GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework # 4

AHMET OKUR
1801042655

1. SYSTEM REQUIREMENTS

Functional Requirement

Heap Structer

1) Heap can search for an elemenet

2) Heap can merge with anathor heap

3) Heap can removing ith largest elemenet from the heap

4) Heap can set the value

BSTHeap

1) BSTHeap can add item

2) BSTHeap can add remove item

3) BSTHeap can add find item occurrancy

4) BSTHeap can find mode

Non-Functional Requirement

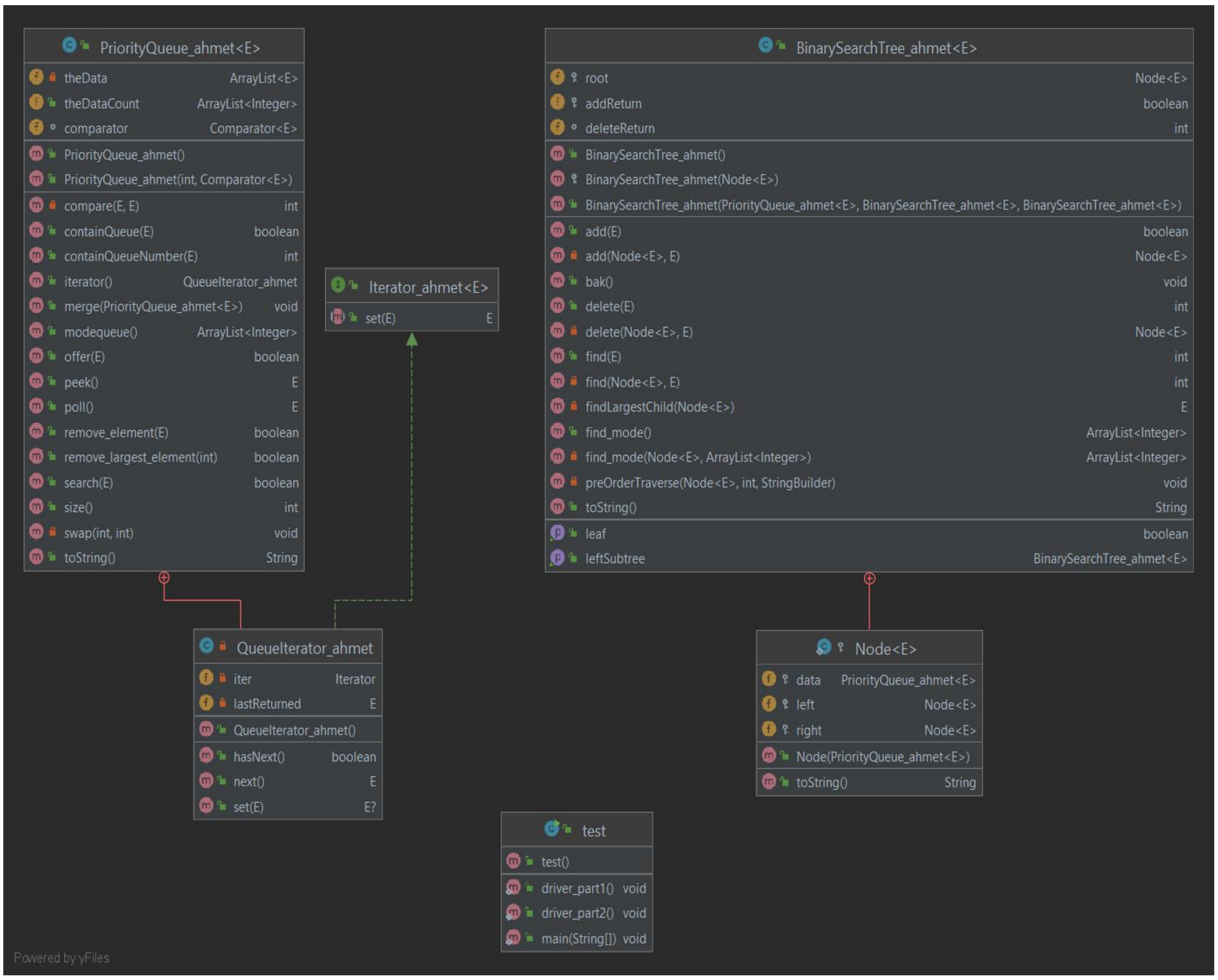
1) Back-end software: Java 11

2) Hardware Interfaces: Mac, Linux and Windows operating systems

3) The system has an internal bug reporting system that keeps track of what types errors happened and when

4) The system holds all the data to keep track of everything available in itself.

