

CSE 5331 - DAA

PROJECT REPORT

SUBMITTED BY:

ASFIYA MISBA (1002028239)

Project 2 - Search Algorithms

- Linear Search
- Binary Search
- Binary Search Tree
- Red-Black Tree

LINEAR SEARCH

In linear search, we compare the element to be searched with all the elements of the array sequentially.

Time Complexity: $O(n)$, where n is the size of the array to be searched.

Best case is when the search element is the first element of the array, $O(1)$.

Best Case	Average Case	Worst Case
$O(1)$	$O(n)$	$O(n)$

Steps:

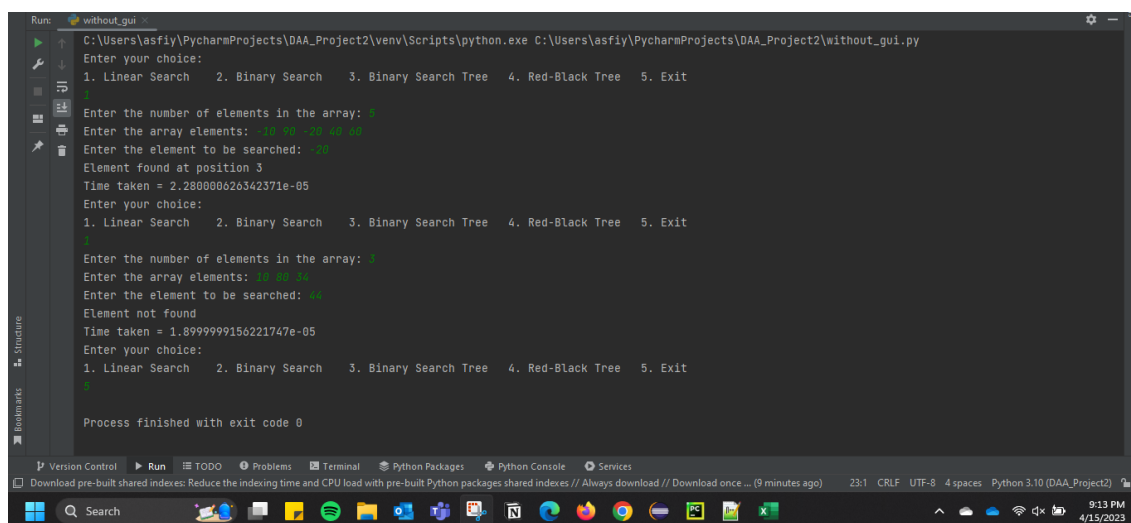
- First, read the search element (Target element) in the array.
- Set an integer $i = 0$ and repeat steps 3 to 4 till i reaches the end of the array.
- Match the key with $arr[i]$.
- If the key matches, return the index. Otherwise, increment i by 1 and continue.
- Finally, if key does not match return -1

Data Structures & Components:

An array is used which is traversed sequentially until the element is found or till the end of the array. In the project, `linearSearch(arr, x)` method is used to implement this algorithm.

