

**Gebze Technical University
ComputerEngineering**

CSE 222 -2018 Spring

HOMEWORK 6 REPORT

**YUNUS ÇEVİK
141044080**

Course Assistant: Fatma Nur Esirci

1 WorstRedBlackTree

1.1 Problem Solution Approach

Rotation Left Pseudo Code' u → protected Node < E > rotateLeft(Node < E > localRoot)

1. Remember the value of root. right (temp = root. right).
2. Set root. right to the value of temp. left.
3. Set temp. left to root.
4. Set root to temp.

Yukarıdaki Pseudo Code' da belirtilmek istenen swap işlemidir. Eğer Red – Black Tree' nin balans değeri sağ taraftan dolayı bozuk ise sola doğru rotation işlemi yapılır. Bunun içinde 4 adımda swap yapılır ve sayıların yeri değiştirilerek Red – Black Tree' nin balansı dengelenir.

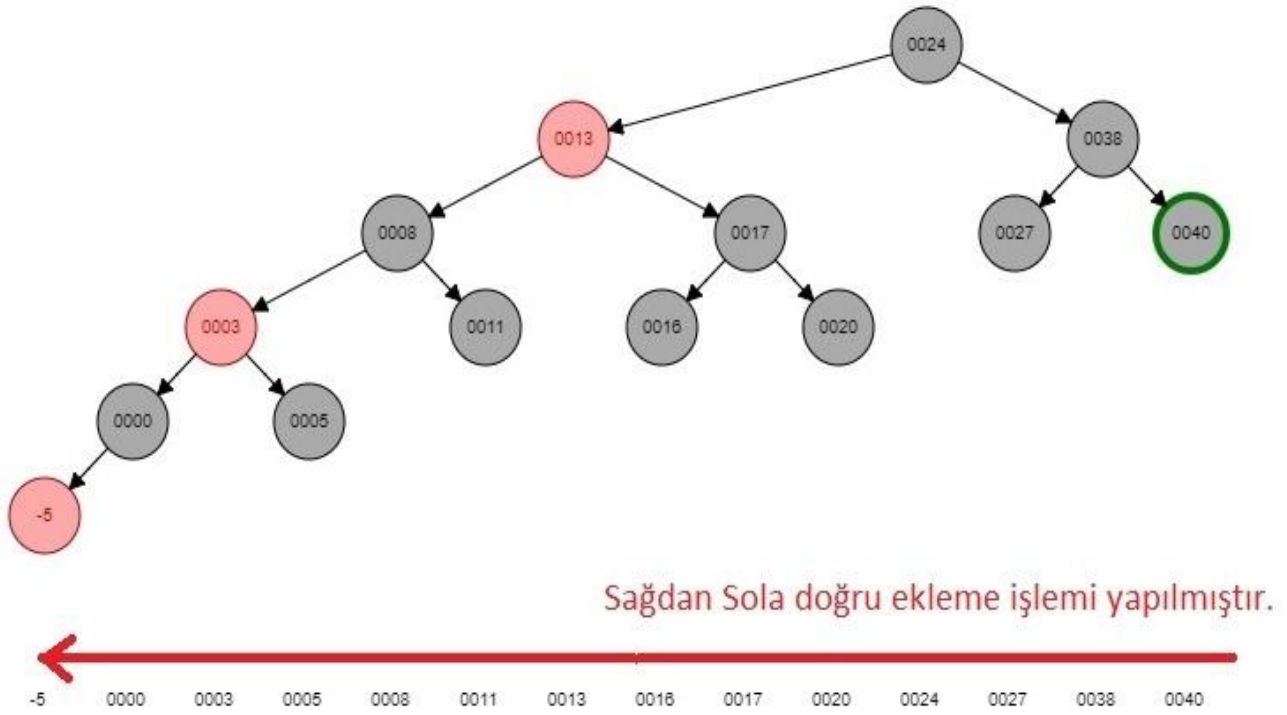
1.2 Test Cases

1.2.1 Red – Black Tree 1

1.2.1.1 Ekleme işlemi ve Rotation işlemler

Eleman Sayısı	Ekleme İşlemi	Ekleme Sırası Oluşan İşlemler
1	rbt1.add(40);	40 (Black) eklendi.
2	rbt1.add(38);	38 < 40 => 38 (Red) eklendi.
3	rbt1.add(27);	27 < 38 => iki red node 38' den Sağa Rotation ve 27 (Red) eklendi. (38 Black - 40 Red).
4	rbt1.add(24);	24 < 27 => iki red node 24 (Red) eklendi. (27 ve 40 Black).
5	rbt1.add(20);	20 < 24 => iki red node 24' den Sağa Rotation ve 20 (Red) eklendi. (24 Black - 27 Red).
6	rbt1.add(17);	17 < 20 => iki red node 17 (Red) eklendi. (24 Red - 20 ve 27 Black).
7	rbt1.add(16);	16 < 17 => iki red node 16 (Red) eklendi. (17 Black - 20 Red).
8	rbt1.add(13);	13 < 16 => iki red node renk değiştir ve 24' den Sağa Rotation 13 (Red) eklendi. (17,38 Red - 20, 24, 27, 40 Black).
9	rbt1.add(11);	11 < 13 => iki red node 13' den Sağa Rotation ve 11 (Red) eklendi. (13 Black - 16 Red).
10	rbt1.add(8);	8 < 11 => iki red node renk değiştir ve 8 (Red) eklendi. (11, 16, 17, 38 Black - 13 Red).
11	rbt1.add(5);	5 < 8 => iki red node 8' den Sağa Rotation ve 5 (Red) eklendi. (8 Black - 11 Red).
12	rbt1.add(3);	3 < 5 => iki red node renk değiştir, 13' den Sağa Rotation 3 (Red) eklendi. (5, 11, 13 Black - 8,17 Red).
13	rbt1.add(0);	0 < 3 => iki red node 3' den Sağa Rotation 0 (Red) eklendi. (3 Black - 5 Red).
14	rbt1.add(-5);	(-5) < 0 => iki red node renk değiştir ve -5 (Red) eklendi. (0, 5, 8, 17 Black - 3,13 Red).

