GTU Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework #4 Report

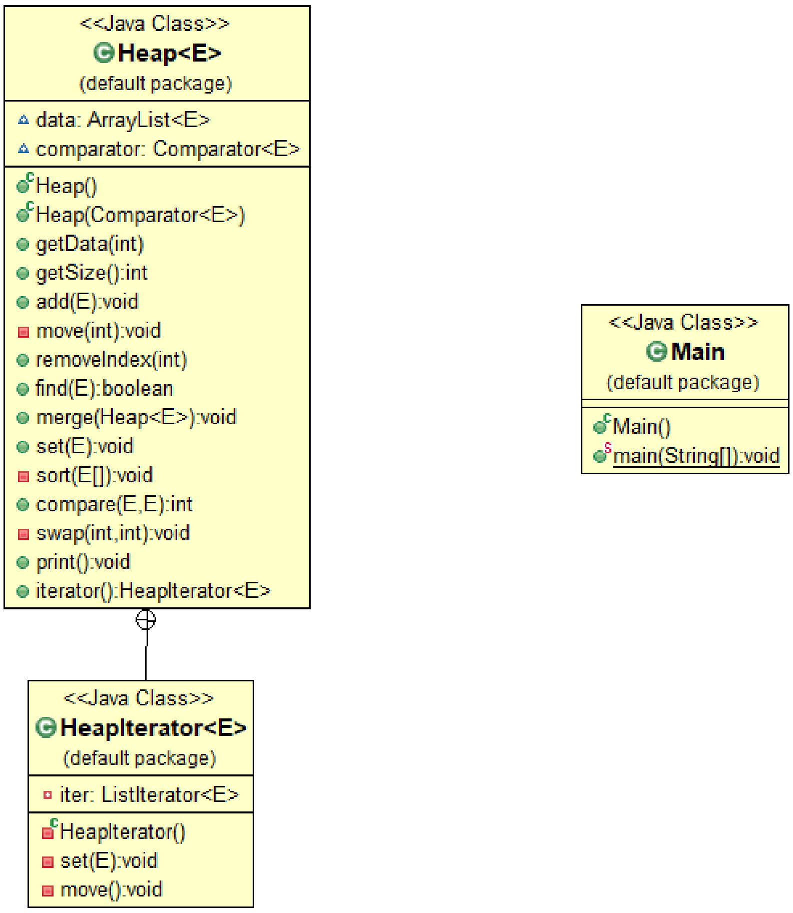
171044076

ŞEYMA NUR CANBAZ

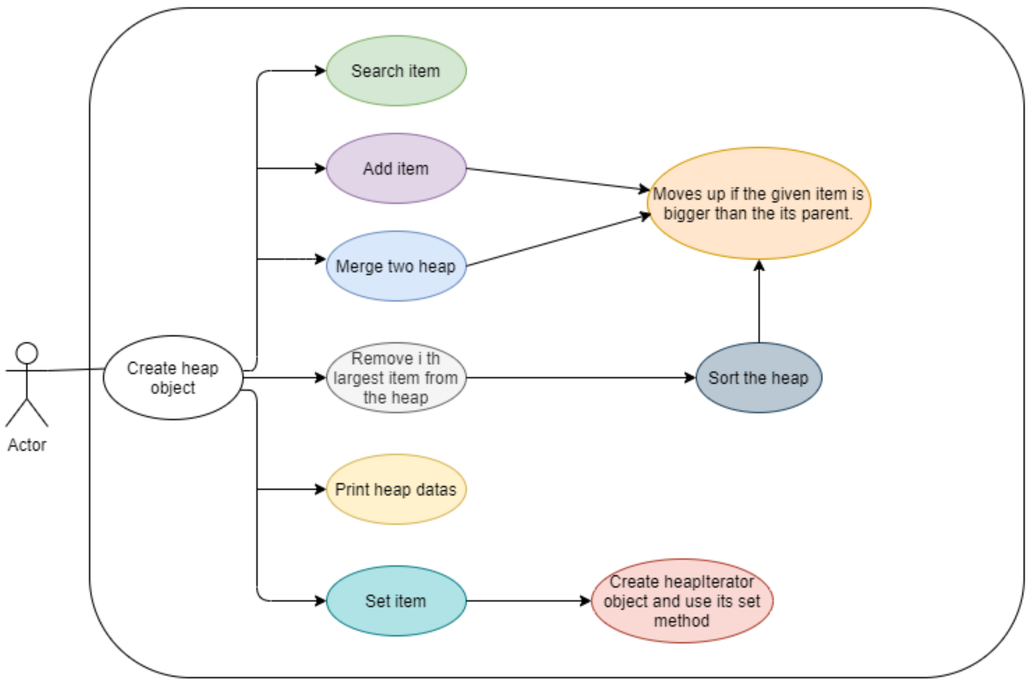
PART1:

1. SYSTEM REQUIREMENTS:
   In this homework, we implement the heap class that includes add, remove ith largest element, search, merge and set (using iterator) methods. There is a need a heap structure for keep the datas. So I created arraylist for keeping the heap data. For set method, we must use the iterator. Since the default iterator class in Java does not support the set method, I created my own class called ‘HeapIterator’. This class provides the sets an item in the heap. The system consists of add, remove, search, merge, set methods and necessary helper methods for them.

2. CLASS DIAGRAM:



3. USE CASE DIAGRAM:


Search item
Add item
Moves up if the given item is bigger than the its parent.
Merge two heap
Actor
Create heap object
Remove i th largest item from the heap
Sort the heap
Print heap datas
Set item
Create heapIterator object and use its set method

4. PROBLEM SOLUTION APPROACH:

Firstly I decided to use maximum heap. Then I created the ‘Heap’ class. In this class, I hold theap data as arraylist. Since the heap class is not comparable, I hold the comparator object. Then I wrote the add method. After add an item in the heap, I reheap. So I wrote the private helper ‘move’ method. This method moves up the data if it is bigger then its parent. So I must compared the two data. For this, I created the compare method that uses the comparator and compare given datas. If the data needs to be swapped its parent, I wrote the private helper method called 'swap'. Then I wrote the remove ith largest element method. For this method, I must found the largest ith element in the heap. So, I created a tempArray and I copied of the heap to this array. Then I created the private helper method called ‘sort’, this method sorts the given array from largest to smallest. After I found the largest ith element, I found their index and I removed it in the heap. Then I called the ‘move’ method for reheap. If given index is the greater than the size of the heap, I throwed ‘IndexOutOfBoundsException’. Then wrote the ‘merge’ method. This method merged the two heap. Then I wrote the ‘find’ method for searching an element in the heap. If the given element is in the heap returns true, otherwise returns false. Then I wrote the set method. For this, I must use the iterator. But the default iterator class in Java does not support the set method, so I created my own class called ‘HeapIterator’. In this class I created the arrayList iterator and I used this iterator for ‘next’ and ‘hasNext’ method. I setted the given element is in last of the heap and then I created the private helper ‘move’ method in this class. This method same as ‘move’ method in ‘Heap’ class. Then I wrote the ‘print’ method for the printing the heap.

5. TEST CASES:

1. Add an item:

Test Steps:

1. Write the data.

Expected Result: The data added in the heap.

Actual Result: As expected.

2. Add the same item:

Test Steps:

1. Write the same data that already in the heap.

Expected Result: The data added in the heap.

Actual Result: As expected.

3. Delete the largest i th item:

Test Steps:

1. Write down which largest number will be deleted.

Expected Result: The data is deleted.

Actual Result: As expected.

4. Delete the largest i th item (out of bounds):

Test Steps:

1. Write the index that larger than the heap size.

Expected Result: Throws ‘indexOutofBoundsException’.

Actual Result: As expected.

5. Merge two heap:

Test Steps:

1. Create the another heap object.
2. Add items in heap.
3. Call the merge method.

Expected Result: Two heaps are merged.

Actual Result: As expected.

6. Find the given item in the heap:

Test Steps:

1. Write the data that already added in the heap.

Expected Result: ‘Yes’.

Actual Result: As expected.

7. Find the given item in the heap (if it is not in the heap):

Test Steps:

1. Write a data that is not in the heap.

Expected Result: ‘No’.

Actual Result: As expected.

8. Set an item:

Test Steps:

1. Write the data to be setted and call the set method.

Expected Result: The data is setted in the heap.