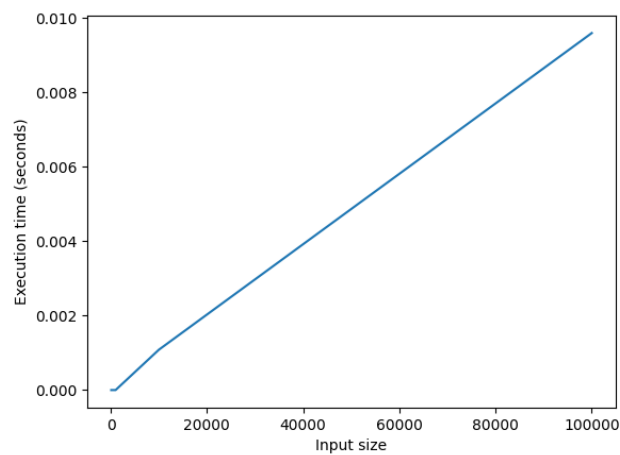
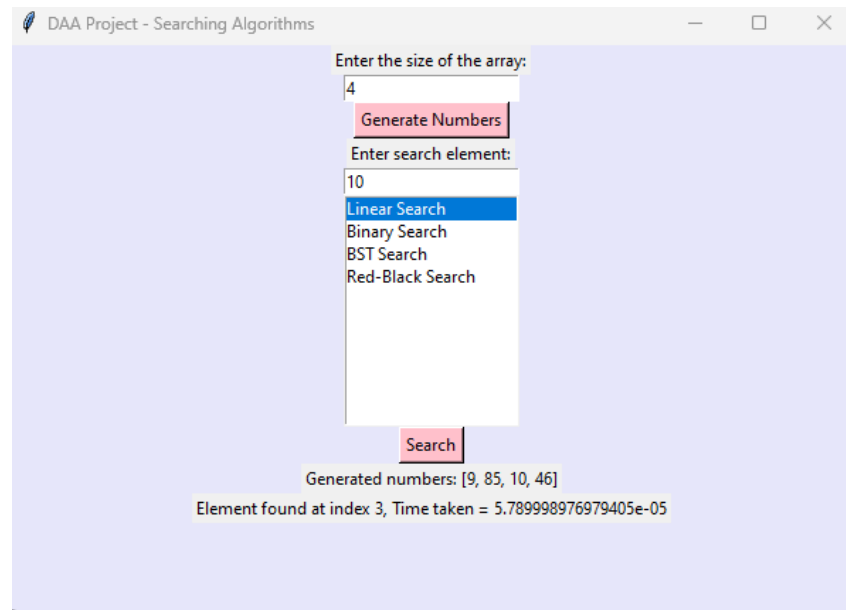
OUTPUT:

Without GUI:



```
Run without_gui
C:\Users\asfiy\PycharmProjects\DAA_Project2\venv\Scripts\python.exe C:\Users\asfiy\PycharmProjects\DAA_Project2\without_gui.py
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
1
Enter the number of elements in the array: 3
Enter the array elements: 10 20 30
Enter the element to be searched: 20
Element found at position 3
Time taken = 2.280000626342371e-05
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
1
Enter the number of elements in the array: 3
Enter the array elements: 10 20 30
Enter the element to be searched: 40
Element not found
Time taken = 1.8999999156221747e-05
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
5
Process finished with exit code 0
```

With GUI:



This search is easy to implement and is effective when the array contains only a few elements.

BINARY SEARCH

In binary search, we search for an element from a sorted array by repeatedly dividing the search interval in half.

Time Complexity: $O(\log n)$, where n is the size of the array to be searched.

Best case is when the search element is the middle element, $O(1)$.

Best Case	Average Case	Worst Case
$O(1)$	$O(\log n)$	$O(\log n)$

Steps:

- Sort the array in ascending order.
- Set the low index to the first element of the array and the high index to the last element.
- Set the middle index to the average of the low and high indices.
- If the element at the middle index is the target element, return the middle index.
- If the target element is less than the element at the middle index, set the high index to the middle index $- 1$.
- If the target element is greater than the element at the middle index, set the low index to the middle index $+ 1$.
- Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array.

Data Structures & Components:

An array is used which is divided into halves and the search is continued until the array can no longer be divided or the element is found. In the project, `binarySearch(arr, x)` method is used to implement this algorithm.

OUTPUT:

Without GUI:

```
Run: without_gui x
C:\Users\asfiy\PchamProjects\DAA_Project2\venv\Scripts\python.exe C:\Users\asfiy\PchamProjects\DAA_Project2\without_gui.py
Enter your choice:
1. Linear Search  2. Binary Search  3. Binary Search Tree  4. Red-Black Tree  5. Exit
2
Enter the number of elements in the array: 4
Enter the array elements in a sorted order: 10 20 30 40
Enter the element to be searched: 30
Element found at position 3
Time taken = 1.8600003386382014e-05
Enter your choice:
1. Linear Search  2. Binary Search  3. Binary Search Tree  4. Red-Black Tree  5. Exit
2
Enter the number of elements in the array: 4
Enter the array elements in a sorted order: 10 20 30 40
Enter the element to be searched: 45
Element not found
Time taken = 1.5300000086426735e-05
Enter your choice:
1. Linear Search  2. Binary Search  3. Binary Search Tree  4. Red-Black Tree  5. Exit
5
Process finished with exit code 0
```

With GUI:

DAA Project - Searching Algorithms

Enter the size of the array:
5

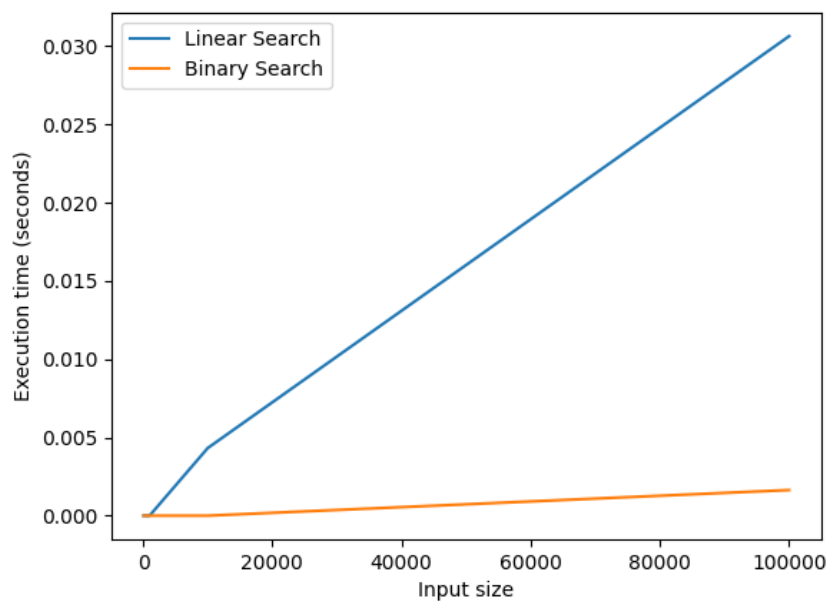
Generate Numbers

Enter search element:
100

Linear Search
Binary Search
BST Search
Red-Black Search

Search

Generated numbers: [73, 74, 79, 93, 17]
Element not found, Time taken = 1.4599994756281376e-05



From the graph, we can observe that binary search performs better than linear search. One drawback of using binary search is the array has to be first sorted before the search begins.

BINARY SEARCH TREE

A binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
 - The right subtree of a node contains only nodes with keys greater than the node's key.
 - The left and right subtree each must also be a binary search tree.
- There must be no duplicate nodes.

For searching an element, we start at the root, and then we compare the value to be searched with the value of the root, if it's equal we are done with the search if it's smaller we continue the search in the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger. Otherwise, we continue the search in the right subtree.

Time Complexity: $O(h)$, where h is the height of the tree.

Height of the tree is given by $\log n$.

Best Case	Average Case	Worst Case
$O(\log n)$	$O(\log n)$	$O(n)$

Data Structures & Components:

class BSTNode is used to define the node of the BST.

class BST consists of insert() and BST_Search().

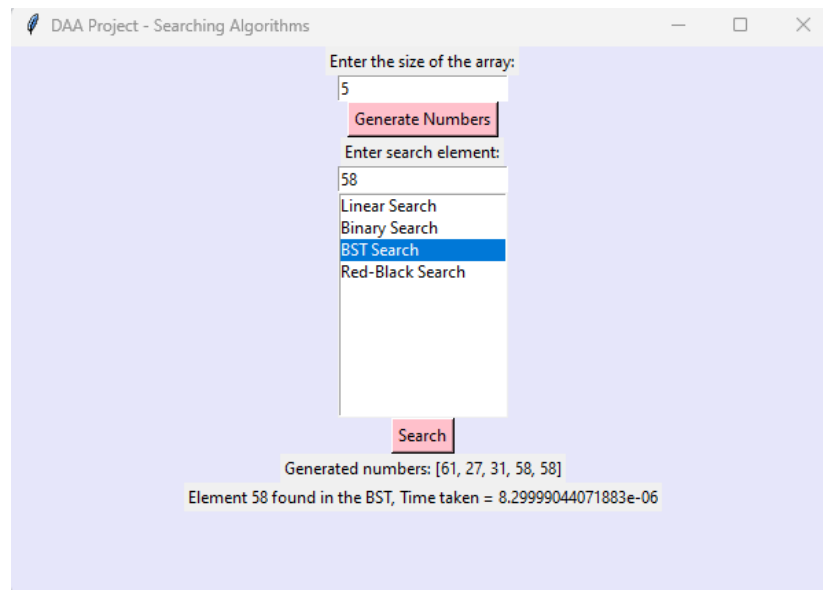
insert() method is used to insert the elements into the BST after which the search is done.