$height \leq 2\log(n+1) \rightarrow n = \text{internal node}$



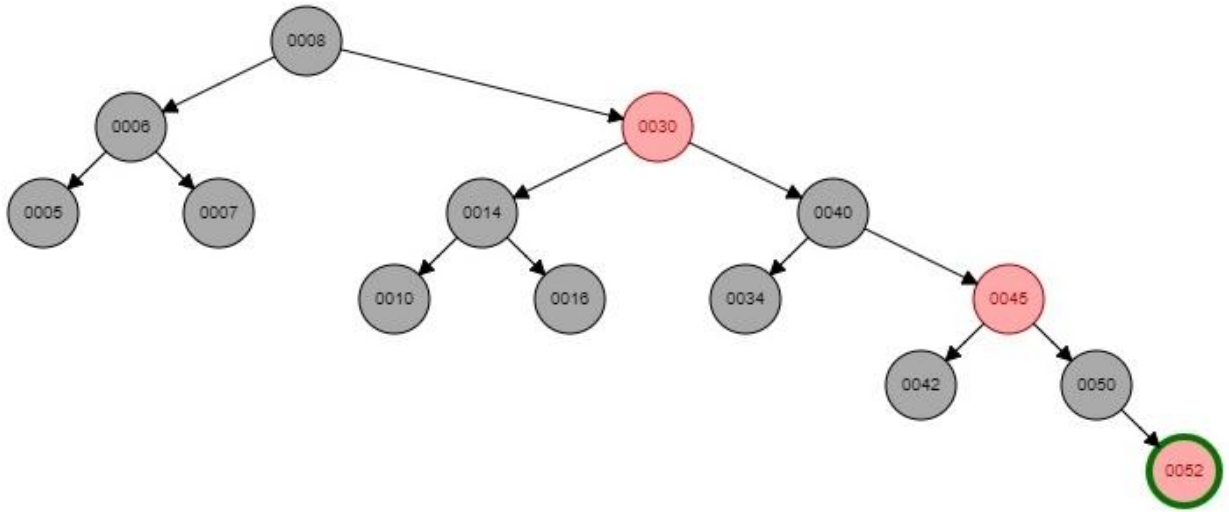
Verilen değerler büyükten küçüğe azalan şekilde eklendiği takdirde Worst Red – Black Tree oluşur.

1.2.2 Red – Black Tree 2

1.2.2.1 Ekleme işlemi ve Rotation işlemler

Eleman Sayısı	Ekleme İşlemi	Ekleme Sırası Oluşan İşlemler
1	rbt2.add(5);	5 (Black) eklendi.
2	rbt2.add(6);	$6 \geq 5 \Rightarrow 6$ (Red) eklendi.
3	rbt2.add(7);	$7 \geq 6 \Rightarrow$ iki red node 6' dan Sola Rotation ve 7 (Red) eklendi. (6 Black - 5 Red).
4	rbt2.add(8);	$8 \geq 7 \Rightarrow$ iki red node 8 (Red) eklendi. (5 ve 7 Black).
5	rbt2.add(10);	$10 \geq 8 \Rightarrow$ iki red node 8 den Sola Rotation ve 10 (Red) eklendi. (8 Black - 7 Red).
6	rbt2.add(14);	$14 \geq 10 \Rightarrow$ iki red node 14 (Red) eklendi. (8 Red - 7 ve 10 Black).
7	rbt2.add(16);	$16 \geq 14 \Rightarrow$ iki red node 14' den Sola Rotation ve 16 (Red) eklendi. (14 Black - 10 Red).
8	rbt2.add(30);	$30 \geq 16 \Rightarrow$ iki red node renk değiştir ve 8' den Sola Rotation 30 (Red) eklendi. (6, 14 Red - 8, 10, 16 Black).
9	rbt2.add(34);	$34 \geq 30 \Rightarrow$ iki red node 30' dan Sola Rotation ve 34 (Red) eklendi. (30 Black - 16 Red).
10	rbt2.add(40);	$40 \geq 34 \Rightarrow$ iki red node renk değiştir ve 40 (Red) eklendi. (6, 14, 16, 34 Black - 30 Red).
11	rbt2.add(42);	$42 \geq 40 \Rightarrow$ iki red node 40' dan Sola Rotation ve 42 (Red) eklendi. (40 Black - 34 Red).
12	rbt2.add(45);	$45 \geq 42 \Rightarrow$ iki red node renk değiştir, 30'dan Sola Rotation 45 (Red) eklendi. (30, 34, 42 Black - 14, 40 Red).
13	rbt2.add(50);	$50 \geq 45 \Rightarrow$ iki red node 45' den Sola Rotation 0 (Red) eklendi. (45 Black - 42 Red).
14	rbt2.add(52);	$52 \geq 50 \Rightarrow$ iki red node renk değiştir ve 52 (Red) eklendi. (14, 40, 42, 50 Black - 30, 45 Red).

$height \leq 2\log(n+1) \rightarrow n = \text{internal node}$



Soldan Sağa doğru ekleme işlemi yapılmıştır



Verilen değerler küçükten büyüğe artan şekilde eklendiği takdirde Worst Red – Black Tree oluşur.