Actual Result: As expected.

9. Set the same item:

Test Steps:

1. Write the same data to be setted and call the set method.

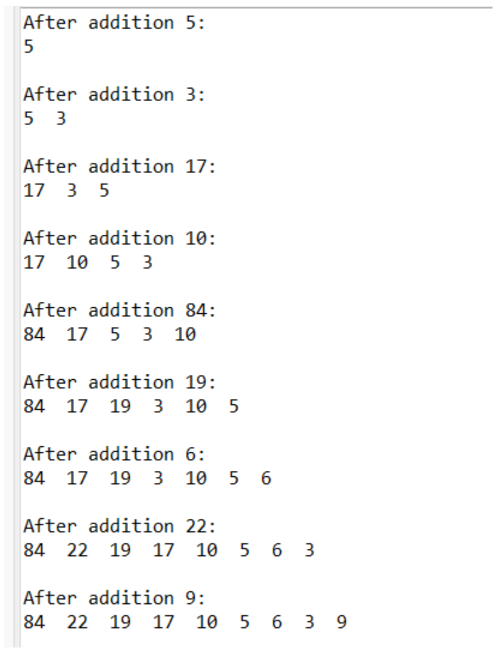
Expected Result: The data is setted in the heap.

Actual Result: As expected.

6. RUNNING COMMAND AND RESULTS:

1.Add an item:

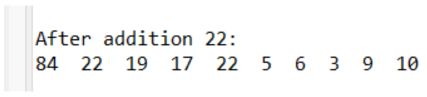
Add 5,3,17,10,84,19,6,22,9



```
After addition 5:
5

After addition 3:
5  3

After addition 17:
17  3  5

After addition 10:
17  10  5  3

After addition 84:
84  17  5  3  10

After addition 19:
84  17  19  3  10  5

After addition 6:
84  17  19  3  10  5  6

After addition 22:
84  22  19  17  10  5  6  3

After addition 9:
84  22  19  17  10  5  6  3  9
```

2.Add the same item:

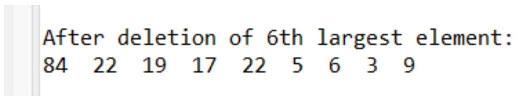
Add 22 again



```
After addition 22:
84  22  19  17  22  5  6  3  9  10
```

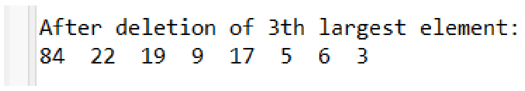
3.Delete the largest i th item:

Delete 6th largest element (data: 10)



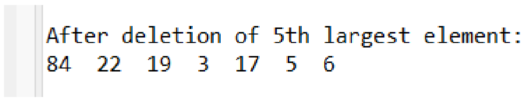
```
After deletion of 6th largest element:
84  22  19  17  22  5  6  3  9
```

Delete 3rd largest element (data: 22)



```
After deletion of 3th largest element:
84  22  19  9  17  5  6  3
```

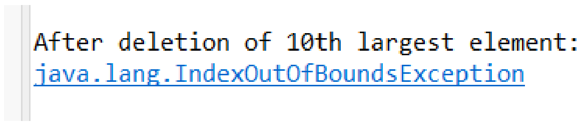
Delete 5th largest element (data: 9)



```
After deletion of 5th largest element:
84  22  19  3  17  5  6
```

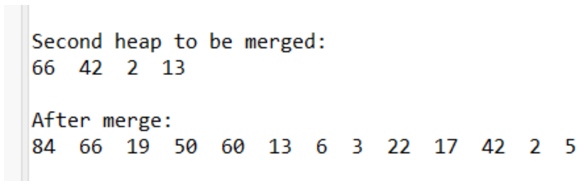
4.Delete the largest i th item (out of bounds):

Delete 10th largest element



```
After deletion of 10th largest element:
java.lang.IndexOutOfBoundsException
```

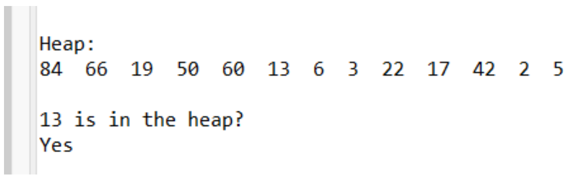
5.Merge two heap:



```
Second heap to be merged:
66  42  2  13

After merge:
84  66  19  50  60  13  6  3  22  17  42  2  5
```

