**GTU Department of Computer Engineering**

**CSE 222/505 - Spring 2021**

**Homework 4# Report**

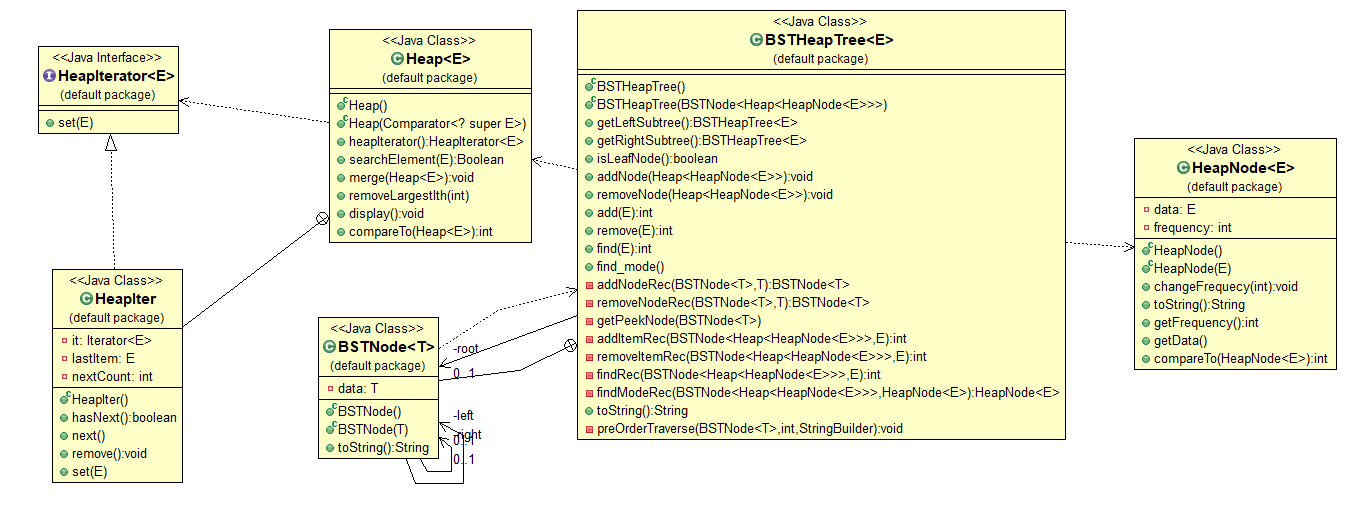
**Sena Nur Ulukaya**

**1901042622**

1. **SYSTEM REQUIREMENTS**
   1. **Non-Functional Requirements**

* Java Runtime Environment to run code.
* Some memory to test the program. (50-100KB)
* Java as programming language.
  1. **Functional Requirements**
* I used PriorityQueue represent Heap, since Heap extends PriorityQueue it has all the functionality of PriorityQueue. (add, remove, contains etc.)
* Searching element in Heap.
* Merging a heap with another heap.
* Removing largest element of the heap. 1st largest element of the heap is largest element, sizeth largest element of the heap is smallest element.
* Displaying heap.
* Heap is comparable.
* HeapIterator with extended set method to set heap elements with specified element while iterating.
* Adding item to the BSTHeapTree.
* Removing item from the BSTHeapTree.
* Finding frequency of each data of the BSTHeapTree.
* Finding mode of the BSTHeapTree.

1. **USE CASE AND CLASS DIAGRAMS**



1. **PROBLEM SOLUTION APPROACH**

Firstly, for part 1, I have used PriorityQueue to represent my Heap class. I have extended it. Also, I have created a interface, which extends Iterator, with a new set method. Then I used my interface to create inner class HeapIter and added my new feature there.

To merging I have simply used feature of the PriorityQueue addAll. You can’t merge a Heap with itself, it will give exception.

To search element, I iterated with my iterator and search the whole heap.

To remove largest ith element, I made an array from my Heap and then I sorted it.