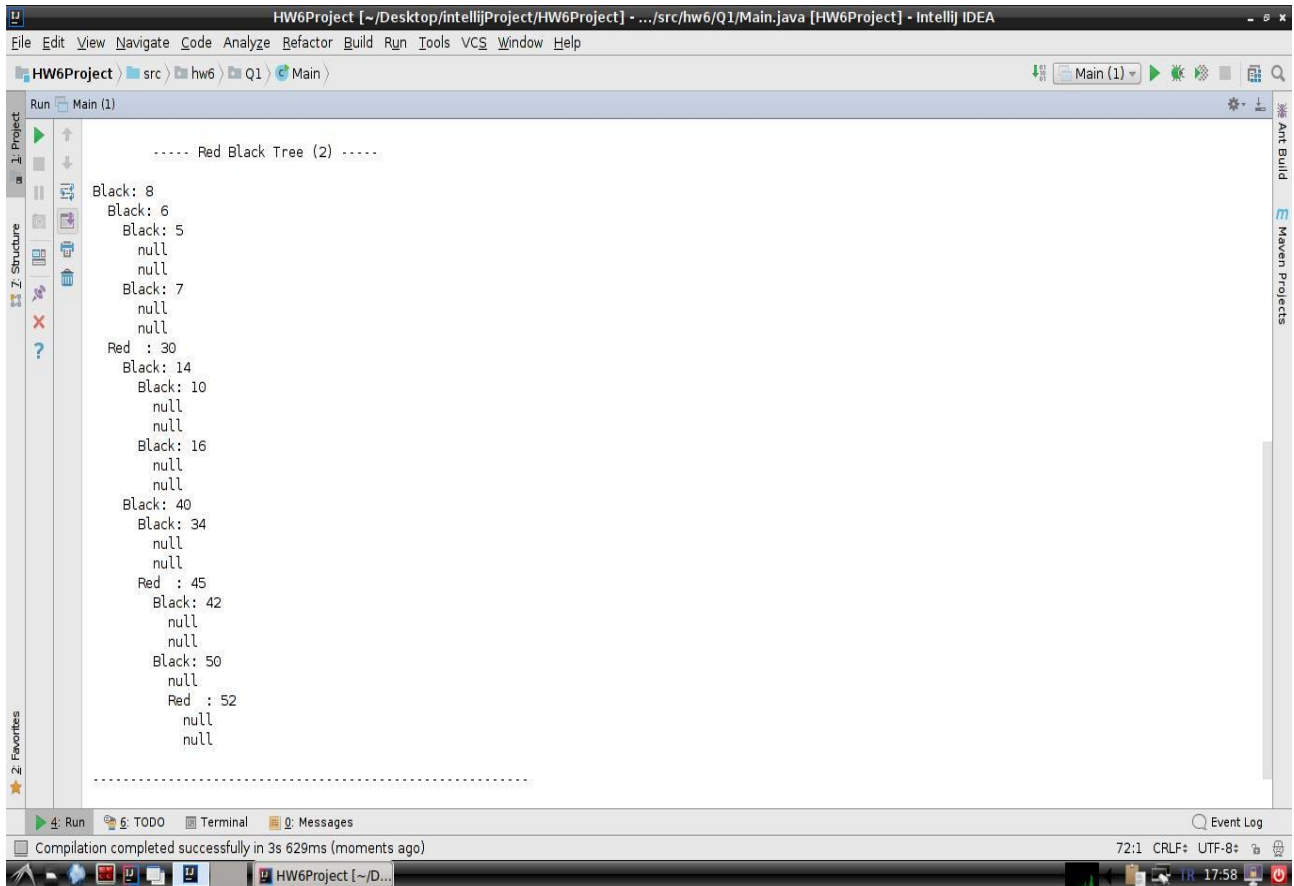
1.3 Running Commands and Results

1.3.1 Q1-> Main.java dosyasında yazılan Test çıktısı 1.

```
Run: Main (1)
/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...
..... Red Black Tree (1) .....

Black: 24
Red : 13
Black: 8
Red : 3
Black: 0
Red : -5
null
null
null
Black: 5
null
null
Black: 11
null
null
Black: 17
Black: 16
null
null
Black: 20
null
null
Black: 38
Black: 27
null
null
Black: 40
null
null
```

1.3.2 Q1-> Main.java dosyasında yazılan Test çıktısı 2.



```
----- Red Black Tree (2) -----
Black: 8
  Black: 6
    Black: 5
      null
      null
    Black: 7
      null
      null
  Red : 30
    Black: 14
      Black: 10
        null
        null
      Black: 16
        null
        null
    Black: 40
      Black: 34
        null
        null
    Red : 45
      Black: 42
        null
        null
      Black: 50
        null
        Red : 52
          null
          null
          null
```

2 binarySearch method

2.1 Problem Solution Approach

Binary Search Algoritma' sının Pseudo Code' u

private int binarySearch(E item, E[] data, int beginValue, int endValue)

1. if item is bigger than data[endValue -1].
2. Return endValue.
3. else if item is equal and smaller than data[beginValue].
4. Return beginValue.
5. if (beginValue + endValue) % 2 equal to 0.
6. Set middleValue to (beginValue + endValue) / 2.
7. else
8. Set midleValue to (beginValue + endValue) / 2 + 1.
9. if middleValue is equal and bigger than endValue.
10. Return middleValue.
11. if item equal to data[middleValue].
12. Return middleValue.

```
bTree2.add(111.66);
```