6.Find the given item in the heap:

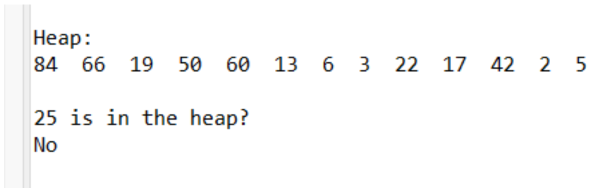
Find 13



```
Heap:
84  66  19  50  60  13  6  3  22  17  42  2  5

13 is in the heap?
Yes
```

7.Find the given item in the heap (if it is not in the heap):

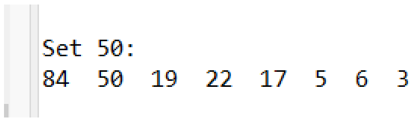
Find 25



```
Heap:
84  66  19  50  60  13  6  3  22  17  42  2  5

25 is in the heap?
No
```

8.Set an item:

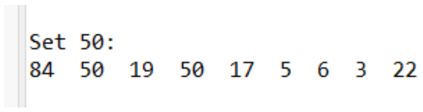
Set 50



```
Set 50:
84  50  19  22  17  5  6  3
```

9.Set the same item:

Set 50 again

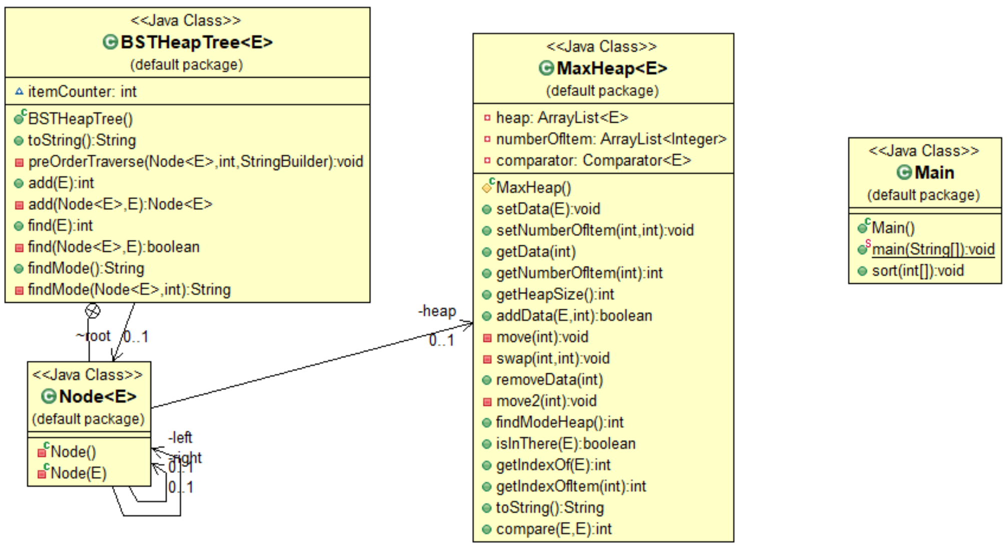


```
Set 50:
84  50  19  50  17  5  6  3  22
```

PART 2:

1. SYSTEM REQUIREMENTS:
   In this homework, we created the BSTHeapTree. So I needed the node class for hold the root and its child. Each root have the maximum heap. And in maximum heap, I should hold the number of occurrences of the data. So I created the MaxHeap Class and I hold the nodes as MaxHeap. In MaxHeap class I hold the heap data as arraylist and I hold the number of occurrences as arraylist. I should implemet the add, remove, find method for heap nodes for using add, remove, find in the BSTHeapTree class.