It helped me to easily find the index of the element. Because they’re already in increasing order. My range is between 1-size. Because 0th largest element means nothing. 1st largest element is largest element and sizeth largest element is smallest element.

Secondly, for part 2, I have created BSTNode class to represent BST nodes and HeapNodes to represent Heap nodes with a data and a frequency. My Heaps are the same with part 1 but I make it maxheap with the comparator.

I have HeapNodes of Heaps of BSTNodes to my class to represent our structures. Basically, BST has BSTNodes in it and BSTNodes has Heap in it and Heap has HeapNodes with a data and frequency.

I have used classic BST methods to add and remove nodes. (The ones from the class.)

I compared the Heap’s peek for traversing and building the structure.

I have used this algorithm to add item. Firstly, I tried to find if item exist before, if it is I just increased frequency. If it does not, I have tried to find a not-full node recursively for it within the BST structure suitable.

I have used this algorithm to remove item. Firstly, If there’s more than one of that item I just decreased the frequency. But if there’s only 1, then I find it recursively and then remove it. But since removing occurs emptiness in the Heap and I don’t want it for the not-leaf Heaps, I removed the Heap and add its element one by one again. By doing this, I add the items to the proper places and prevent the emptiness. Exception if element does not exist.

For find method, again I traversed nodes recursively and search the inside of the heaps.

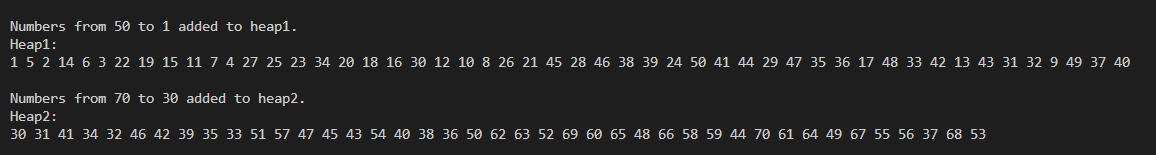
If I can find the item, I returned its frequency, else exception.

For mode method, I have find the most occurred element of the heap and take it. I have compare it with its right and left and I get the biggest frequency each time. After recursively traversing each heap, I returned mode if it does exist and null if it does not.

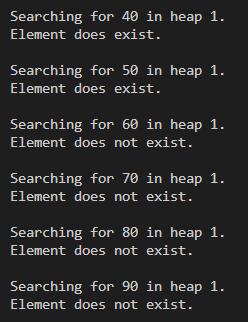
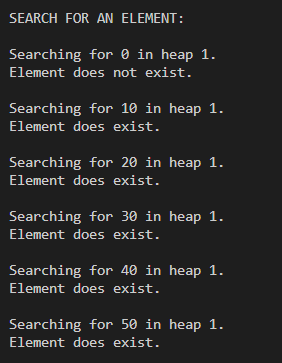
1. **TEST CASES AND RUNNING AND RESULTS**

**PART1:**

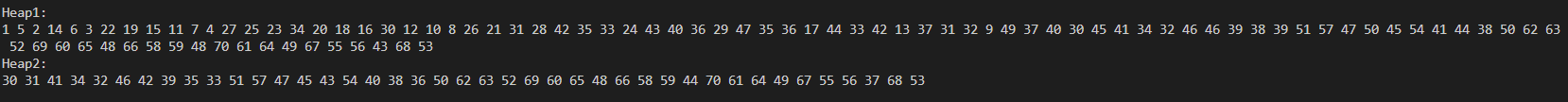
* I created 2 heaps and add some numbers to them.

****

* I search for some exist and some non-exist element.