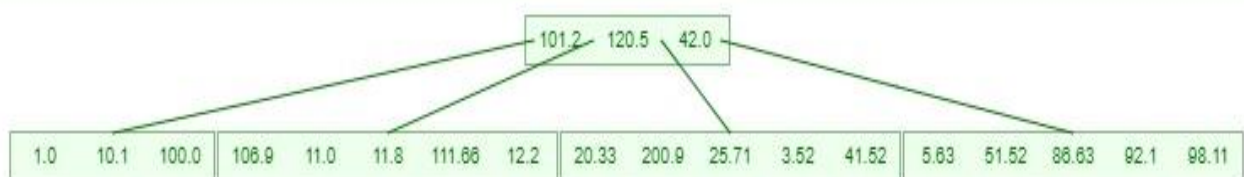
```

bTree2.add(120.5);
bTree2.add(10.1);
bTree2.add(11.8);
bTree2.add(92.1);
bTree2.add(98.11);
bTree2.add(20.33);
bTree2.add(25.71);
bTree2.add(41.52);
bTree2.add(42.0);
bTree2.add(51.52);
bTree2.add(86.63);
bTree2.add(100.0);
bTree2.add(101.2);
bTree2.add(106.9);
bTree2.add(1.0);
bTree2.add(3.52);
bTree2.add(5.63);
bTree2.add(11.0);
bTree2.add(12.2);
bTree2.add(200.9);

System.out.println( bTree2.toString());

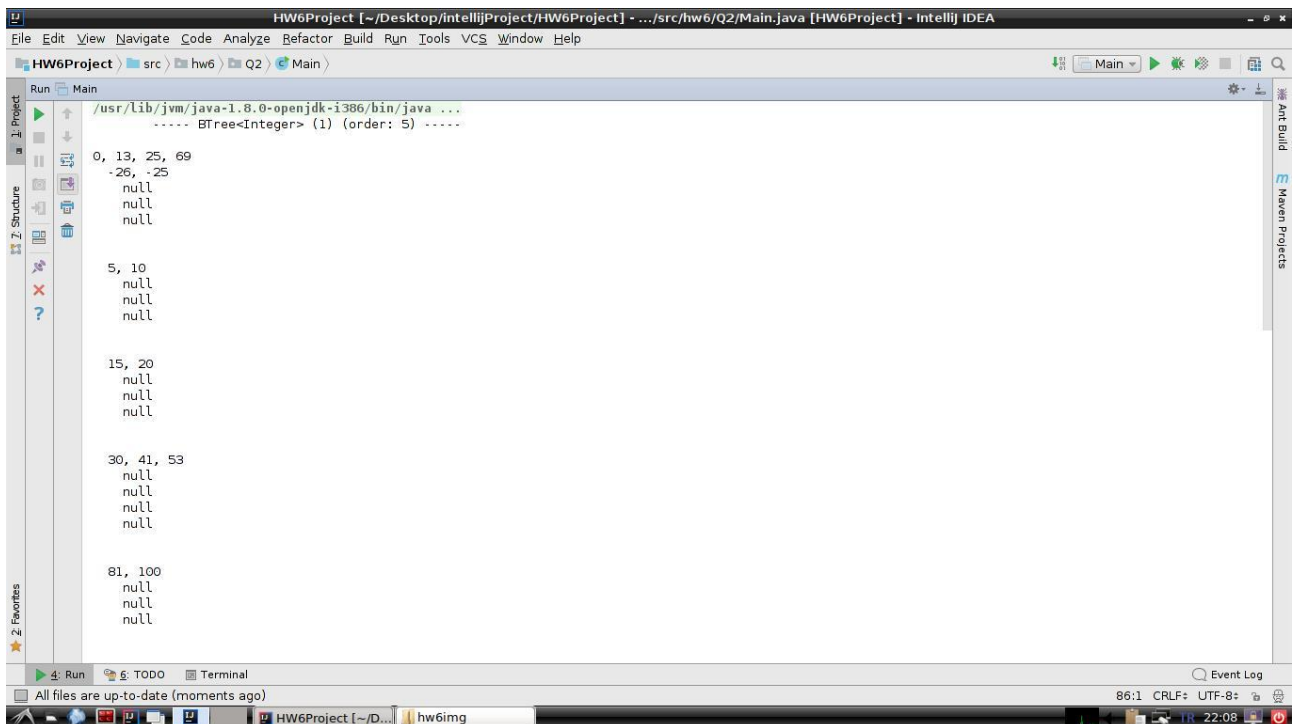
```

```
System.out.println("-----\n");
```



2.3 Running Commands and Results

2.3.1 Q2-> Main.java dosyasında yazılan Test çıktısı 1.



```
----- BTree<Integer> (1) (order: 5) -----
0, 13, 25, 69
-26, -25
null
null
null

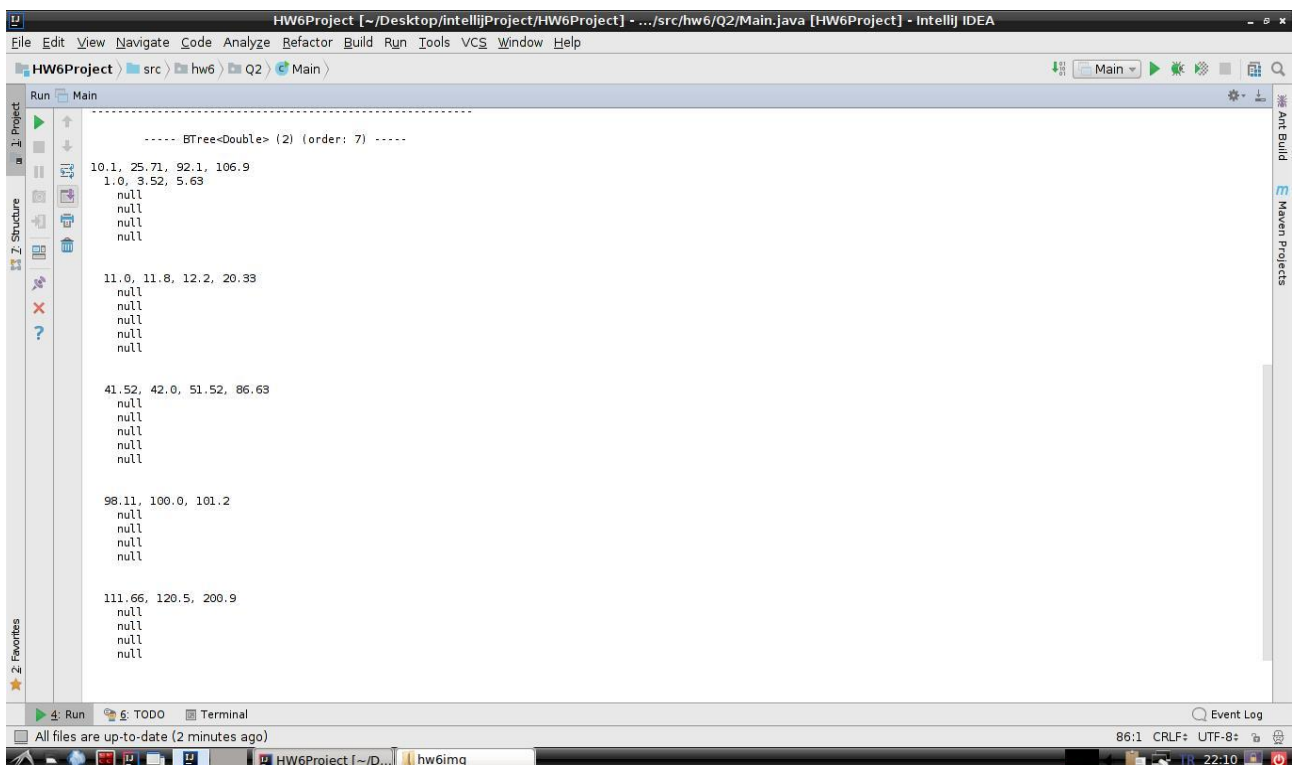
5, 10
null
null
null

15, 20
null
null
null

30, 41, 53
null
null
null
null

81, 100
null
null
null
null
```