2. CLASS DIAGRAM:



3. PROBLEM SOLUTION APPROACH:

Firstly, I created BSTHeapTree class. In this tree, each node should be maximum heap. So I created the 'MaxHeap' class. In 'MaxHeap' class I created 'heap' as arrayList and 'numberOfItem' as arrayList. I keep the data and number of occurences of the data information is seperately. 'heap' keeps the data and 'numberOfItem' keeps the number of occurrence of the each data in the heap. (This class is very similar to the heap class in the part1). When an element is added, I increased the number of occurrence in that index by finding the index of the added element. Likewise, I reduced the number of occurrences by taking the index of the number I deleted in the remove method. In 'BSTHeapTree' class, I created the add method. This method fistly checks the item is in the tree or not. If is in the tree, then increases the number of occurences of this data. Else checks the size of the heap in node. If the node size is less than 7, then adds the data in this node. Otherwise, creates the new node and adds the data. Then I created the find method. This method finds the number of occurrences the given data. Then I createed the findMode method. This method finds the greates number of occurrences of data in all tree. So I traversed the tree as the preorder and I found the largest occurrences. I think remove method as 'if the given item is in this root and number of occurrence greater than 1, decrease the its number of occurrence. Else if the given item is in this root and number of occurrence is 0, delete this item and reheap (uses maxHeap remove). After removal, if current root is null, remove it' but I could not make it in time.

4. TEST CASES:

1. Add an item:

Test Steps:

1. Write the data.

Expected Result: The data added in the tree.

Actual Result: As expected.

2. Add the same item:

Test Steps:

1. Write the same data that already in the tree.

Expected Result: Number of occurrence of data is increased.

Actual Result: As expected.

3. Find the item:

Test Steps:

1. Write the data that already added in the tree.

Expected Result: Data and the number of occurrences of data is printed.

Actual Result: As expected.

4. Find the item (not in tree):

Test Steps:

1. Write a data that is not in the tree.

Expected Result: Printed the number of occurrences of data is zero.

Actual Result: As expected.

5. Find modes:

Test Steps:

1. Call the findMode() method.

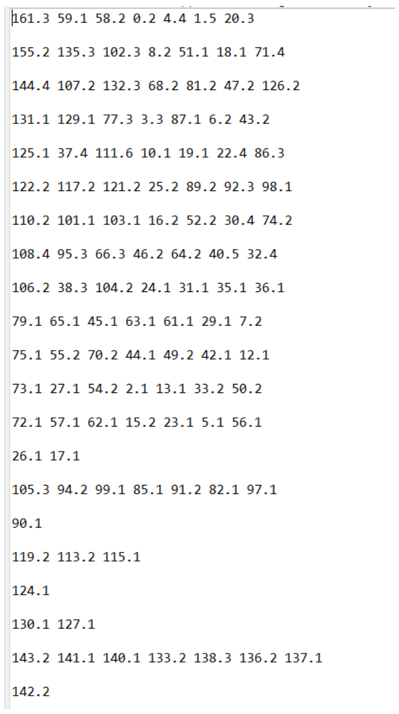
Expected Result: The largest number of occurrences of data in tree is printed.

Actual Result: As expected.

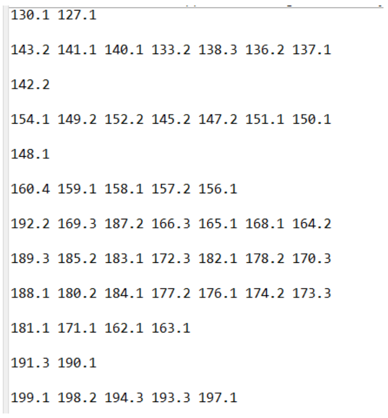
5. RUNNING COMMAND AND RESULTS:

*I tested my code with 30 (0-200 range) data to get an easier screenshot.

