Homework #4 - 1801042634

Student ID	1801042634
Student Name	Muhammed
Student Surname	Oğuz

— — — HW Part 3 — — —

Analysis

Heap Methods

1- Add method

```
@Override
public boolean add(E item) {
    HeapData<E> temp = new HeapData<E>(item);
    size++;
    return data.add(temp);
}
```

I use PriorityQueue as a data field. So, its add method takes $O(1)$

2- getItem method

```
public HeapData<E> getItem(E item) {

    Iterator<HeapData<E>> it = iterator();

    while(it.hasNext())
    {
        HeapData<E> temp = it.next();
        if (temp.getData().equals(item))
            return temp;
    }
    return null;
}
```

Takes $O(n)time$. looks for all variables with iterator

3- find and search method

```java
@Override
public E find(E item) throws NoSuchElementException {

    HeapIter<E> it = heapIter();
    while (it.hasNext())
    {
        E temp = it.next();
        if (temp.equals(item))
            return temp;
    }

    throw new NoSuchElementException();
}

@Override
public E search(E item) {
    return find(item);
}
```

Takes $O(n)$ time. Looks all elements

4- Merge method

```java
@Override
public boolean merge(Heap<E> other) {
    HeapIter<E> it = other.heapIter();

    while(it.hasNext())
    {
        HeapData<E> temp = new HeapData<E>(it.next());
        data.add(temp);
    }
    return true;
}
```

Takes $O(n)$ time. Add method takes $O(1)$ and this method adds all values of other variable

5- RemovelthBiggest method

```java
@SuppressWarnings("unchecked")
@Override
public Object removeIthBiggestElement(int index) {
    if (index < 0 || index > data.size())
        throw new IndexOutOfBoundsException();

    Object[] arr = data.toArray();
    Arrays.sort(arr);

    HeapData<E> temp =  (HeapData<E>) arr[index];
    remove(temp);
    return temp;
}
```

Uses toArray method. It takes $O(n^2)$. After uses Sort method it takes $O(n)$. Eventually calls remove method. It also takes $O(n)$. Finally all method takes $O(n)$

6- remove(int index) method

```java
}

@SuppressWarnings("unchecked")
@Override
public Object remove(int index) {
    if (index < 0 || index > data.size())
        throw new IndexOutOfBoundsException();

    Object[] arr = data.toArray();

    HeapData<E> temp =  (HeapData<E>) arr[index];
    remove(temp);
    return temp;
}
```

Same as 5.

7- toString() getSize() remove(E item) methods

```java
@Override
public int getSize() {
    return size;
}


@Override
public String toString() {
    return data.toString();
}

public boolean remove(HeapData<E> item) {
    size--;
    return data.remove(item);
}

@Override
public int compareTo(E arg0) {
    if (size == 0)
    {
        return 0;
    }
    else
    {
        return arg0.compareTo(data.peek().getData());
    }
}
```

All takes $O(1)$

8 - mostCount method

```java
public int mostCount() {
    Iterator<HeapData<E>> it = iterator();

    int max = it.next().getCount();

    while(it.hasNext())
    {
        int temp = it.next().getCount();
        if (temp > max)
            max = temp;
    }

    return max;
}
```

This method is helper for BSTHeapTree. It calculates max occurrence count in a heap. It iterates all values. So takes O(n)

BSTHeapTree Methods

1- add method

```
public int add(E item){
    if (data.size() == 0)
        data.add(new Heap<>());

    Heap<E> heap = getItem(item);
    if (heap != null)
    {
        heap.addToTree(item);
        HeapData<E> val = heap.getItem(item);
        return val.getCount();
    }

    Iterator<Heap<E>> it = data.iterator();
    while (it.hasNext())
    {

        Heap<E> temp = it.next();
        if (temp.getSize() < 7)
        {
            temp.addToTree(item);
            HeapData<E> val = temp.getItem(item);
            return val.getCount();
        }
    }

    Heap<E> temp = new Heap<E>();
    temp.addToTree(item);
    data.add(temp);
    HeapData<E> val = temp.getItem(item);
    return val.getCount();

}
```