****

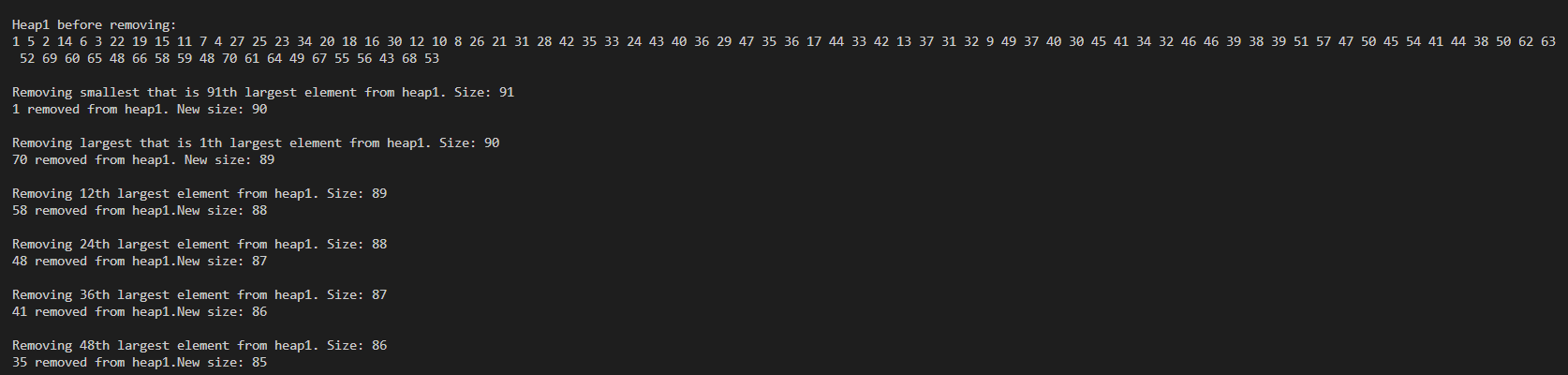
* I merged Heap2 in Heap1.

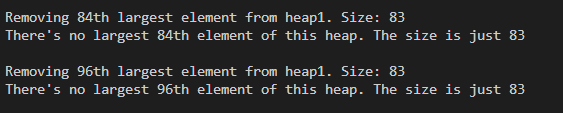
****

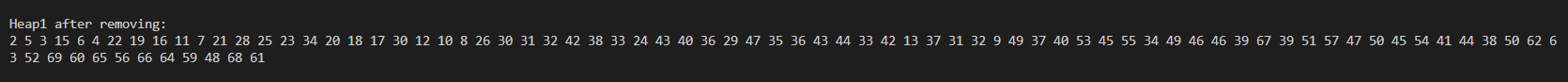
* I merged heap2 with itself, it will give error.

****

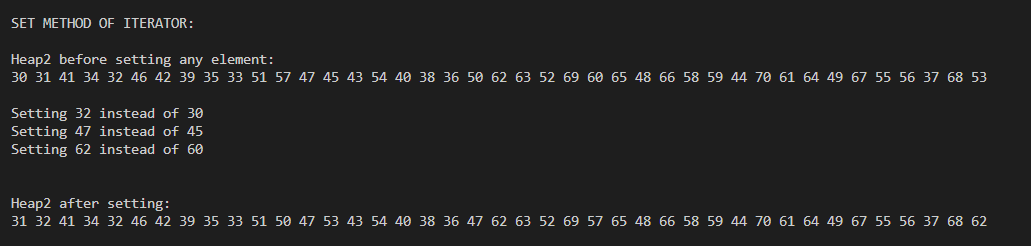
* I removed ith largest elements from Heap1.



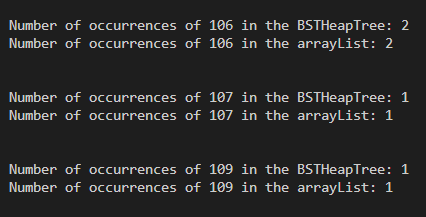
****

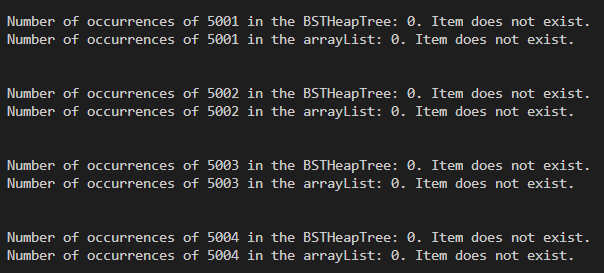
****

* I used my iterator set method.