1. Add items/same items:

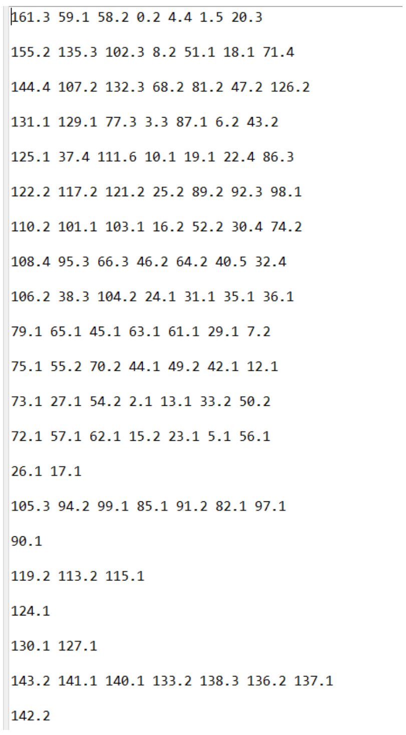


```
161.3 59.1 58.2 0.2 4.4 1.5 20.3

155.2 135.3 102.3 8.2 51.1 18.1 71.4

144.4 107.2 132.3 68.2 81.2 47.2 126.2

131.1 129.1 77.3 3.3 87.1 6.2 43.2

125.1 37.4 111.6 10.1 19.1 22.4 86.3

122.2 117.2 121.2 25.2 89.2 92.3 98.1

110.2 101.1 103.1 16.2 52.2 30.4 74.2

108.4 95.3 66.3 46.2 64.2 40.5 32.4

106.2 38.3 104.2 24.1 31.1 35.1 36.1

79.1 65.1 45.1 63.1 61.1 29.1 7.2

75.1 55.2 70.2 44.1 49.2 42.1 12.1

73.1 27.1 54.2 2.1 13.1 33.2 50.2

72.1 57.1 62.1 15.2 23.1 5.1 56.1

26.1 17.1

105.3 94.2 99.1 85.1 91.2 82.1 97.1

90.1

119.2 113.2 115.1

124.1

130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2
```

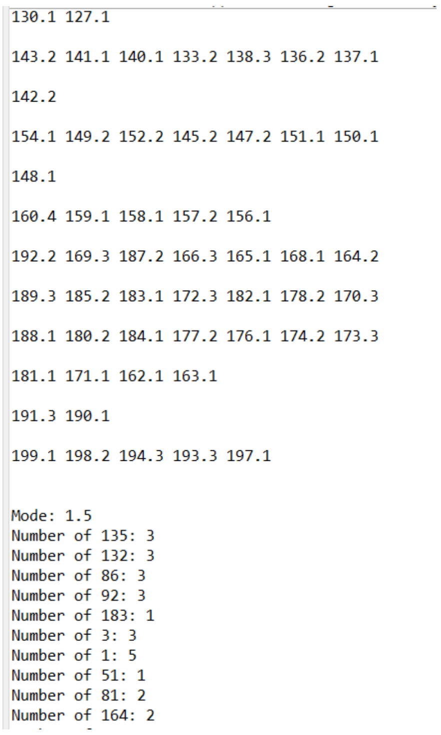


```
130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2

154.1 149.2 152.2 145.2 147.2 151.1 150.1

148.1

160.4 159.1 158.1 157.2 156.1

192.2 169.3 187.2 166.3 165.1 168.1 164.2

189.3 185.2 183.1 172.3 182.1 178.2 170.3

188.1 180.2 184.1 177.2 176.1 174.2 173.3

181.1 171.1 162.1 163.1

191.3 190.1

199.1 198.2 194.3 193.3 197.1
```

2. Find the item:

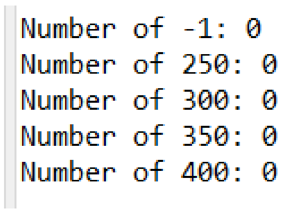


```
161.3 59.1 58.2 0.2 4.4 1.5 20.3

155.2 135.3 102.3 8.2 51.1 18.1 71.4

144.4 107.2 132.3 68.2 81.2 47.2 126.2

131.1 129.1 77.3 3.3 87.1 6.2 43.2

125.1 37.4 111.6 10.1 19.1 22.4 86.3

122.2 117.2 121.2 25.2 89.2 92.3 98.1

110.2 101.1 103.1 16.2 52.2 30.4 74.2

108.4 95.3 66.3 46.2 64.2 40.5 32.4

106.2 38.3 104.2 24.1 31.1 35.1 36.1

79.1 65.1 45.1 63.1 61.1 29.1 7.2

75.1 55.2 70.2 44.1 49.2 42.1 12.1

73.1 27.1 54.2 2.1 13.1 33.2 50.2

72.1 57.1 62.1 15.2 23.1 5.1 56.1

26.1 17.1

105.3 94.2 99.1 85.1 91.2 82.1 97.1

90.1

119.2 113.2 115.1

124.1

130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2
```