This method eventually calls Heap's add method. It takes $O(1)$. And some circumstances iterate all tree. It also takes $O(n)$. So this method takes $O(n)$

2- Remove method

```java
public int remove (E item) throws Exception {
    Heap<E> get = getItem(item);
    if (get == null)
        throw new NoSuchElementException("There is no with this element\n");


    HeapData<E> hData = get.getItem(item);
    if (hData.getCount() == 1)
    {
        get.remove(hData);
        return 0;
    }
    else
        hData.decreaseCount();

    return hData.getCount();
}
```

getItem methods takes $O(n)$ time. Cause this methods iterates all data. So takes $O(n)$

3- GetItem method

```java
public Heap<E> getItem(E item) {
    Iterator<Heap<E>> it = data.iterator();

    while (it.hasNext())
    {
        Heap<E> temp = it.next();
        if (temp.getItem(item) != null)
        {
            return temp;
        }
    }
    return null;
}
```

Iterates All tree. So takes $O(n * 7) = O(n)$

4- Find method

```java
public int find(E item) {
    HeapData<E> get = getItem(item).getItem(item);

    if (get == null)
        throw new NoSuchElementException();

    return get.getCount();

}
```

Calls getItem method. It takes $O(n)$. So this also takes $O(n)$

5- find_mode() method

```java
public int find_mode() {
    if (data.size() == 0)
        throw new NullPointerException();

    Iterator<Heap<E>> it = data.iterator();

    int max = it.next().mostCount();

    while(it.hasNext())
    {
        int temp = it.next().mostCount();
        if (temp > max)
            max = temp;
    }
    return max;
}
```

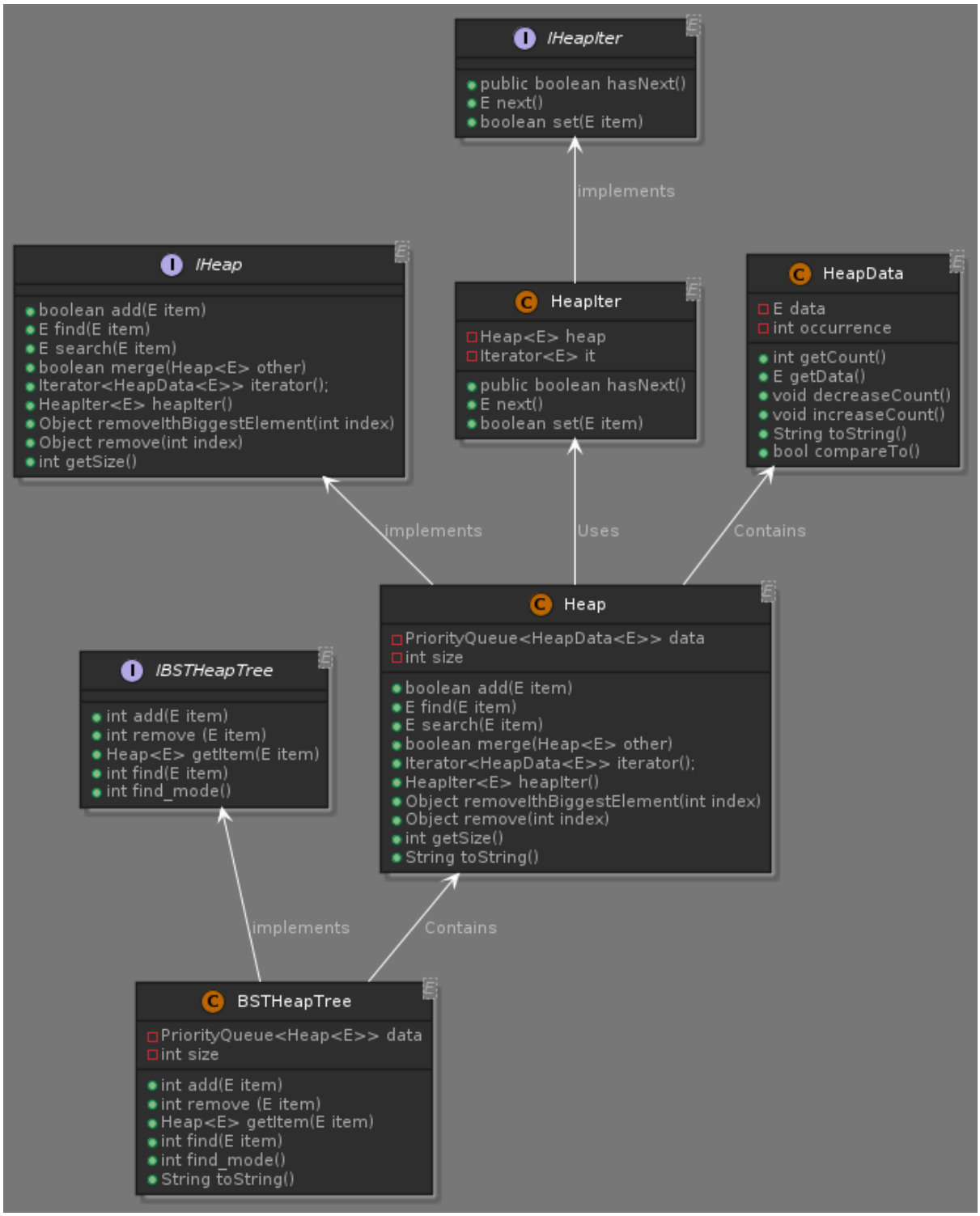
This function calls, Heap's mostCount function in a loop. MosutCount function takes $O(n)$. Also this loop iterates all tree. It becames $O(n * n) = O(n^2)$