2. DIAGRAMS


PriorityQueue_ahmet<E>
theData ArrayList<E>
theDataCount ArrayList<Integer>
comparator Comparator<E>
PriorityQueue_ahmet()
PriorityQueue_ahmet(int, Comparator<E>)
compare(E, E) int
containQueue(E) boolean
containQueueNumber(E) int
iterator() QueueIterator_ahmet
merge(PriorityQueue_ahmet<E>) void
modequeue() ArrayList<Integer>
offer(E) boolean
peek() E
poll() E
remove_element(E) boolean
remove_largest_element(int) boolean
search(E) boolean
size() int
swap(int, int) void
toString() String
Iterator_ahmet<E>
set(E) E
BinarySearchTree_ahmet<E>
root Node<E>
addReturn boolean
deleteReturn int
BinarySearchTree_ahmet()
BinarySearchTree_ahmet(Node<E>)
BinarySearchTree_ahmet(PriorityQueue_ahmet<E>, BinarySearchTree_ahmet<E>, BinarySearchTree_ahmet<E>)
add(E) boolean
add(Node<E>, E) Node<E>
bak() void
delete(E) int
delete(Node<E>, E) Node<E>
find(E) int
find(Node<E>, E) int
findLargestChild(Node<E>) E
find_mode() ArrayList<Integer>
find_mode(Node<E>, ArrayList<Integer>) ArrayList<Integer>
preOrderTraverse(Node<E>, int, StringBuilder) void
toString() String
leaf boolean
leftSubtree BinarySearchTree_ahmet<E>
QueueIterator_ahmet
iter Iterator
lastReturned E
QueueIterator_ahmet()
hasNext() boolean
next() E
set(E) E?
Node<E>
data PriorityQueue_ahmet<E>
left Node<E>
right Node<E>
Node(PriorityQueue_ahmet<E>)
toString() String
test
test()
driver_part1() void
driver_part2() void
main(String[]) void
Powered by yFiles

3. PROBLEM SOLUTION APPROACH

PART 1

I used arraylist for heap. I stored the amount of data in a separate integer array. I also edited the arrays that control the amount of data simultaneously for each data insertion and deletion. This way I can add the same element to the heap twice. I used the contains property of arraylist in the data search method .I did a series of mathematical operations in the merge heap and remove method

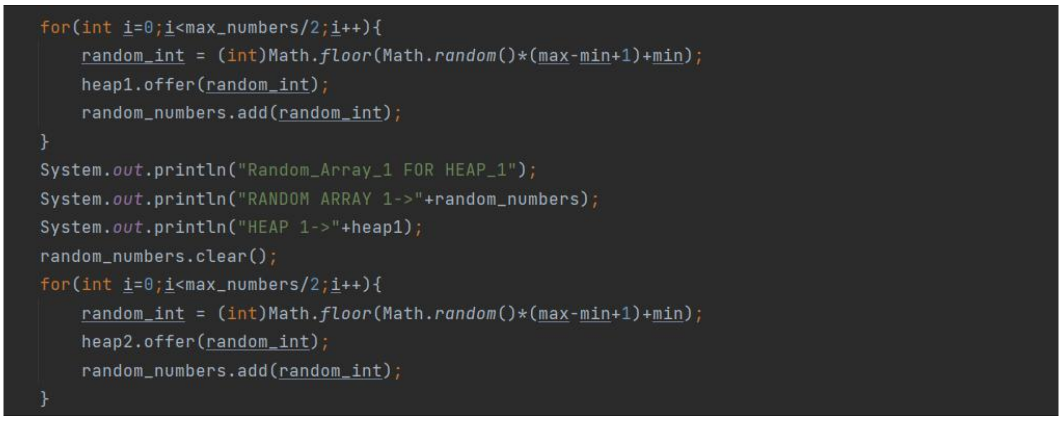
PART 2

I used the structure from part 1 in this section. For the add method, I combined the logic of the BST add method with the Heap's offer method. For the remove method, I combined the logic of the BST remove method with the remove method in the Hapin (a series of mathematical operations have been performed).It is completely navigating the tree for the find method, using the search method in the heap.In the mode method, it traverses the whole tree and returns the most repeating number and number as arraylist.