**PART2:**

* 3000 numbers within range 5000 added to the BSTHeapTree. (You can print them but I couldn’t because it’ll take a lot of place in terminal.)
* Frequency Test:
* Search for 100 numbers in the array, some of it is: 
* 10 numbers not in the array:

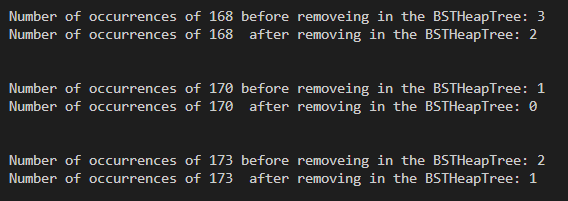


* Find the mode of the BSTHeapTree: I have returned array of Modes in arrayList because there can be more than one and it can be different from BSTHeapTree’s.

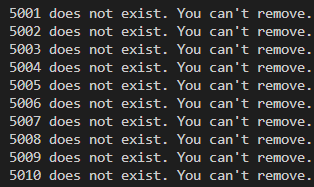




* Remove 100 numbers in the array:

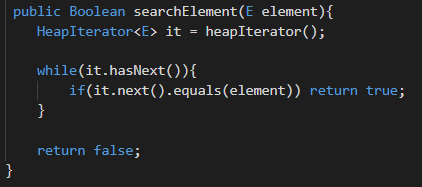


* 10 numbers not in the array:

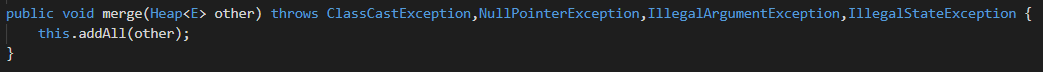


1. **PART3: TIME COMPLEXITY ANALYZE**

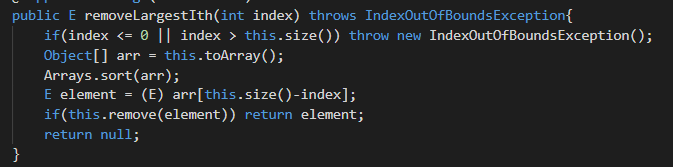
* Heap Class



* Tw(n) = θ(n), because if element does not exist it will loop n times.
* Tb(n) = θ(1), it can be first element.
* T(n) = O(n)

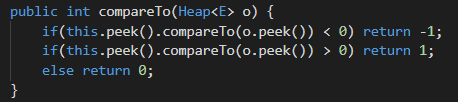


* addAll’s complexity is O(mlogn). m is the size of other Heap.
* T(n) = O(m.logn).

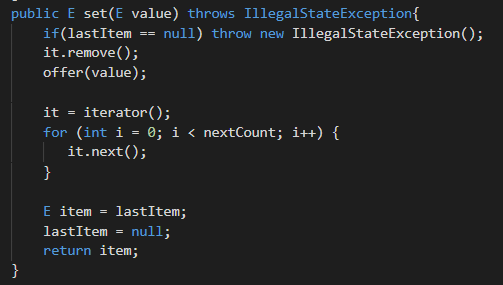


toArray is θ(n) and sort θ(nlogn) and removing is O(logn) since finding the item takes n generally and deleting it takes logn times.