A Function in Main Class

```java
public static int find_mode_array(ArrayList<Integer> arr) {
    int max = 0;
    int temp = 1;
    for (int i = 0; i < arr.size() - 1; i++)
    {
        if ( arr.get(i).equals(arr.get(i + 1)) )
        {
            temp++;
        }
        else
        {
            if (max < temp)
                max = temp;
            temp = 1;
        }
    }
    return max;
}
```

Finds most occurrence count in arraylist. Iterates all over the array. Takes $O(n)$

Class Diagram



System Requirements

1- Have to create objects

```java
Heap<Integer> heap = new Heap<Integer>();
Heap<Integer> heap2 = new Heap<Integer>();
```

```java
BSTHeapTree<Integer> tree = new BSTHeapTree<>();
```

After, available methods here. All methods could use freely

2- Available functions

```java
/**
 * IHash
 */
public interface IHeap<E extends Comparable<E>> {


    /**
     * Add item to heap
     * @param item The item going to add heap
     * @return true
     */
    boolean add(E item);

    /**
     * Find given element if exist
     * @param item The element going to find
     * @return element if exist, if not throw exception
     * @throws NoSuchElementException IF element not exist, throw exception
     */
    E find(E item) throws NoSuchElementException;

    /**
     * Search element. Same as find
     * @param item The element going to searched
     * @return return element if exist
     */
    E search(E item);

    /**
     * Merge with another Heap structure
     * @param other The structure to going to merged
     * @return true
     */
    boolean merge(Heap<E> other);

    /**
     * Return a iterator with Private data
     * @return data.iterator();
     */
    Iterator<HeapData<E>> iterator();

    /**
     * Return a HeapIterator. A special iterator for this structure
     * @return HeapIterator
     */
    HeapIter<E> heapIter();

    /**
     * Sort all values and remove Ith biggest element
     * @param index Determine, which biggest element going to removed
     * @return removed object
     * @throws IndexOutOfBoundsException if index is invalid, throw exception
     */
    Object removeIthBiggestElement(int index) throws IndexOutOfBoundsException;

    /**
     * Remove element at given index
```

```java

public interface IBSTHeapTree<E extends Comparable<E>> {


    /**
     * Add item to tree due to order
     * @param item The item going to add to tree
     * @return number of occurrence after insertion
     */
    int add(E item);

    /**
     * Remove item
     * @param item The item going to removed
     * @return number of occurrence after remove
     * @throws Exception If not exist
     */
    int remove (E item) throws Exception;

    /**
     * Get given item
     * @param item The item going to get
     * @return item
     */
    Heap<E> getItem(E item);

    /**
     * Find number of occurrence of given item
     * @param item The item going to search
     * @return number of occurrence
     * @throws NoSuchElementException if not exist
     */
    int find(E item) throws NoSuchElementException;

    /**
     * Find max occurrence of all
     * @return Number of max occurrence
     * @throws NullPointerException erro
     */
    int find_mode() throws NullPointerException;


}
```

