

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**CSE2025 Data Structures**

**PROJECT #2**

Due Date: 07.01.2022

**NAME SURNAME:** SERKAN KORKUT

**STUDENT ID:** 150119036

**Problem Definition**

In this project, I will construct a BST with the keys that are in the given input file. The BST has the following properties:

* its nodes are distributed over 3⎣log4 n⎦ depth levels.
* it provides the best possible total access time

**Functions**

* **BTNodeTypePtr createBTNode():**

This function creates a file pointer to read from the input.txt file. This file contains the numbers to be transferred to BTnode. It reads the numbers one by one and checks whether these numbers are 0 or the same as the previous numbers. It puts these numbers in the array if there is no problem. If the numbers in the array are less than 16, it exits the program. After that, it sorts the array using arraySort function. It splits this array into 2 arrays according to the required depth level. Using arrayOrder function, it puts the numbers in the first array into a new array in a special order and transfers the numbers in this new array sequentially to BTNode using insert function. Finally, it transfers the numbers in the second array sequentially to BTNode using insert function.

**Array:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10** | **5** | **3** | **25** | **70** | **97** | **115** | **2** | **65** | **69** | **32** | **152** | **1** | **21** | **49** | **8** | **12** | **78** |

**After sorting the array:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **5** | **8** | **10** | **12** | **21** | **25** | **32** | **49** | **65** | **69** | **70** | **78** | **97** | **115** | **152** |

**First array:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **5** | **8** | **10** | **12** | **21** | **25** | **32** | **49** | **65** | **69** | **70** | **78** | **97** |

**After arrayOrder function:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **25** | **8** | **69** | **3** | **12** | **49** | **78** | **1** | **2** | **5** | **10** | **21** | **32** | **65** | **70** | **97** |

**First array in a special order:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **25** | **8** | **69** | **3** | **12** | **49** | **78** | **1** | **2** | **5** | **10** | **21** | **32** | **65** | **70** | **97** |

**Second array:**

|  |  |
| --- | --- |
| **115** | **152** |

* **void arrayOrder(int \*array, int size, int \*ordered):**

This function creates another array using the array in the first parameter. First if the length of the array is 3 or greater, it puts the median element of the array into the new array. After that, it creates 2 different arrays using the splitArray function. It calls the function again 2 times by putting these 2 arrays in the first parameter of the function. If the length of the array is less than 3, it puts the numbers in the array into the new array.

* **BTNodeTypePtr insert(int key, BTNodeType \*p):**

This function first checks if BTNode is empty. If BTNode is empty, it creates a node. If it is not null, it recursively calls the function to find a suitable space for the key.

* **int find(BTNodeType \*p, int key):**

This function runs recursively and calls the function again until it finds the key. Returns 1 if key is found, 0 otherwise.

* **int findDepth(BTNodeType \*p, int key):**

This function works recursively and calls the function again until it finds the key, adding one to the depth with each call. As a result, when the key is found, the depth of that key is obtained. The depth of that number is returned.

* **void findNumbersInDepthN(BTNodeType \*p, int depth, int currentDepth, int \*numbers):**

This function works recursively and calls the function again by increasing the currentDepth parameter by 1 until the desired depth level. When the desired depth level is reached, it puts the number in BTNode in the array.

* **int findRank(BTNodeType \*p, int key):**

This function first uses findDepth function to find the depth level where the key is located. After that, it creates an array using findNumbersInDepthN function. This array consists of numbers at the depth level found. Then it searches for the key in this array, finds the rank it occupies in the array, and returns this number.

* **int numberOfKeysInDepthN(BTNodeType \*p, int depth):**

This function creates an array using findNumbersInDepthN function. This array consists of numbers at the desired depth level. It counts the numbers in this array one by one until it reaches 0. Finally returns the result.

* **int findTotalDepth(BTNodeType \*p):**

This function runs recursively and calls the function again until the last BTNode. Finds how many depth levels in BTNode and returns that number.

* **int arraySearch(int \*array, int key):**

This function searches the array and checks if the number in the parameter is in the array. Returns 1 if the number is found, 0 otherwise.

* **void splitArray(int \*array, int n, int \*a, int \*b):**

This function splits an array into two arrays, a and b. It copies the first half of the array into array a, excluding array[n/2]. After that, it copies the numbers after array[n/2] into array b.

* **void arraySort(int \*array, int size):**

This function sorts the array in its parameter from smallest to largest.