OUTPUT:

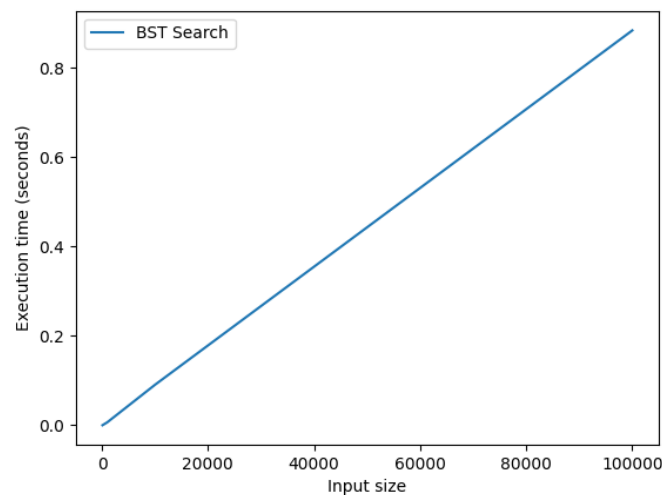
Without GUI:

```
Run: without_gui.py
C:\Users\asfiy\PcharmProjects\DAA_Project2\venv\Scripts\python.exe C:\Users\asfiy\PcharmProjects\DAA_Project2\without_gui.py
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
3
Enter the number of elements in the BST: 4
Enter the elements of the BST : 10 20 30 40
Enter the element to be searched: 30
Element 30 found in the BST
Time taken = 8.000002708286047e-06
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
3
Enter the number of elements in the BST: 4
Enter the elements of the BST : 10 20 30 40
Enter the element to be searched: 100
Element 100 not found in the BST
Time taken = 1.0700001439545304e-05
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
5
Process finished with exit code 0
```

With GUI:



BST search is effective in situations where the elements can be compared in a less than / greater than manner. The performance of the BST depends on the shape of the tree generated. A skewed tree will give the worst case time complexity. To overcome this, we can use red-black trees as they are self-balancing.



RED-BLACK TREE

Red-black tree is a self-balancing BST with the following properties:

- Every node has a colour either red or black.
- The root of the tree is always black.
- There are no two adjacent red nodes (A red node cannot have a red parent or red child).
- Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.
- Every leaf (e.i. NULL node) must be coloured BLACK.

Balancing is done by performing rotations. Searching is similar to BST search, we start at the root, and then we compare the value to be searched with the value of the root, if it's equal we are done with the search if it's smaller we continue the search in the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger. Otherwise, we continue the search in the right subtree.

Time Complexity: $O(\log n)$, where n is the number of elements.

Best Case	Average Case	Worst Case
-	$O(n)$	$O(n)$

Data Structures & Components:

class RedBlackNode is used to define the node of the red-black tree.

class RedBlack consists of the following methods:

- insert() - To insert the elements into the red-black tree.
- balance() - To balance the red-black tree by performing rotations.
- left_rotate() - To perform left rotation.
- right_rotate() - To perform right rotation.
- RedBlackSearch() - To perform the search operation.