4. TEST CASES, RUNNING AND RESULTS

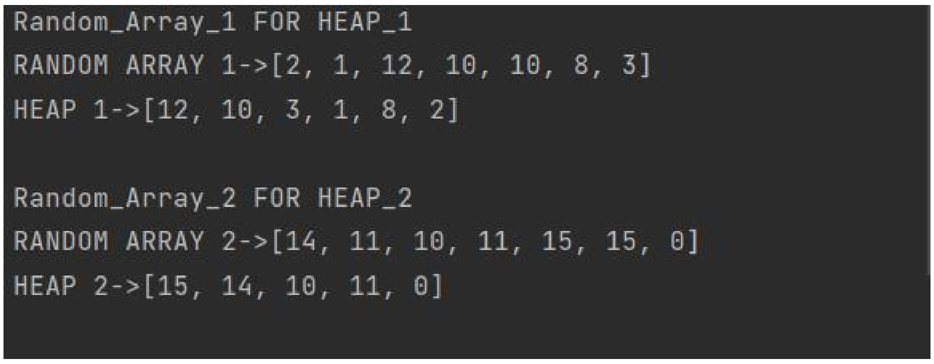
Part 1

I also did arraylist to show what I added (random numbers) to the heap



```
for(int i=0;i<max_numbers/2;i++){
    random_int = (int)Math.floor(Math.random()*(max-min+1)+min);
    heap1.offer(random_int);
    random_numbers.add(random_int);
}
System.out.println("Random_Array_1 FOR HEAP_1");
System.out.println("RANDOM ARRAY 1->"+random_numbers);
System.out.println("HEAP 1->"+heap1);
random_numbers.clear();
for(int i=0;i<max_numbers/2;i++){
    random_int = (int)Math.floor(Math.random()*(max-min+1)+min);
    heap2.offer(random_int);
    random_numbers.add(random_int);
}
```

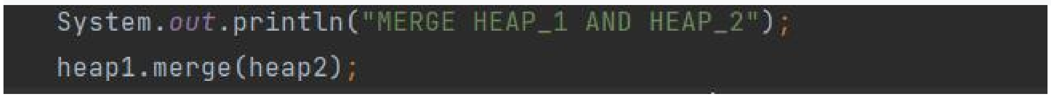
as you can see the heap is working correctly



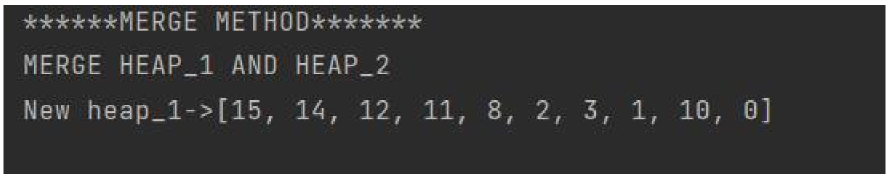
```
Random_Array_1 FOR HEAP_1
RANDOM ARRAY 1->[2, 1, 12, 10, 10, 8, 3]
HEAP 1->[12, 10, 3, 1, 8, 2]

Random_Array_2 FOR HEAP_2
RANDOM ARRAY 2->[14, 11, 10, 11, 15, 15, 0]
HEAP 2->[15, 14, 10, 11, 0]
```