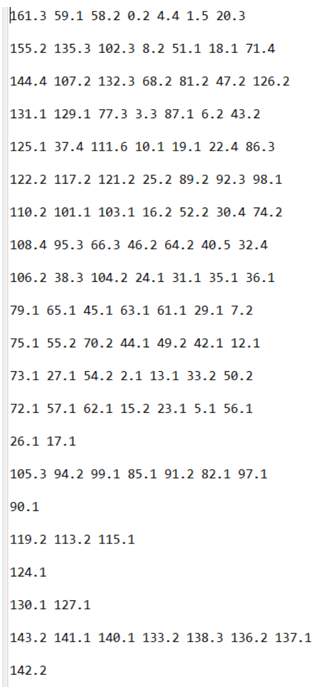
```
130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2

154.1 149.2 152.2 145.2 147.2 151.1 150.1

148.1

160.4 159.1 158.1 157.2 156.1

192.2 169.3 187.2 166.3 165.1 168.1 164.2

189.3 185.2 183.1 172.3 182.1 178.2 170.3

188.1 180.2 184.1 177.2 176.1 174.2 173.3

181.1 171.1 162.1 163.1

191.3 190.1

199.1 198.2 194.3 193.3 197.1


Mode: 1.5
Number of 135: 3
Number of 132: 3
Number of 86: 3
Number of 92: 3
Number of 183: 1
Number of 3: 3
Number of 1: 5
Number of 51: 1
Number of 81: 2
Number of 164: 2
```

3. Find the item (not in the tree):

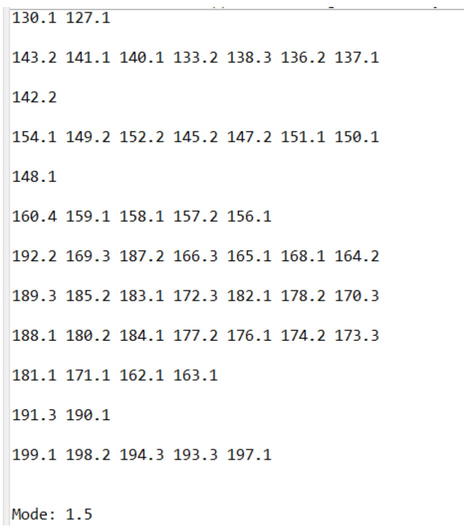


```
Number of -1: 0
Number of 250: 0
Number of 300: 0
Number of 350: 0
Number of 400: 0
```

4. Find the mode:



```
161.3 59.1 58.2 0.2 4.4 1.5 20.3

155.2 135.3 102.3 8.2 51.1 18.1 71.4

144.4 107.2 132.3 68.2 81.2 47.2 126.2

131.1 129.1 77.3 3.3 87.1 6.2 43.2

125.1 37.4 111.6 10.1 19.1 22.4 86.3

122.2 117.2 121.2 25.2 89.2 92.3 98.1

110.2 101.1 103.1 16.2 52.2 30.4 74.2

108.4 95.3 66.3 46.2 64.2 40.5 32.4

106.2 38.3 104.2 24.1 31.1 35.1 36.1

79.1 65.1 45.1 63.1 61.1 29.1 7.2

75.1 55.2 70.2 44.1 49.2 42.1 12.1

73.1 27.1 54.2 2.1 13.1 33.2 50.2

72.1 57.1 62.1 15.2 23.1 5.1 56.1

26.1 17.1

105.3 94.2 99.1 85.1 91.2 82.1 97.1

90.1

119.2 113.2 115.1

124.1

130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2
```



```
130.1 127.1

143.2 141.1 140.1 133.2 138.3 136.2 137.1

142.2

154.1 149.2 152.2 145.2 147.2 151.1 150.1

148.1

160.4 159.1 158.1 157.2 156.1

192.2 169.3 187.2 166.3 165.1 168.1 164.2

189.3 185.2 183.1 172.3 182.1 178.2 170.3

188.1 180.2 184.1 177.2 176.1 174.2 173.3

181.1 171.1 162.1 163.1

191.3 190.1

199.1 198.2 194.3 193.3 197.1


Mode: 1.5
```