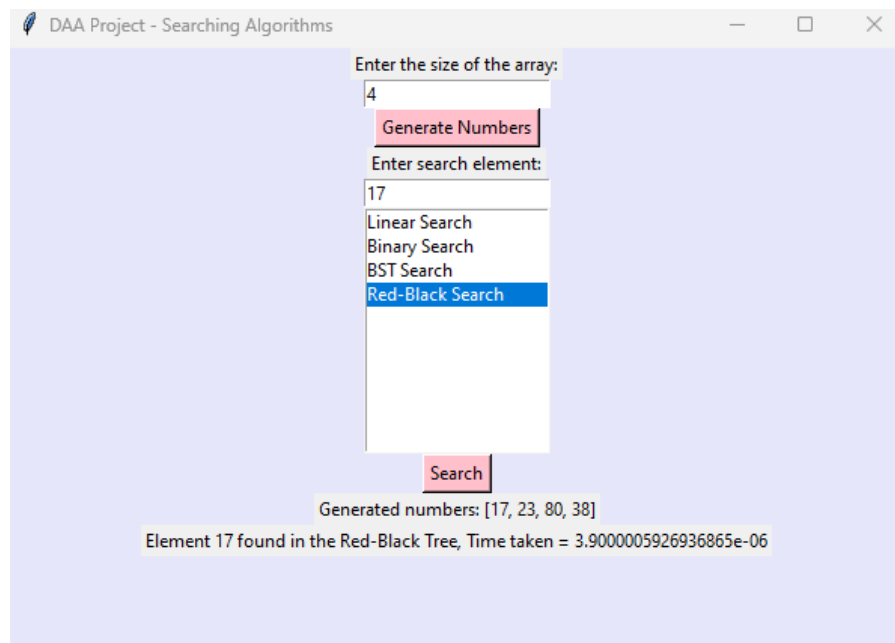
OUTPUT:

Without GUI:

```
Run: without_gui.py
C:\Users\asfiy\PchamProjects\DAA_Project2\venv\Scripts\python.exe C:\Users\asfiy\PchamProjects\DAA_Project2\without_gui.py
Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
> 4
Enter the number of elements in the red-black tree: 5
> 5
Enter the elements of the red-black tree: 20 10 30 40 50
> 
Enter the element to be searched: 28
> 
Element 28 found in the red-black tree
Time taken = 5.999994755256921e-06

Enter your choice:
1. Linear Search 2. Binary Search 3. Binary Search Tree 4. Red-Black Tree 5. Exit
> 4
Enter the number of elements in the red-black tree: 5
> 5
Enter the elements of the red-black tree: 20 10 30 40 50
> 
Enter the element to be searched: 200
> 
Element 200 not found in the red-black tree
Time taken = 3.6000055843032897e-06
```


With GUI:



DAA Project - Searching Algorithms

Enter the size of the array:
4

Generate Numbers

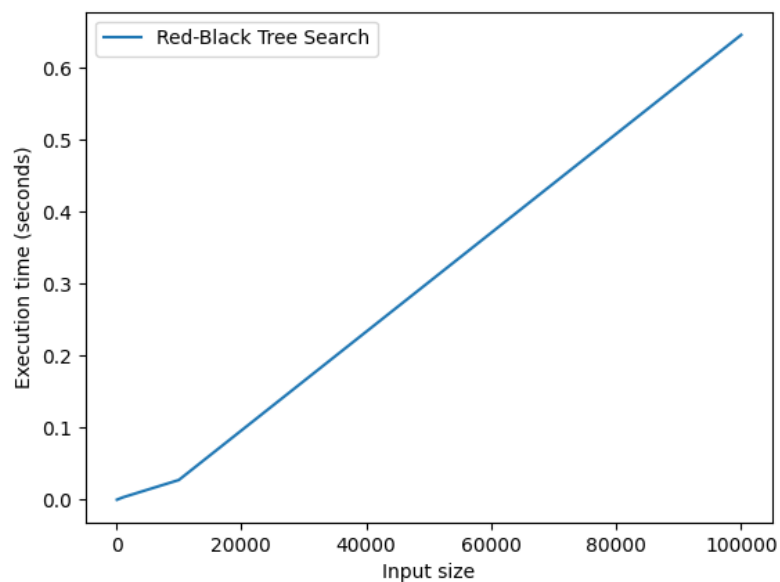
Enter search element:
17

Linear Search
Binary Search
BST Search
Red-Black Search

Search

Generated numbers: [17, 23, 80, 38]
Element 17 found in the Red-Black Tree, Time taken = 3.9000005926936865e-06

Red-Black tree search is better than BST search as the worst case time complexity is $O(\log n)$. The drawback is additional space is required to store the colour of each node and the algorithm is difficult to implement.



GUI

Tkinter library of python is used to design the GUI. We are generating random numbers by taking the size of the array as input from the user.

To show the options for the user to select an algorithm, listbox is used.

CONCLUSION

Searching Algo	Best Case	Average Case	Worst Case
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$
BST Search	$O(\log n)$	$O(\log n)$	$O(n)$
Red-Black Tree Search	-	$O(n)$	$O(n)$