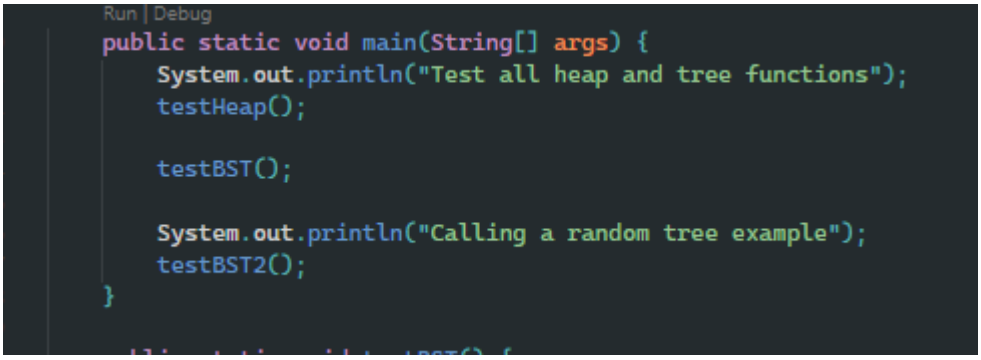
Problem Solution Approach

First, I have to implement a data class for keeping both occurrence and value. So I created a data class.
After that, this class should be `Comperable` cause of PriorityQueue only excepts Comperable Objects. So I implement `CompareTo` Function.
Secondly, I have to create heap class. I created and Implemented with PriorityQueue and my new Data Class. Also I should Implement a new iterator method called `set` due to instructions. I implemented PriorityQueue's iterator with extra set method.

Finally, after that implementation, using PriorityQueue of this new Heap class allow me to create BSTHeapTree very easily.