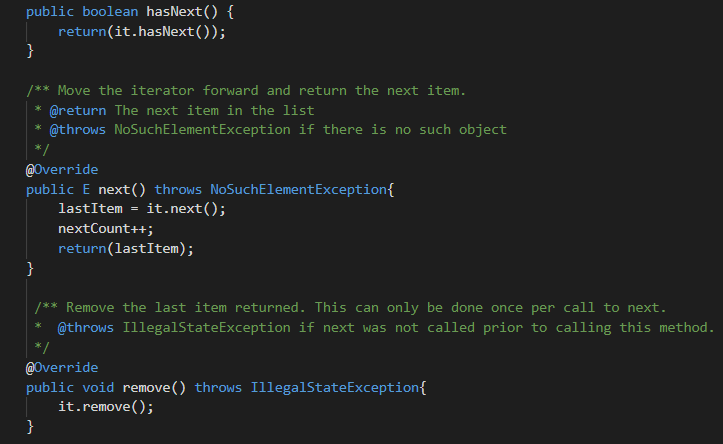
* T(n) = O(nlogn)



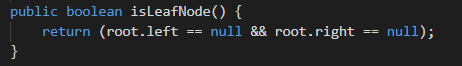
* Since finding peek takes constant time. It is constant time.
* T(n) = θ(1).



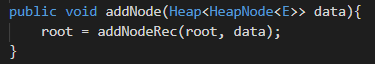
* Offer is O(logn) and there’s loop until nextCount which is to prevent exception of the iterator and it runs n times in the worst case and 1 time in the best case.
* T(n) = O(n)

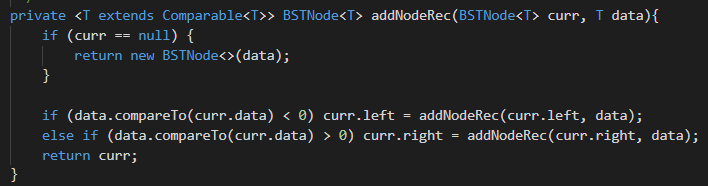


* hasNext, next is θ(1) and remove is O(logn).
* BSTHeapIterator Class

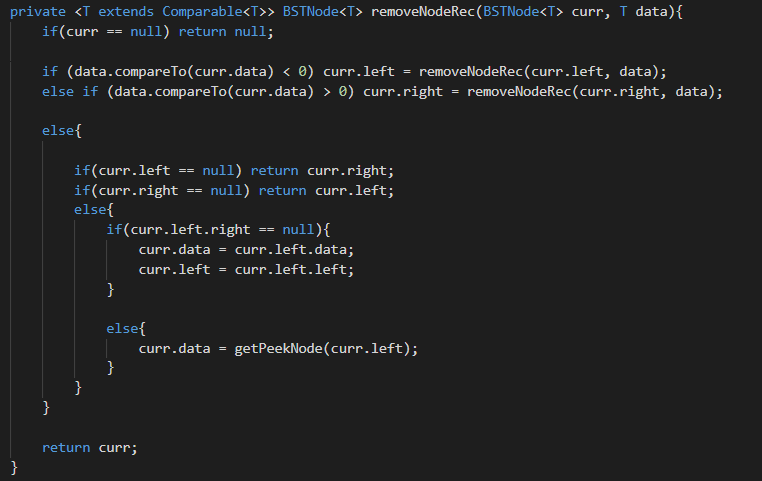


* T(n) = θ(1).



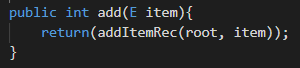
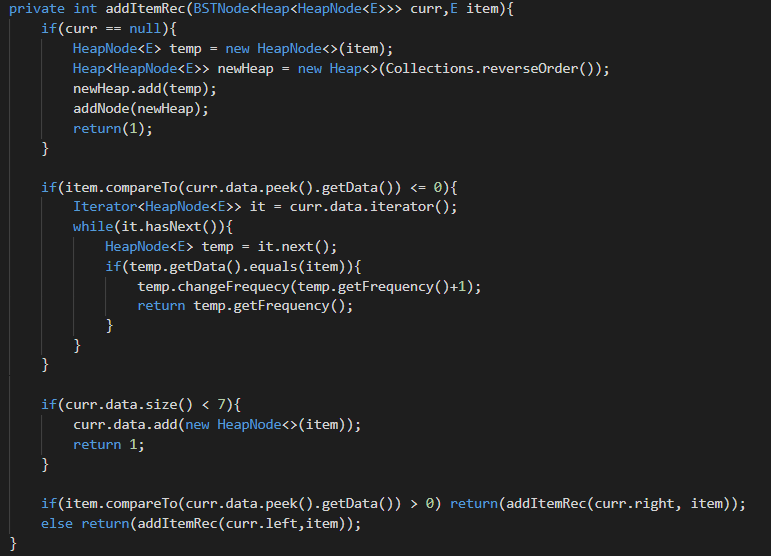