2.3.2 Q2-> Main.java dosyasında yazılan Test çıktısı 2.



```
----- BTree<Double> (2) (order: 7) -----
10.1, 25.71, 92.1, 106.9
1.0, 3.52, 5.63
null
null
null
null

11.0, 11.8, 12.2, 20.33
null
null
null
null
null

41.52, 42.0, 51.52, 86.63
null
null
null
null
null

98.11, 100.0, 101.2
null
null
null
null

111.66, 120.5, 200.9
null
null
null
null
```


3 Project 9.5 in book

3.1 Problem Solution Approach

3.1.1 AVL Tree' nin Binary Tree alan Constructor Algoritmasının Pseudo Code' u.

public AVLTree(BinaryTree<E> binaryTree) throws Exception

1. Set result to isAVLTree(BinaryTree<E> binaryTreeRoot). // (binaryTree)
2. If result equal to true.
3. Called addAllBinaryTreeToAVLTree(BinaryTree<E> binaryTree, AVLTree<E> avl) metod. // (binaryTree, this)
Print operation: Binary Tree is a AVL Tree.
4. else
5. throw new Exception("Binary Tree is not AVL Tree. ").

Yukarıdaki Pseudo Code' da isAVLTree metodundan dönen değere göre parametrede gelen Binary Tree, AVL Tree' ye eklenir ve Binary Tree' nin bir AVL Tree olduğu belirtilir. Aksi takdirde, Binary Tree' nin AVL Tree olmadığı belirtilir.

3.1.2 AVL Tree' nin Delete metodunun Algoritmasının Pseudo Code' u.

3.1.2.1 *public E delete(E item)*

1. Set decrease to false.
2. if item equal to null.
3. Return null.
4. Set root to delete(AVLNode<E> localRoot, E item). // ((AVLNode<E>) root, item)
5. Return deleteReturn.

Yukarıdaki Pseudo Code' da delete metodu gösterilmektedir. Delete işlemi olduğu takdirde silinen eleman geri döndürülür. Ayrıca 4. Satırda bulunan delete metodu ise helper metottur. Silme işlemi burada gerçekleştirilir ve recursive bir metottur. Ayrıca silme işlemi ile sırasında birçok yardımcı metotta vardır bunlar proje dosyasında ve Javadoc'ta açıkça belirtilmiştir.

3.1.2.2 *private AVLNode<E> delete(AVLNode<E> localRoot, E item) // Helper Metot*

1. if localRoot equal to null
2. Set localRoot to null and Return localRoot.

```

3.  if item equal to localRoot. data
4.      Set deleteReturn to localRoot. data.
5.      if localRoot. left equal to null
6.          Set decrease to true and Return localRoot. right.
7.      else if localRoot. right equal to null
8.          Set decrease to true and Return localRoot. left.
9.      else
10.         if localRoot. left. right equal to null
11.             Set localRoot. data to localRoot. left. data.
12.             Set localRoot. left to localRoot. left. left and Return localRoot.
13.         else
14.             Set localRoot. data to findLargestChild(localRoot. left ).
15.             Return localRoot.
16. else if item is smaller than localRoot. data
17.     Set localRoot. left to delete( localRoot. left, item). // Recursive Call
18.     if decrease equal to true
19.         Call incrementBalance(localRoot) metod.
20.         if localRoot. balance is bigger than AVLNode. RIGHT_HEAVY
21.             Return rebalanceRightForDelete (localRoot).
22.         else
23.             Return localRoot.
24.     else
25.         Return localRoot.
26. else
27.     Set localRoot. right to delete( localRoot. right, item).// Recursive Call
28.     if decrease equal to true
29.         Call decrementBalance(localRoot) metod.
30.         if localRoot. balance is smaller than AVLNode. LEFT_HEAVY
31.             Return rebalanceLeftForDelete (localRoot).
32.         else
33.             Return localRoot.
34.     else
35.         Return localRoot.

```

3.1.3 AVL Tree' nin addAllBinaryTreeToAVLTree metodunun Algoritmasının Pseudo Code' u.

private void addAllBinaryTreeToAVLTree(BinaryTree<E> binaryTree, AVLTree<E> avl) =>
Binary Tree icindekileri AVL Tree ye ekler.

1. if binaryTree not equal to null
2. avl.add(binaryTree.getData()).
3. addAllBinaryTreeToAVLTree(binaryTree.getLeftSubtree(), avl).
4. addAllBinaryTreeToAVLTree(binaryTree.getRightSubtree(), avl).

3.1.4 AVL Tree' nin isAVLTree metodunun Algoritmasının Pseudo Code' u.

private boolean isAVLTree(BinaryTree<E> binaryTreeRoot) => AVL Tree olup olmadığına

1. Return (findMaxDepth(binaryTreeRoot) - findMinDepth(binaryTreeRoot)) <= 1;

3.1.5 AVL Tree' nin findMaxDepth metodunun Algoritmasının Pseudo Code' u.

private int findMaxDepth(BinaryTree<E> binaryTreeRoot) => Maksimum derinliği bulur.