Merge method

I merged the above 2 structures



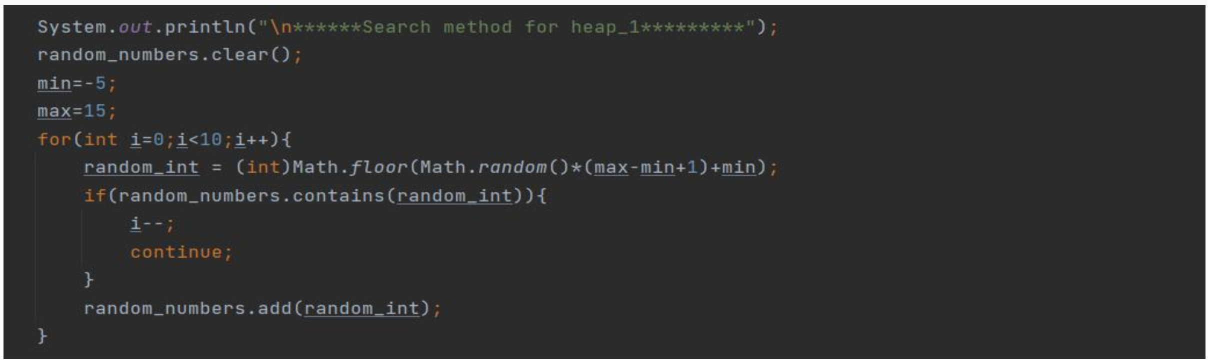
```
System.out.println("MERGE HEAP_1 AND HEAP_2");
heap1.merge(heap2);
```



```
******MERGE METHOD*******
MERGE HEAP_1 AND HEAP_2
New heap_1->[15, 14, 12, 11, 8, 2, 3, 1, 10, 0]
```

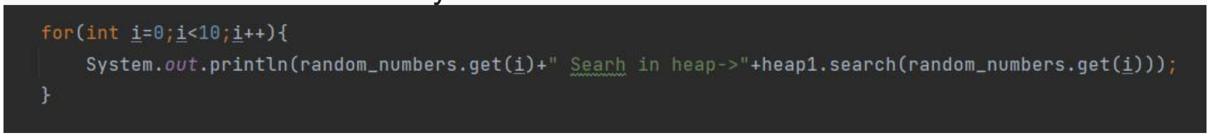
Search method

first create arraylist consisting of random numbers



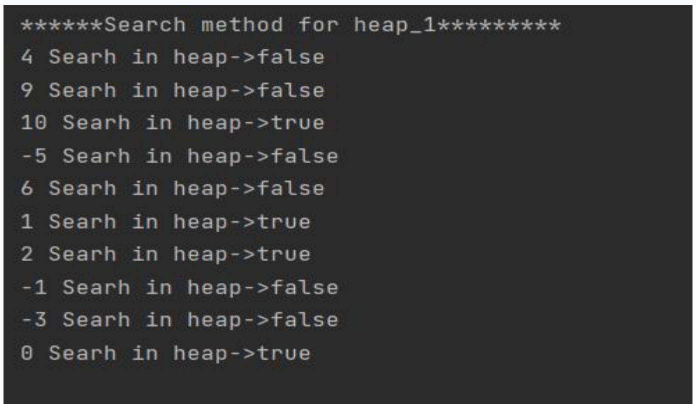
```
System.out.println("\n******Search method for heap_1*********");
random_numbers.clear();
min=-5;
max=15;
for(int i=0;i<10;i++){
    random_int = (int)Math.floor(Math.random()*(max-min+1)+min);
    if(random_numbers.contains(random_int)){
        i--;
        continue;
    }
    random_numbers.add(random_int);
}
```

I searched for numbers in arraylist also in structure



```
for(int i=0;i<10;i++){
    System.out.println(random_numbers.get(i)+" Searh in heap->"+heap1.search(random_numbers.get(i)));
}
```

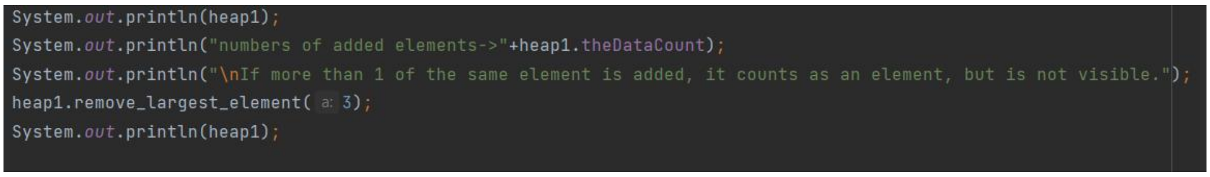
The results of the searched values are below



```
******Search method for heap_1*********
4 Searh in heap->false
9 Searh in heap->false
10 Searh in heap->true
-5 Searh in heap->false
6 Searh in heap->false
1 Searh in heap->true
2 Searh in heap->true
-1 Searh in heap->false
-3 Searh in heap->false
0 Searh in heap->true
```

