44 66

/ | | \

5 10 50 68

/ \ | | | | \

2 3 6 17 21 67 70

| | |

55 66 68