1. if binaryTreeRoot equal null
2. Return 0.
3. Return 1 + Math.max(findMaxDepth(binaryTreeRoot.getLeftSubtree()), findMaxDepth(binaryTreeRoot.getRightSubtree())).

3.1.6 AVL Tree' nin findMinDepth metodunun Algoritmasının Pseudo Code' u.

private int findMinDepth(BinaryTree<E> binaryTreeRoot) => Minimum derinliği bulur.

1. if binaryTreeRoot equal null
2. Return 0.
3. Return 1 + Math.min(findMinDepth(binaryTreeRoot.getLeftSubtree()), findMinDepth(binaryTreeRoot.getRightSubtree())).

3.1.7 AVL Tree' nin findMinDepth metodunun Algoritmasının Pseudo Code' u.

private void incrementBalance(AVLNode<E> node) => Verilen Node' un artan dengesi.

1. ++node. balance.
2. if node. balance is bigger than AVLNode. BALANCED.
3. Set increase to true and set decrease to false.
4. else
5. Set increase to false and set decrease to true.

Not: Yukarıda yer alan Constructor ve AVL kontrolü için yazılmış helper metodlarla birlikte, Delete ve delete metodunun helper metodunun Psuedo Code' ları yer almaktadır.

Yardımcı metod olarak yazılan kodlar Project dosyasının Q3 klasörü içindeki AVLTree. Java içinde mevcuttur.

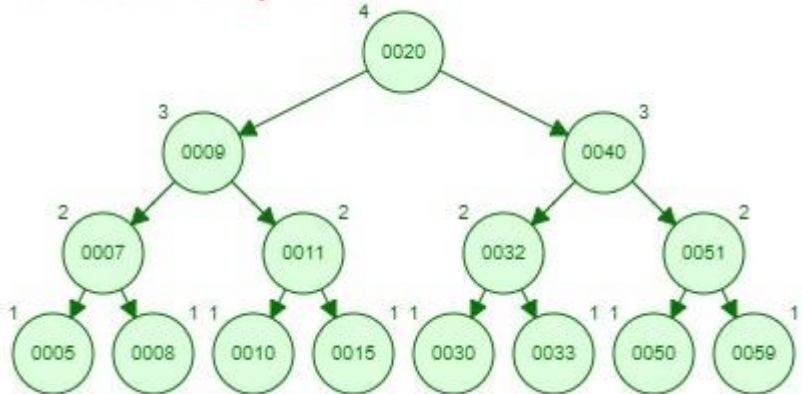
3.2 Test Cases

```
System.out.println("\t\t----- AVL<Integer> (1) -----\\n");

BinarySearchTree<Integer> bst1 = new BinarySearchTree();
for(int i = 10; i < 20; i++)
    bst1.add(i);
try {
    System.out.println(bst1 + "\\n\\n");
    AVLTree<Integer> avl1 = new AVLTree<>(bst1);
} catch (Exception ex) {
    System.out.println(ex.getMessage());
}

System.out.println("-----\\n");
System.out.println("\t\t----- AVL<Integer> (2) -----\\n");
BinarySearchTree<Integer> bst2 = new BinarySearchTree();
bst2.add(20);
bst2.add(9);
bst2.add(40);
bst2.add(7);
bst2.add(11);
bst2.add(32);
bst2.add(51);
bst2.add(5);
bst2.add(8);
bst2.add(10);
bst2.add(15);
bst2.add(30);
bst2.add(33);
bst2.add(50);
bst2.add(59);
```

AVL Tree olan Binary Tree



```

try {
    System.out.println(bst2 + "\n\n");
    AVLTree<Integer> avl2 = new AVLTree<>(bst2);
    System.out.println(avl2.toString() + "\n\n");

    System.out.println(avl2.delete(7) + " <= bst2.delete(7)\n");
    System.out.println(avl2.delete(11) + " <= bst2.delete(11)\n");

    System.out.println(avl2.toString());
} catch (Exception ex) {
    System.out.println(ex.getMessage());
}