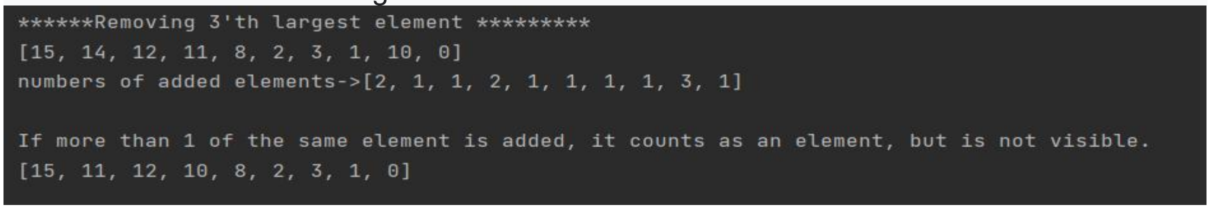
Remove ith largest number

I deleted the 3rd largest element



```
System.out.println(heap1);
System.out.println("numbers of added elements->"+heap1.theDataCount);
System.out.println("\nIf more than 1 of the same element is added, it counts as an element, but is not visible.");
heap1.remove_largest_element( a: 3);
System.out.println(heap1);
```

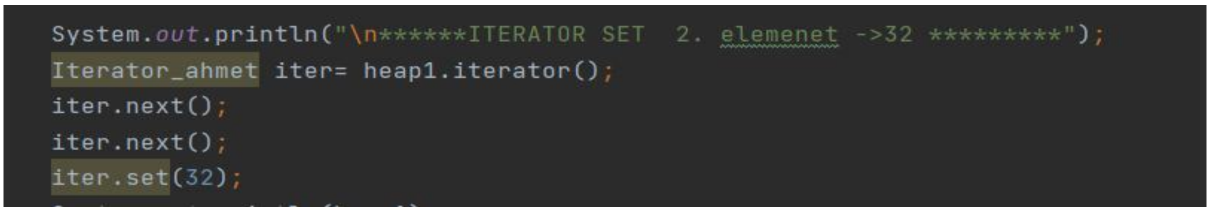
The number 14 is the 3rd largest element since there are 2 numbers 15 in the structure.



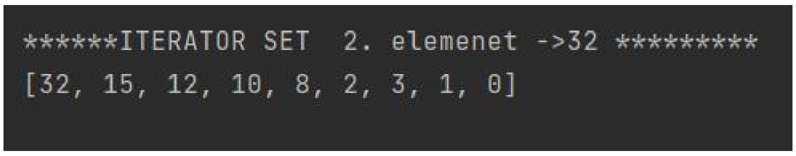
```
******Removing 3'th largest element *********
[15, 14, 12, 11, 8, 2, 3, 1, 10, 0]
numbers of added elements->[2, 1, 1, 2, 1, 1, 1, 1, 3, 1]

If more than 1 of the same element is added, it counts as an element, but is not visible.
[15, 11, 12, 10, 8, 2, 3, 1, 0]
```

Iterator Set

I chose to set element 2



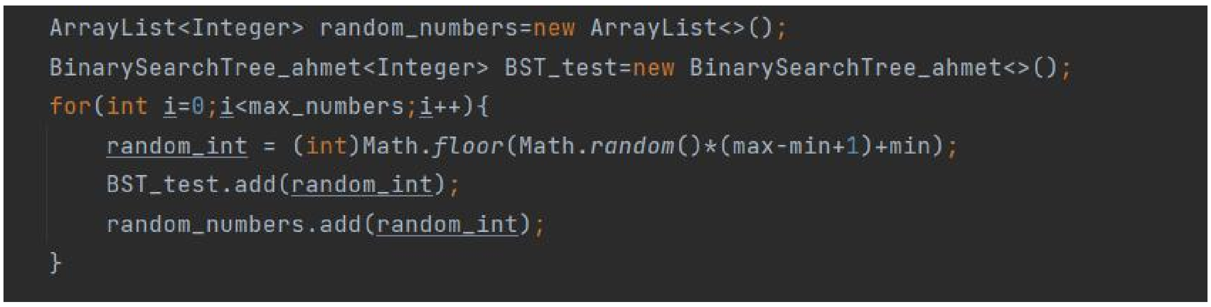
```
System.out.println("\n******ITERATOR SET  2. elemenet ->32 *********");
Iterator_ahmet iter= heap1.iterator();
iter.next();
iter.next();
iter.set(32);
```



```
******ITERATOR SET  2. elemenet ->32 *********
[32, 15, 12, 10, 8, 2, 3, 1, 0]
```

