DSA

Assignment02

(Outputs)

In-Memory DBMS

Using Trees

23K2001

M.Muzammil Siddiqui

BCS-3J

Some operations might be rounded off to 0 ms due to the value being so small in microseconds that the library fails to account it as a millisecond time value!

```
----------------------------------------------
Tree Type: Binary Search Tree (BST)
----------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------
1000            0.996           0               0
10,000          8.388           0               72.966
50,000          31.244          0               3707.02
----------------------------------------------
----------------------------------------------
Tree Type: AVL Tree
----------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------
1000            3.012           0               0
10,000          20.237          0               0
50,000          142.339         0               31.434
----------------------------------------------
----------------------------------------------
Tree Type: B-Tree
----------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------
1000            1.878           0               1.052
10,000          11.551          0               15.683
50,000          94.543          0               31.303
----------------------------------------------
PS F:\Semester Material - Muzammil\FAST-KHI-Semester-3\Data Structures
```

Searching times are being rounded to 0 ms due to very small number of search queries (20). The time taken to search is too small to be calculated by the implementation in this scenario

However, with progression in the search queries, the time being taken starts to increase, in this the search queries were set to 10000. Still tree data structures prove to be efficient for such large datasets.

```
----------------------------------------------------
Tree Type: Binary Search Tree (BST)
----------------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------------
1000            0.91            2.047           0
10,000          8.736           1.996           78.95
50,000          29.231          0               3721.54
----------------------------------------------------
----------------------------------------------------
Tree Type: AVL Tree
----------------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------------
1000            3.703           1.017           0
10,000          28.536          1.003           5.997
50,000          156.656         0               31.657
----------------------------------------------------
----------------------------------------------------
Tree Type: B-Tree
----------------------------------------------------
Dataset         Insert (ms)     Search (ms)     Delete (ms)
----------------------------------------------------
1000            2.351           0.999           0
10,000          17.738          0               7.633
50,000          82.649          0               27.649
----------------------------------------------------
PS F:\Semester Material - Muzammil\FAST-KHI-Semester-3\Data Structures
```

```
-----------------------------------------------
Tree Type: Binary Search Tree (BST)
-----------------------------------------------
Dataset          Insert (ms)      Search (ms)      Delete (ms)
-----------------------------------------------
1000             1                4.998            1
10,000           8.042            8.983            76.189
50,000           46.137           9.777            4046.19
-----------------------------------------------
-----------------------------------------------
Tree Type: AVL Tree
-----------------------------------------------
Dataset          Insert (ms)      Search (ms)      Delete (ms)
-----------------------------------------------
1000             2                4                1
10,000           15.05            15.678           0
50,000           163.161          8.954            35.276
-----------------------------------------------
-----------------------------------------------
Tree Type: B-Tree
-----------------------------------------------
Dataset          Insert (ms)      Search (ms)      Delete (ms)
-----------------------------------------------
1000             1.7              4.259            1.002
10,000           17.402           15.529           6.046
50,000           93.578           7.922            31.814
-----------------------------------------------
PS F:\Semester Material - Muzammil\FAST-KHI-Semester-3\Data Structures
```

Visible increase in search times can be seen for this time, search queries were set to a very large number (50000). Hence, the varying times can be observed.

Different dummy data generation functions for ID and age due to the fact that every record is uniquely identified in a database by its id. However, multiple records can have the same age. Hence, the implementation for unique ID had to be improvised to guarantee that there are no duplicates.

```
void generateIDs(int *arr,int size,int low,int high){
    vector<int> numbers;
    for (int i = low; i <= high; ++i)
        numbers.push_back(i);

    std::shuffle(numbers.begin(), numbers.end(), std::default_random_engine(rand()));

    for (int i = 0; i < size; ++i)
        arr[i] = numbers[i];
}
void generateAges(int *arr, int size, int low, int high) {
    for (int i = 0; i < size; i++)
        arr[i] = low + rand() % (high - low + 1);
}
```

Also, duplicate values as nodes in the trees is also a big issue!

Analysis:

Implementing a data structure can be carefully put into practice by a close observation of the size of data and the required operations, for example in one scenario searching might by the prioritized operation. As in our assignment evident differences could be seen between the three types of trees for different datasets. Noticeably searching in tree data structures seems very efficient by my observation as the computer can process it in a matter of few microseconds which is also why it was being rounded to 0 milliseconds for small number of search queries.