```

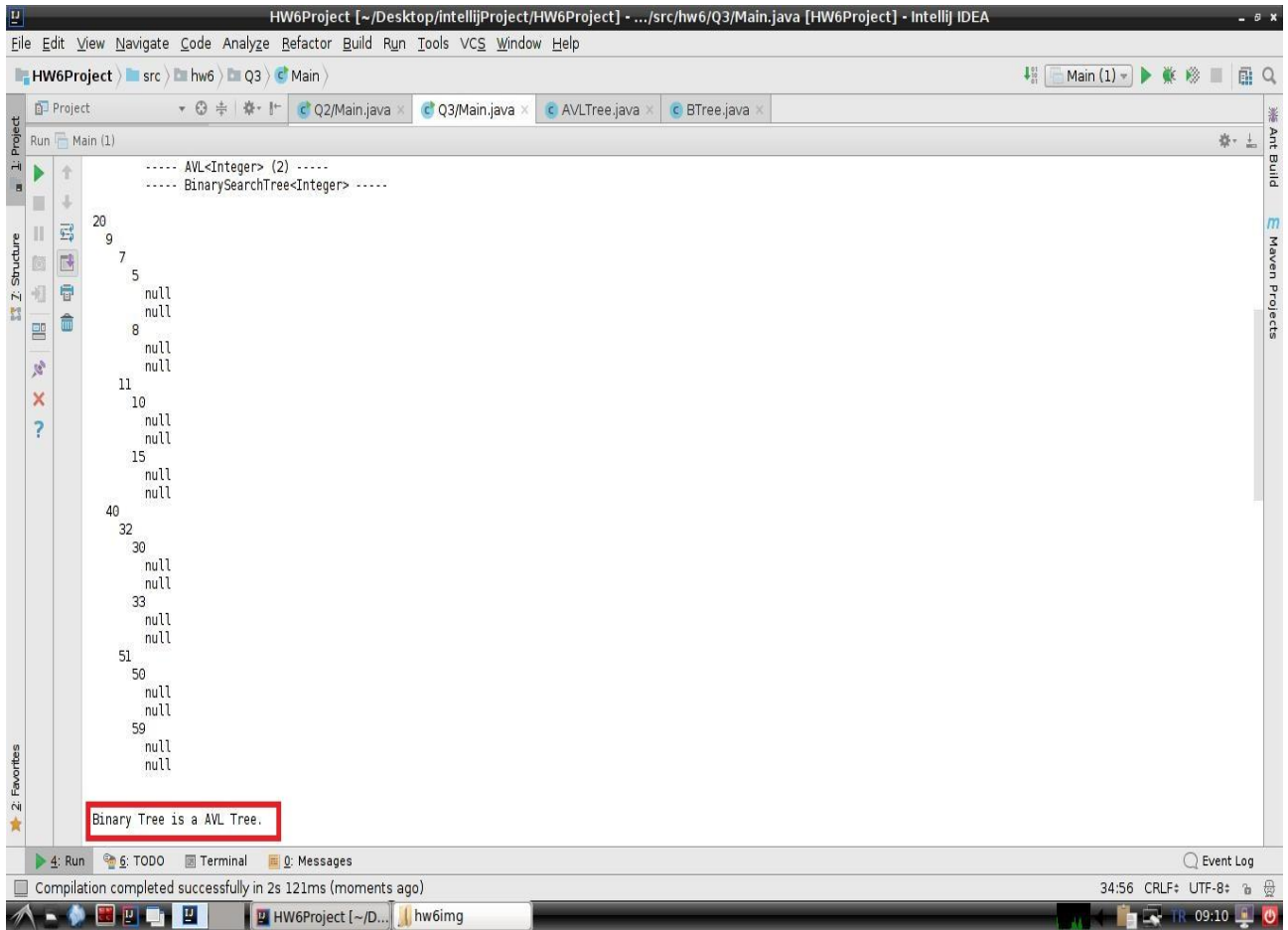
3.3 Running Commands and Results

3.3.1 Q3-> Main.java dosyasında yazılan Test çıktısı 1.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Run Window:** Displays the command `/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...` and the output `----- AVL<Integer> (1) -----`.
- Tree Structure:** A binary tree diagram with root node 10. All child nodes (11 through 20) are labeled 'null'.
- Output:** A red box highlights the text `Binary Tree is not AVL Tree.`
- Status Bar:** Shows `Compilation completed successfully in 10s 954ms (6 minutes ago)` and the time `60:33`.

3.3.2 Q3-> Main.java dosyasında yazılan Test çıktısı 2.



```
----- AVL<Integer> (2) -----
----- BinarySearchTree<Integer> -----

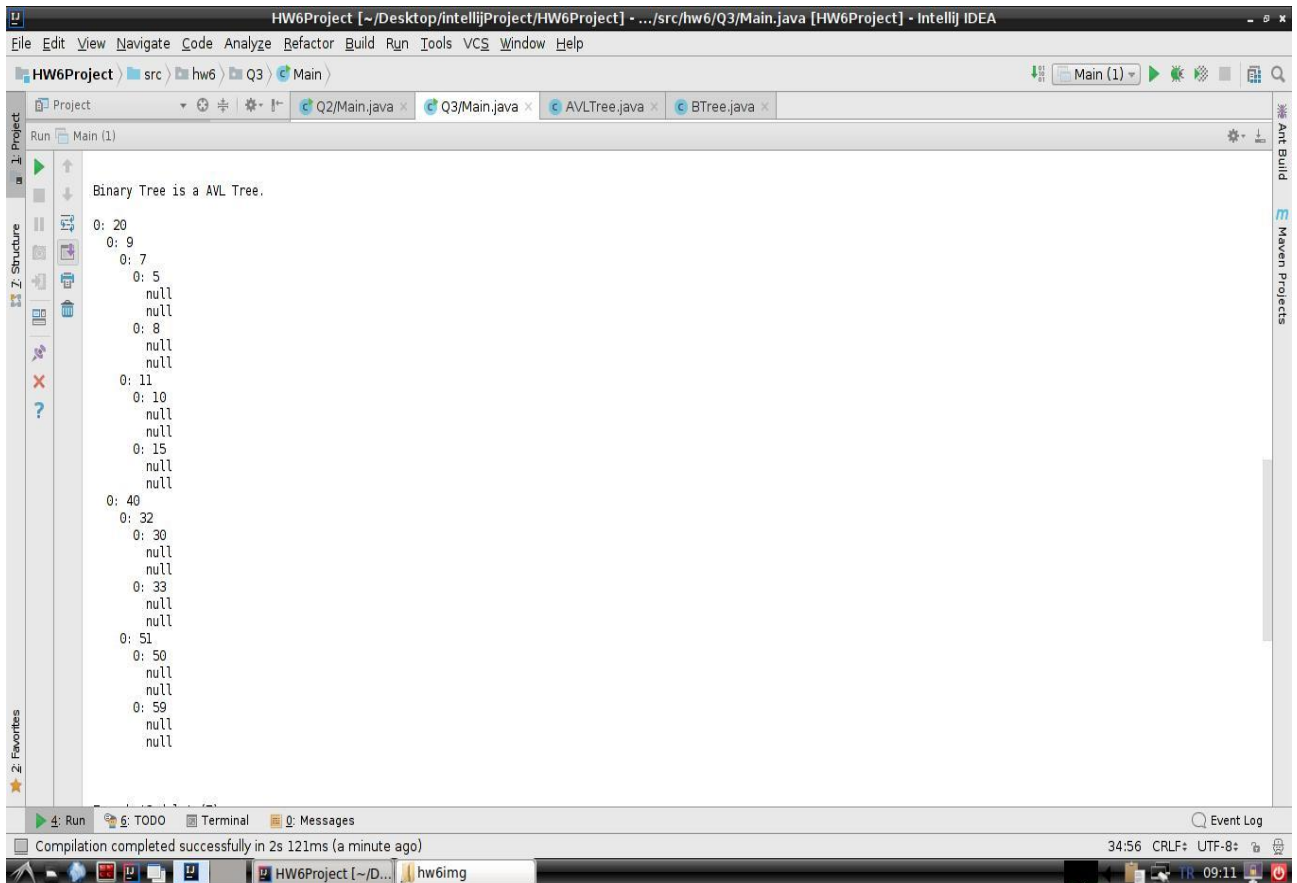
20
 9
  7
   5
    null
    null
   8
    null
    null
  11
   10
    null
    null
   15
    null
    null
 40
 32
 30
  null
  null
 33
  null
  null
 51
 50
  null
  null
 59
  null
  null

Binary Tree is a AVL Tree.
```