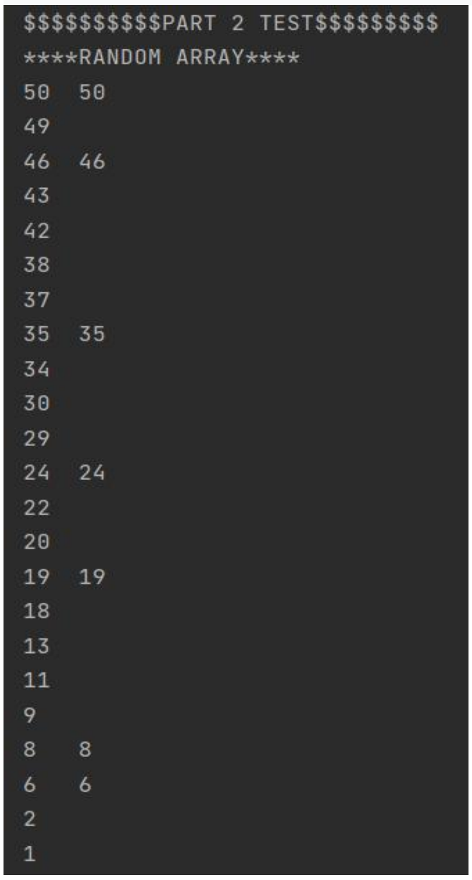
Part 2

I added the random numbers to the tree
I added random numbers to arrayllist and sorted it

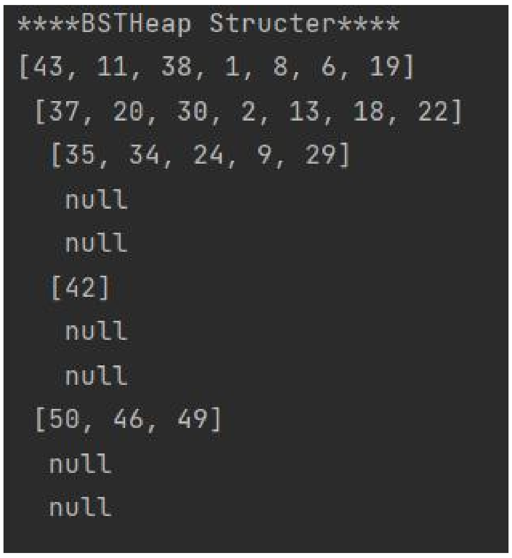


```
ArrayList<Integer> random_numbers=new ArrayList<>();
BinarySearchTree_ahmet<Integer> BST_test=new BinarySearchTree_ahmet<>();
for(int i=0;i<max_numbers;i++){
    random_int = (int)Math.floor(Math.random()*(max-min+1)+min);
    BST_test.add(random_int);
    random_numbers.add(random_int);
}
```

tree and array formed like this



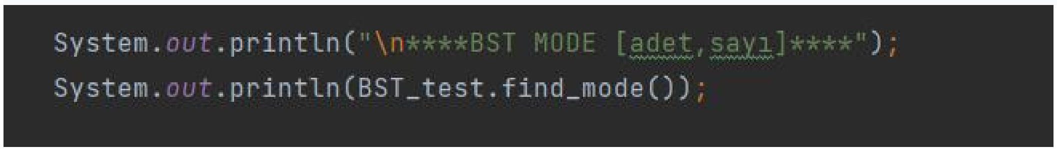
```
$$$$$$$$$$PART 2 TEST$$$$$$$$$
****RANDOM ARRAY****
50  50
49
46  46
43
42
38
37
35  35
34
30
29
24  24
22
20
19  19
18
13
11
9
8   8
6   6
2
1
```



```
****BSTHeap Structer****
[43, 11, 38, 1, 8, 6, 19]
 [37, 20, 30, 2, 13, 18, 22]
  [35, 34, 24, 9, 29]
   null
   null
  [42]
   null
   null
 [50, 46, 49]
  null
  null
```

