GIT Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework 7 Report

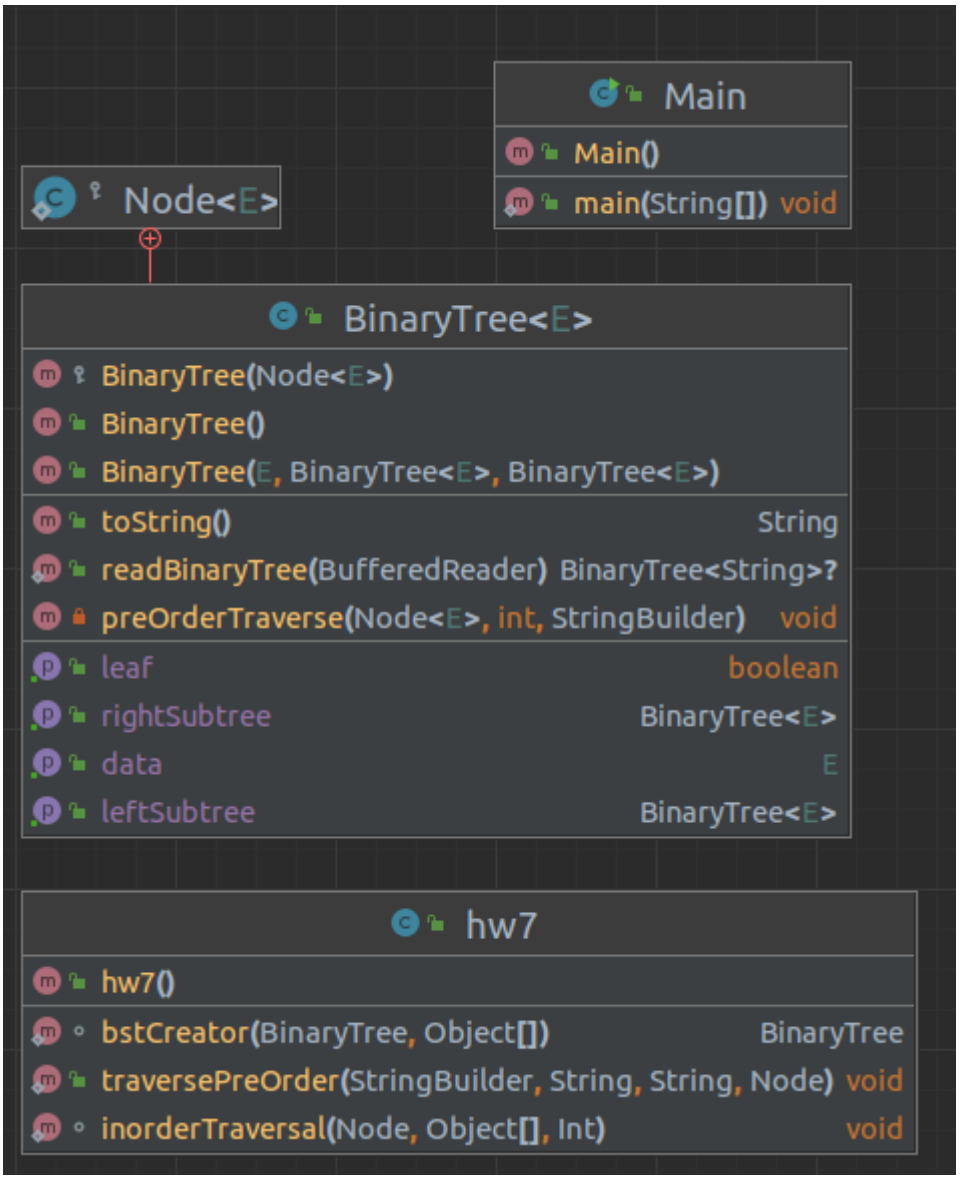
Mustafa MERT

200104004006

1. SYSTEM REQUIREMENTS

To use this program you need an operating system with jdk17 and jre1. We need memory to store strings, binary trees and arrays. Also integers for test cases. Also while this program works it uses CPU, RAM and disk from the pc, if they are better they can give faster results to you.

2. CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

For this assignment I tried to understand binary search tree structure and tree traversals, I studied the textbook's chapters about these topic and researched various internet resources for this. Then I implemented BinaryTree class from the text book and tried to find useful algorithms. After that I implemented the algorithms according to class hierarchy with jdk17.

4. TEST CASES

- Checking the passed array
- Checking if the tree filled with given array successfully
- Checking if the print function works properly

4. TIME COMPLEXITIES

Q1)

bstCreator

sorting => used merge sort time complexity is $T(n) = \Theta(nlogn)$

```
private static void mergeSort(Object[] src,
                              Object[] dest,
                              int low,
                              int high,
                              int off) {
    int length = high - low;

    // Insertion sort on smallest arrays
    if (length < INSERTIONSORT_THRESHOLD) {
        for (int i=low; i<high; i++)
            for (int j=i; j>low &&
                     ((Comparable) dest[j-1]).compareTo(dest[j])>0; j--)
                swap(dest, j, j-1);
        return;
    }

    // Recursively sort halves of dest into src
    int destLow  = low;
    int destHigh = high;
    low  += off;
    high += off;
    int mid = (low + high) >>> 1;
    mergeSort(dest, src, low, mid, -off);
    mergeSort(dest, src, mid, high, -off);

    // If list is already sorted, just copy from src to dest.  This is an
    // optimization that results in faster sorts for nearly ordered lists.
    if (((Comparable)src[mid-1]).compareTo(src[mid]) <= 0) {
        System.arraycopy(src, low, dest, destLow, length);
        return;
    }

    // Merge sorted halves (now in src) into dest
    for(int i = destLow, p = low, q = mid; i < destHigh; i++) {
        if (q >= high || p < mid && ((Comparable)src[p]).compareTo(src[q])<=0)
            dest[i] = src[p++];
        else
            dest[i] = src[q++];
    }
}
```

inorderTraversal => time comlexity **T(n) = 2*T(n/2) + 1 =** $O(n)$

```
static void inorderTraversal(BinaryTree.Node node, Object[] array, Int index)
{
    if (node == null)
        return;


    inorderTraversal(node.left, array, index);

    node.data = array[index.i];
    index.i++;

    inorderTraversal(node.right, array, index);
}
```

Total time complexity is $\Theta(nlogn)$

5. RUNNING AND RESULTS

You can run the main method in Main class to see test cases

```
Array passed to funtion is {4,3,1,7,5,2}

printing the first Tree
4
|  ├──2
|  |  ├──1
|  |  └──3
|  └──5
|  |  └──7

Array passed to funtion is {23,12,6,45,84,25,68,35}

printing the second Tree
45
|  ├──23
|  |  ├──6
|  |  |  └──12
|  |  └──35
|  |  |  └──25
|  └──68
|  |  └──84
```