4: Run | 5: TODO | Terminal | 0: Messages | Event Log

Compilation completed successfully in 2s 121ms (moments ago) 34:56 CRLF: UTF-8+

HW6Project [~/D... | hw6img



```
Binary Tree is a AVL Tree.

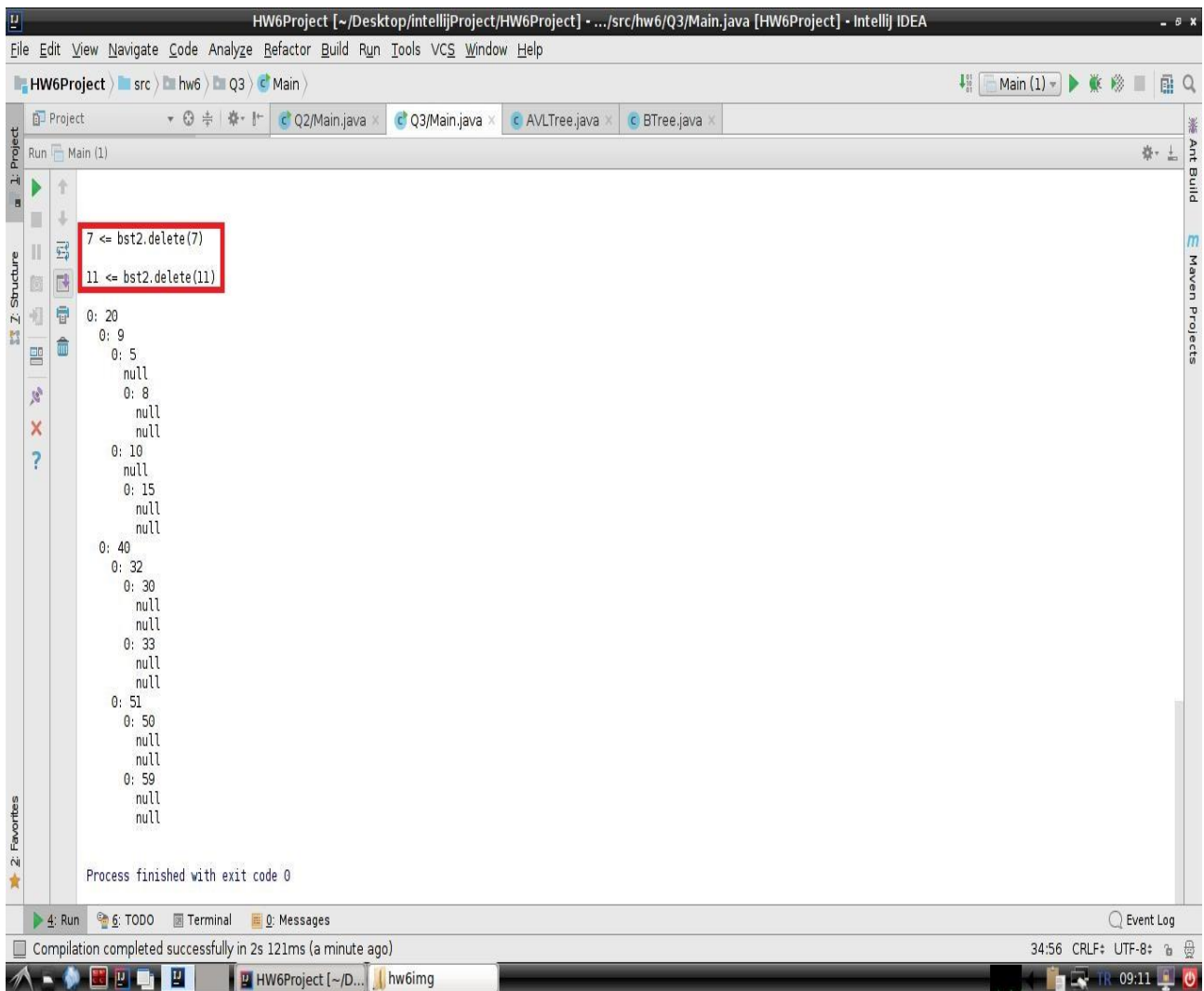
0: 20
 0: 9
   0: 7
    0: 5
     null
     null
    0: 8
     null
     null
   0: 11
    0: 10
     null
     null
    0: 15
     null
     null
  0: 40
   0: 32
    0: 30
     null
     null
    0: 33
     null
     null
  0: 51
   0: 50
     null
     null
   0: 59
     null
     null
```

4: Run | 5: TODO | Terminal | 0: Messages | Event Log

Compilation completed successfully in 2s 121ms (a minute ago) 34:56 CRLF: UTF-8+

HW6Project [~/D... | hw6img

Delete Operation (Removal)



The screenshot shows the IntelliJ IDEA interface with a project named "HW6Project". The main editor displays the file "Main.java" in the "src" directory. The code in the editor is as follows:

```
7 <= bst2.delete(7)
11 <= bst2.delete(11)
```

The output of the program is displayed in the "Run" tab, showing the deletion of nodes 7 and 11 from a binary search tree. The output is as follows:

```
0: 20
0: 9
0: 5
null
0: 8
null
null
0: 10
null
0: 15
null
null
0: 40
0: 32
0: 30
null
null
0: 33
null
null
0: 51
0: 50
null
null
0: 59
null
null
```

The process finished with exit code 0.