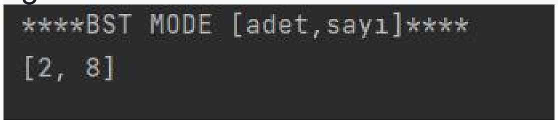
FIND MODE METHOD

function that returns the mode of the tree



```
    System.out.println("\n****BST MODE [adet,sayı]****");
    System.out.println(BST_test.find_mode());
```

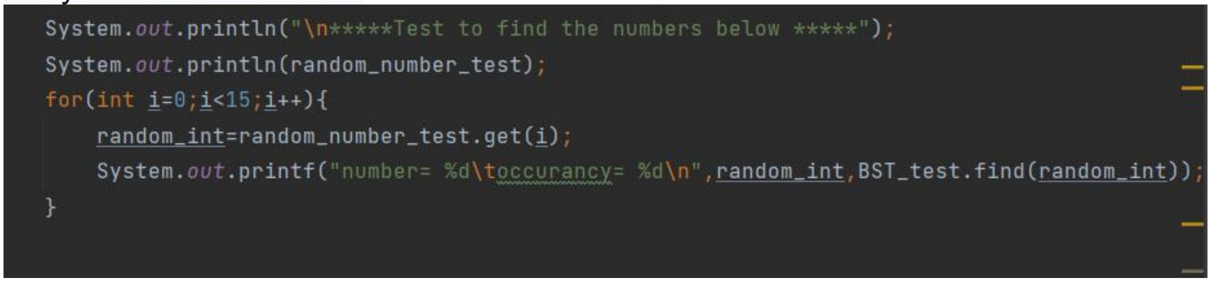
right number mode



```
****BST MODE [adet,sayı]****
[2, 8]
```

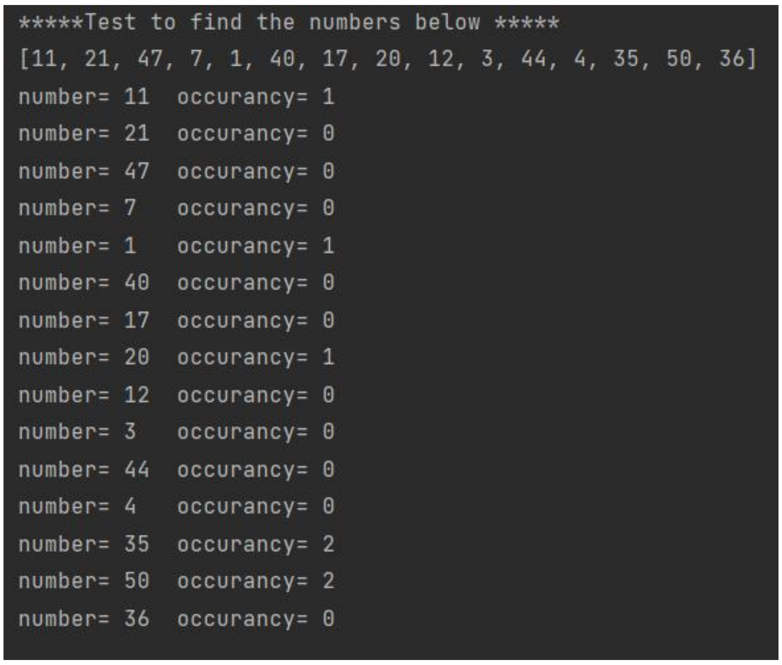
FIND METHOD

array of random numbers



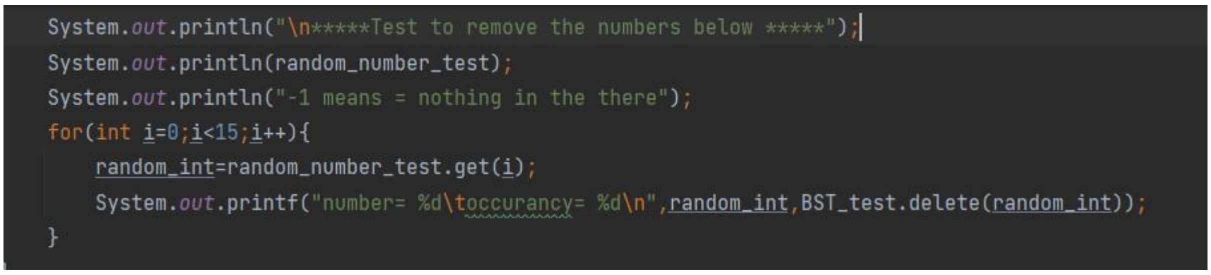
```
System.out.println("\n*****Test to find the numbers below *****");
System.out.println(random_number_test);
for(int i=0;i<15;i++){
    random_int=random_number_test.get(i);
    System.out.printf("number= %d\toccurancy= %d\n",random_int,BST_test.find(random_int));
}
```