Test Cases

1- Entering area

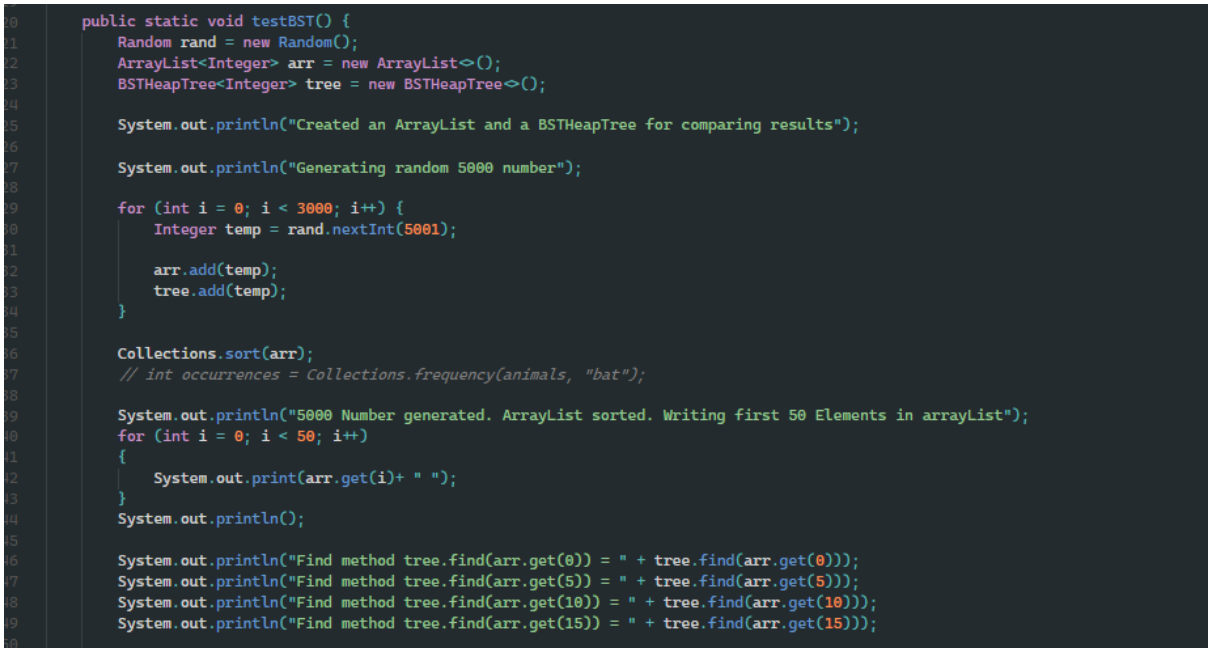


```
Run | Debug
public static void main(String[] args) {
    System.out.println("Test all heap and tree functions");
    testHeap();

    testBST();

    System.out.println("Calling a random tree example");
    testBST2();
}
```

2- Create ArrayList and Tree and fill with random variables



```
public static void testBST() {
    Random rand = new Random();
    ArrayList<Integer> arr = new ArrayList<>();
    BSTHeapTree<Integer> tree = new BSTHeapTree<>();

    System.out.println("Created an ArrayList and a BSTHeapTree for comparing results");

    System.out.println("Generating random 5000 number");

    for (int i = 0; i < 3000; i++) {
        Integer temp = rand.nextInt(5001);

        arr.add(temp);
        tree.add(temp);
    }

    Collections.sort(arr);
    // int occurrences = Collections.frequency(animals, "bat");

    System.out.println("5000 Number generated. ArrayList sorted. Writing first 50 Elements in arrayList");
    for (int i = 0; i < 50; i++)
    {
        System.out.print(arr.get(i)+ " ");
    }
    System.out.println();

    System.out.println("Find method tree.find(arr.get(0)) = " + tree.find(arr.get(0)));
    System.out.println("Find method tree.find(arr.get(5)) = " + tree.find(arr.get(5)));
    System.out.println("Find method tree.find(arr.get(10)) = " + tree.find(arr.get(10)));
    System.out.println("Find method tree.find(arr.get(15)) = " + tree.find(arr.get(15)));
```

3- Compare occurrence values if matches or not

```
System.out.println("Compare 100 elements occurrences with ArrayList and Tree. If one of them is not correct. Warn.");
boolean flag = false;
for (int i = 0; i < 100; i++)
{
    int temp = arr.get(0);
    int occurrenceCollection = Collections.frequency(arr, temp);
    int occurrenceTree = tree.find(temp);
    if (occurrenceCollection ≠ occurrenceTree)
        flag = true;
}
if (flag)
    System.out.println("Failed to test. Some occurrences not matches");
else
    System.out.println("Passed test successfully");
```

4 - Test find method with invalid variables

```
System.out.println("Call tree.find() with negative values (which those values not generated)");
for (int i = -10; i < -1; i++)
{
    System.out.print("Calling tree.find(negative number) = ");
    try {
        System.out.println(tree.find(i));
    } catch (Exception e) {
        System.out.print("Not in the tree\n");
    }
}

System.out.println("ArrayList most occurrence count = " + find_mode_array(arr));
System.out.println("Tree most occurrence count = " + tree.find_mode());
```