* h is the height of the tree and, T(h) = T(h-1) + θ(1) – rest of it is constant time. And h is logn because it’s a complete tree.
* T(n) = O(logn)

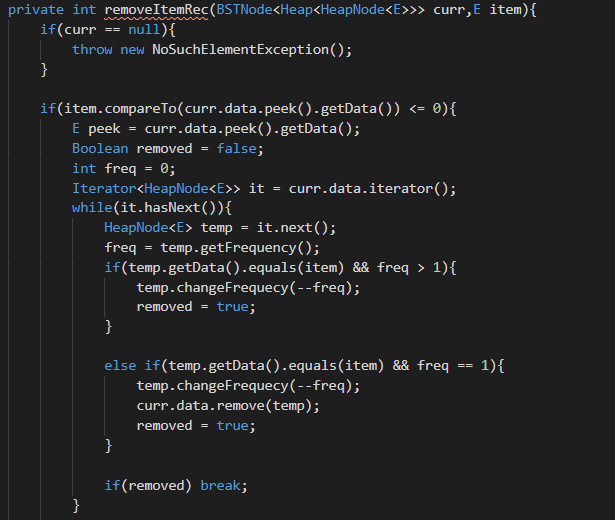
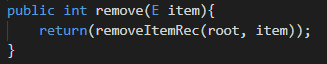
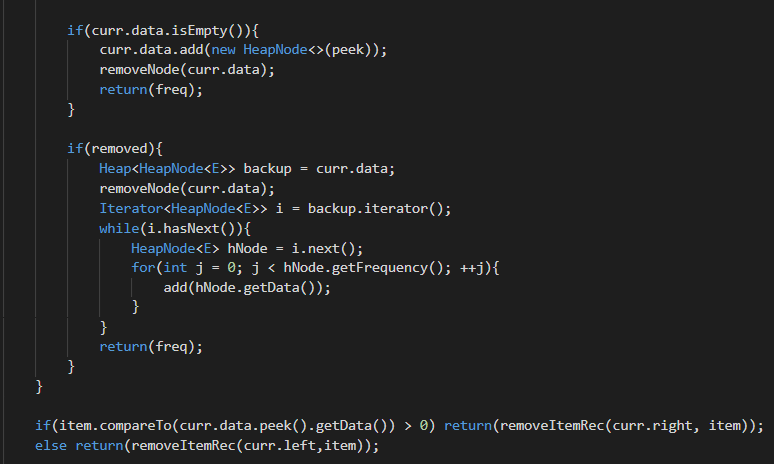




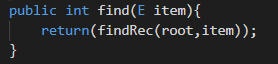
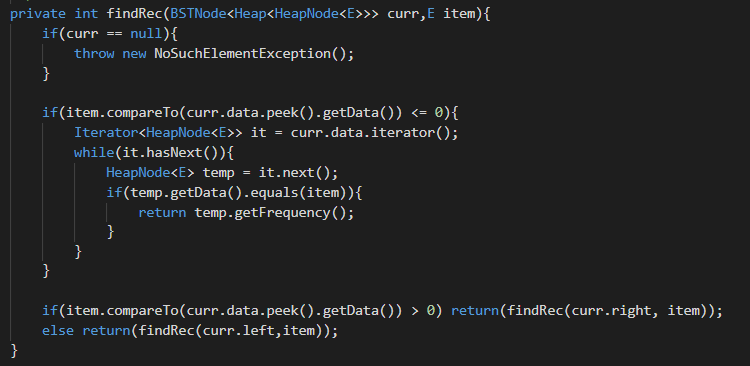
* h is the height of the tree and, T(h) = T(h-1) + θ(1) – rest of it is constant time. And h is logn because it’s a complete tree.
* T(n) = O(logn)