The elements of the array consisting of random numbers are checked in the tree with the find method.



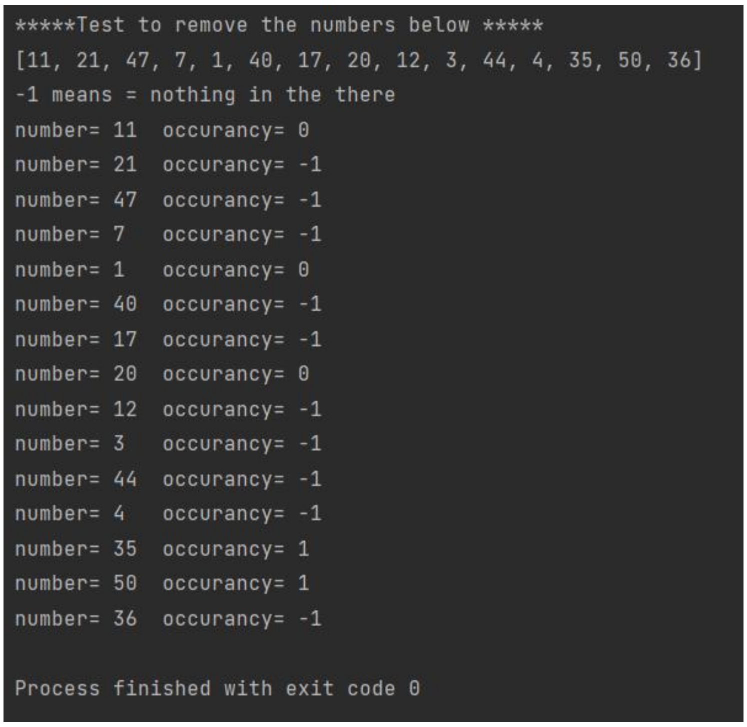
```
*****Test to find the numbers below *****
[11, 21, 47, 7, 1, 40, 17, 20, 12, 3, 44, 4, 35, 50, 36]
number= 11  occurancy= 1
number= 21  occurancy= 0
number= 47  occurancy= 0
number= 7   occurancy= 0
number= 1   occurancy= 1
number= 40  occurancy= 0
number= 17  occurancy= 0
number= 20  occurancy= 1
number= 12  occurancy= 0
number= 3   occurancy= 0
number= 44  occurancy= 0
number= 4   occurancy= 0
number= 35  occurancy= 2
number= 50  occurancy= 2
number= 36  occurancy= 0
```

remove method



```
System.out.println("\n*****Test to remove the numbers below *****");
System.out.println(random_number_test);
System.out.println("-1 means = nothing in the there");
for(int i=0;i<15;i++){
    random_int=random_number_test.get(i);
    System.out.printf("number= %d\toccurancy= %d\n",random_int,BST_test.delete(random_int));
}
```

The areas occupied by the elements in the given array after they are deleted (-1 means not in the tree)



```
*****Test to remove the numbers below *****
[11, 21, 47, 7, 1, 40, 17, 20, 12, 3, 44, 4, 35, 50, 36]
-1 means = nothing in the there
number= 11  occurancy= 0
number= 21  occurancy= -1
number= 47  occurancy= -1
number= 7   occurancy= -1
number= 1   occurancy= 0
number= 40  occurancy= -1
number= 17  occurancy= -1
number= 20  occurancy= 0
number= 12  occurancy= -1
number= 3   occurancy= -1
number= 44  occurancy= -1
number= 4   occurancy= -1
number= 35  occurancy= 1
number= 50  occurancy= 1
number= 36  occurancy= -1

Process finished with exit code 0
```