5- Remove 100 values

```
System.out.println("Tree most occurrence count = " + tree.find_mode());

flag = false;
System.out.println("Remove 100 values.");
for (int i = 0; i < 100; i++)
{
    try {
        tree.remove(arr.get(i));
    } catch (Exception e) {
        flag = true;
        e.printStackTrace();
    }
    arr.remove(arr.get(i));
}
if (flag)
    System.out.println("Failed to test. Some occurrences not matches");
else
    System.out.println("Passed test successfully");
```

6- Remove 10 invalid values

```
        System.out.println("Call tree.remove() with negative values (which those values not generated)");
        for (int i = -10; i < -1; i++)
        {
            System.out.print("Calling tree.remove(negative number) = ");
            try {
                System.out.println(tree.remove(i));
            } catch (Exception e) {
                System.out.print("Not in the tree\n");
            }
        }
```

7 - Test all heap methods

```
    public static void testHeap() {
        Heap<Integer> heap = new Heap<Integer>();
        Heap<Integer> heap2 = new Heap<Integer>();


        heap.add(15); heap.add(2);
        heap.add(8); heap.add(5);
        heap.add(4);

        heap2.add(12); heap2.add(1);
        heap2.add(7); heap2.add(8);

        heap.removeIthBiggestElement(2);
        System.out.println("Heap1 = " + heap.toString());
        HeapIter<Integer> iter = heap.heapIter();
        iter.next();
        iter.set(100);
        System.out.println("Heap1 after removeithBiggest and set with iterator");    "removeith": Unknown
        System.out.println("Heap1 = " + heap.toString());
        System.out.println("Heap2 = " + heap2.toString());
        heap.merge(heap2);
        System.out.println("Merged = " + heap.toString());
        System.out.println(heap);

        System.out.println("heap.find(100) = " + heap.find(100));
    }
```

Running and Results

1- Heap Part

```
make
javac  CSE222_hw04/interface_oguz/*.java CSE222_hw04/src_oguz/*.java Test.java  -d classfiles
java -cp classfiles Test
Test all heap and tree functions
Heap1 = [15, 4, 8, 2]
Heap1 after removeithBiggest and set with iterator
Heap1 = [100, 8, 2, 4]
Heap2 = [12, 8, 7, 1]
Merged = [100, 12, 8, 4, 8, 2, 7, 1]
[100, 12, 8, 4, 8, 2, 7, 1]
heap.find(100) = 100
```

2- Tree Part a

```
Created an ArrayList and a BSTHeapTree for comparing results
Generating random 5000 number
5000 Number generated. ArrayList sorted. Writing first 50 Elements in arrayList
1 3 7 8 9 14 15 15 18 19 22 24 24 25 25 25 26 26 26 27 29 29 33 34 35 37 37 49 50 57 58 59 64 6
 84 84 86 87 87 88 89 90
Find method tree.find(arr.get(0)) = 1
Find method tree.find(arr.get(5)) = 1
Find method tree.find(arr.get(10)) = 1
Find method tree.find(arr.get(15)) = 3
Compare 100 elements occurrences with ArrayList and Tree. If one of them is not correct. Warn.
Passed test successfully
Call tree.find() with negative values (which those values not generated)
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
Calling tree.find(negative number) = Not in the tree
ArrayList most occurrence count = 5
Tree most occurrence count = 5
Remove 100 values.
Passed test successfully
Call tree.remove() with negative values (which those values not generated)
Calling tree.remove(negative number) = Not in the tree
Calling tree.remove(negative number) = Not in the tree
```

3- Tree Part b

```
Calling a random tree example
[[87,1, 80,2, 57,1, 38,1, 60,1, 51,3, 54,1]
[124,1, 31,1, 52,2, 15,1, 29,1, 13,4]
[37,1, 23,1, 10,2, 16,1, 19,1, 9,3, 3,1]
[43,3, 39,2]]
```