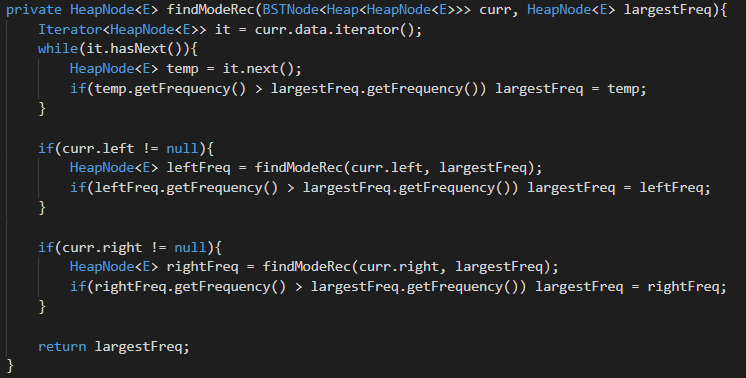
* Adding to heap is 0(logn) and adding node is O(logn) too.
* There’s a loop and it can max loop 7 time we know that so constant.
* T(n) = O(logn)

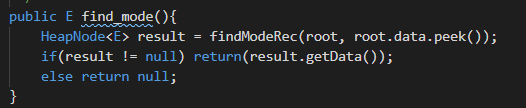
 

* Removing from heap is 0(logn) and removing node is O(logn) too.
* There’s a loop and it can max loop 7 time we know that so constant and there’s another loop in it and it can be m.
* T(n) = O(mlogn)

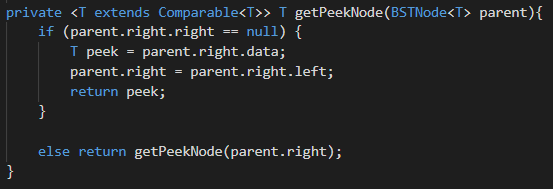


* There’s a loop and it can max loop 7 time we know that so constant. It is traversing recursively that is O(logn).
* T(n) = O(logn)

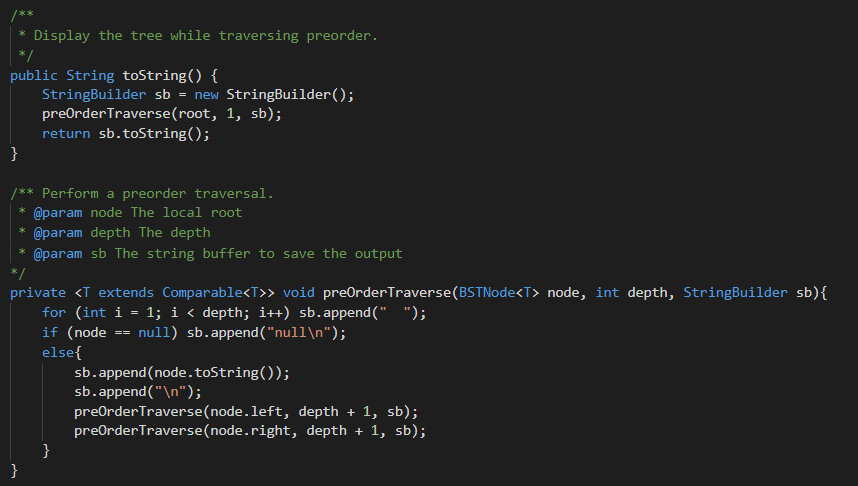




* It is traversing recursively but this time with 2 direction right and left.
* So it is, T(h) = 2.T(h-1) + θ(1) this time and that is O(n).
* T(n) = O(n).



* It is traversing recursively in right direction.
* T(n) = O(logn).



* PreorderTraverse traversing each direction as I mentioned before so O(n).
* To string is O(n) too in this case.