IMPLEMENATION AND COMPARISION OF DIFFERENT SEARCHING TECHNIQUES

BY

VENKATA NAGENDAR SHANTISWAROOP ADIBHATLA

**TABLE OF CONTENTS**

**I. CHAPTER 1. INTRODUCTION**

* 1. What is Searching.
  2. Project Perspective

II. **CHAPTER 2. LINEAR SEARCH**

2.1. Linear Search

2.2. Code implementation of algorithm

III. **CHAPTER 3. BINARY SEARCH**

3.1 Binary Search

3.2 Code implementation of algorithm

IV. **CHAPTER 4. BINARY SEARCH TREE**

4.1 Binary Search Tree

4.2 Code implementation of algorithm

V. **CHAPTER 5. RED-BLACK TREE**

5.1 Red-black Tree

5.2 Code implementation of algorithm

VI. **CHAPTER 6. COMPARISION**

6.1 Run Time complexity Table

6.2 Graph representing run time complexities using random inputs

VII. **CHAPTER 7.GUI and SAMPLE OUTPUTS**

7.1 GUI

7.2 OUTPUT

7.3 SAMPLE RUNS

## **INTRODCUTION**

**1.1 What is Searching**

In computer science, searching algorithm is an algorithm which solves a search problem. Search algorithms work to retrieve information stored within some data structure.

To search a key in a data set which consists of huge amount of data (Example:10000 records) it is tough to element manually. Hence searching algorithms are used to search a particular key using various methodologies.

**1.2 Project Perspective**

The perspective of the project is to help to understand the runtimes of 4 search algorithms/techniques based on the inputs provided.

**CHAPTER -2**

## **LINEAR SEARCH**

**2.1 Linear Search**

In computer science, a linear search or sequential search is a method for finding an element in the list. It sequentially checks each element of the list until a match is found.

**2.2 Code implementation of algorithm**

A simple implementation of linear search looks like below. Function linear\_search does the search.

Graphical user interface, text, application

Description automatically generated

In this implementation, the loop iterates from starting index and returns if the element is found or not.

**CHAPTER-3**

**BINARY SEARCH**

**3.1 Binary Search**

In computer science, binary search also known as half-interval search, logarithmic search or binary chop is a searched algorithm that finds the position of target value within a sorted array.

**2.2 Code implementation of algorithm**

A simple implementation of binary search looks like below. Function binary\_search\_sort does the search

Graphical user interface, text

Description automatically generated

Here the functions calculated mid value and comparing the key with mid it either finds from mid+1 to high or low to mid-1.

**CHAPTER-4**

**BINARY SEARCH TREE**

**4.1 Binary Search** **Tree**

In computer science, Binary search tree also called as ordered or sorted binary tree is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node’s right subtree and less than those in left subtree.

**4.2 Code Implementation of Algorithm**

Text

Description automatically generated with medium confidence

Here the function search\_element runs recursively by comparing the key to root and then finding it left subtree or right sub tree.

**CHAPTER-5**

**RED-BLACK TREE**

**5.1 Red-Black** **Tree**

In computer science, a red-black tree is a self-balancing binary search tree. Each node stores an extra bit representing color, used to ensure that the tree remains balanced during insertion and deletion.

**Insertion operation:**

**Graphical user interface, application

Description automatically generated**

**Balancing Tree:**

**Text

Description automatically generated with medium confidence**

**Rotation & Search Operation**:

A picture containing table

Description automatically generated

Here the function insert\_value is to read inputs from the GUI, function balance\_tree balances the tree by rotating and coloring. Function search\_tree is used to search the key.

**CHAPTER-6**

**COMPARISION**

**6.1 Run time complexity Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best-case** | **Average-case** | **Worst-case** |
| **Linear search** | Ω(1) | Θ(n/2) | O(n) |
| **Binary search** | Ω(1) | Θ(log n) | O(log n) |
| **Binary search tree** | Ω(1) | Θ(log n) | O(n) |
| **Red-Black tree** | Ω(1) | Θ(log n) | O(log n) |

**6.2 Graph representing the run-time complexities using sample inputs**

**Chart, line chart

Description automatically generated**

**CHAPTER-7**

**GUI AND SAMPLE TESTS**

**7.1 GUI:**

* GUI is made using tkinter an inbuilt package of python3.
* Radio button to select desired searching algorithm.
* Check box to select desired algorithms to compare.

**7.2 GUI INTERFACE:**

The interface allows inputs of search algorithm with radio button and also comparison of run-times with desired algorithms using checkbox.

Graphical user interface, text, application

Description automatically generated**7.3 SAMPLE TEST:**

A sample test run of element searching binary search tree

Graphical user interface, text, application, email

Description automatically generated

Output at console of BST search.

Graphical user interface, text, application, email

Description automatically generated

Comparing with other search algorithms

