GIT Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework 4 Report

Ahmet Tuğkan Ayhan

1901042692

1. System Requirements

a. Functional Requirements

- For class MaxHeap there are 6 important methods that is available for the user and these are :
  1) int insert(E e)
  2) int searchElement(E e)
  3) void merge(Node<E> secHeap)
  4) E removeWithPriority(int priority)
  5) void print()
  6) setLast() !! accessed by Iterator of the MaxHeap class
- Insert method let's you add an element into the MaxHeap class. By adding you can reach number of occurrence of that element in the tree by getting it's return value.
- searchElement method returns the index (also priority) of that element in the class. If method returns -1, that means the element does not exist in the MaxHeap class.
- merge method is used when merging 2 MaxHeap tree. But the base tree must have available capacity to hold both classes' elements.
- by using removeWithPriority method, you can remove an element from the Heap according to it's priority in that tree. Gives error message when you try to remove from an empty heap or remove non existing element.
- print method prints out the current look of MaxHeap class to the console.

- For class BSTHeapTree there are 5 important methods that is available for the user:
  1) int add(E e)
  2) int remove(E e)
  3) int find(E e)
  4) E find_mode()
  5) int find_mode_occurrence()
- add method let's you add an element into the BSTHeapTree. By adding you can reach number of occurrence of that element in the tree by getting it's return value.
- remove method removes given item from the BSTHeapTree, or(if item's occurrence is more than 1) decreases occurrence value of given item by 1. You can't remove from an empty tree or you can't remove a non existing item(it will return 0).
- find method let's you get the occurrence value of any item in the BSTHeapTree. If item doesn't exist it will return 0.
- find_mode method will return the mode of the BSTHeapTree. and the mode element is the most occurred element in the BSTHeapTree. If there are 2 or more elements with same occurrence value, method will return first element that it finds.
- find_mode_occurrence method returns the occurrence value of mode of the tree. For example, if 12 is the most occurred element in the BSTHeapTree with 6 times, then this method returns 6.

b. Non-Functional Requirements

- MaxHeap :
  insert operations cannot exceed max number of any Heap. Initially if you don't give a number, it can hold max 7 numbers. You can give any max value, as high as you want.

  search operation searches through all elements in the heap and when it finds it returns that numbers index(priority).

  merge cannot happen if base heap that will store both values doesn't have enough room.

  you cannot **remove** from an empty heap.

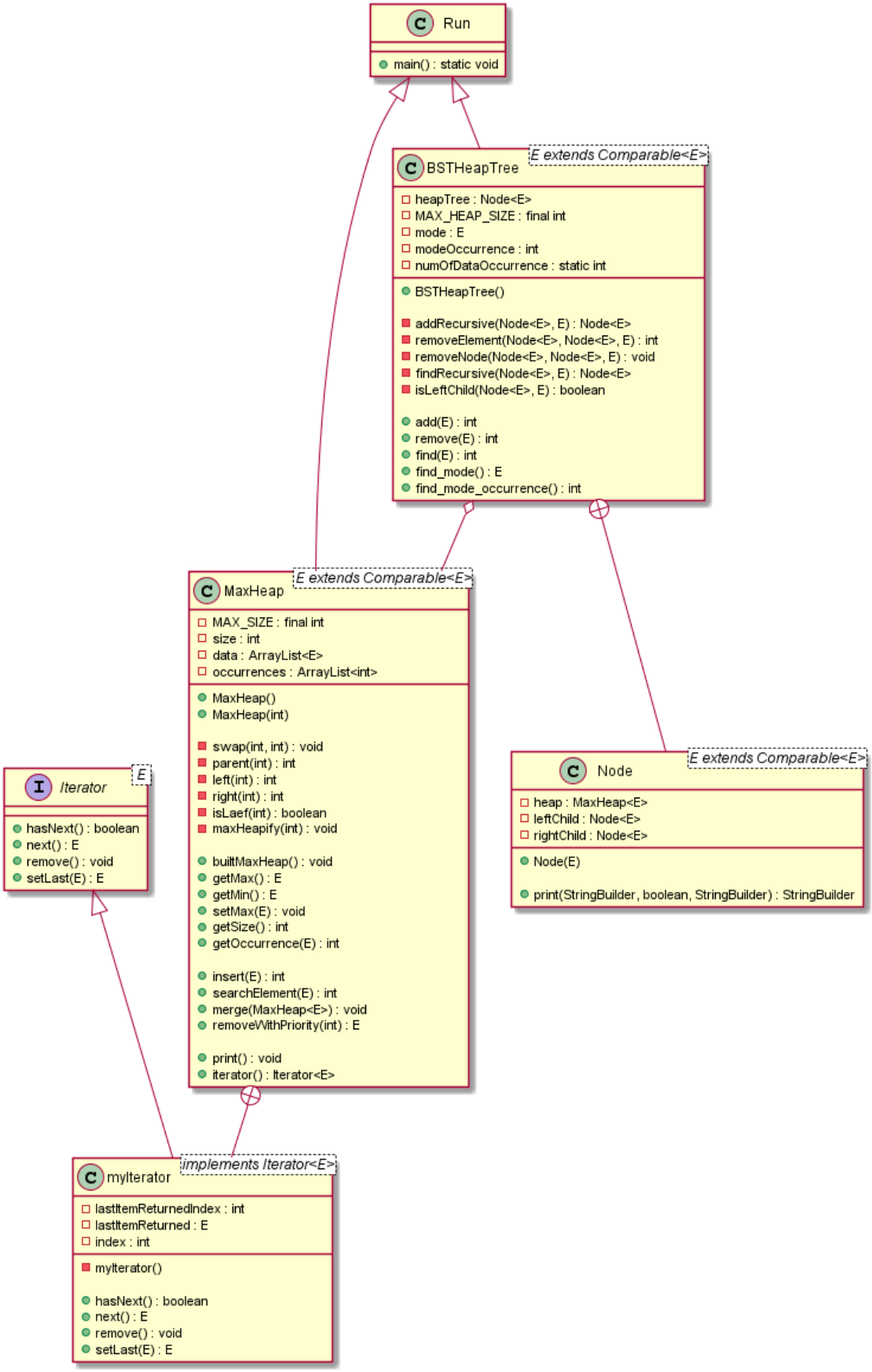
  setLast method will throw nullpointerexception if you try to set directly without using next method. it changes the value for lastItemReturned.

- BSTHeapTree :
  unlike **insert** method of MaxHeap class, **add** method can add infinite amount of item to the BSTHeapTree. When a node(which is a MaxHeap object) is full, it creates another node and adds the element into it.

  remove works by getting the item you want to remove. It won't remove the item if there is more than 1 occurrence. It will decrease it's occurrence value by 1. If there is 1 occurrence of any item and that item is the last item in a node, then it removes both element and that node.

  find method searches through every node in the BSTHeapTree. If any node contains given element then finds how many times it occurred and returns it, and if it does not contain given element then returns 0, which means it never occurred.

2. Class Diagram



3. Problem Solution Approach

Part1

insert => There are 4 possible cases:

- Case 1: The heap I'm trying to insert an item can be null
  Solution : I just allocated new space for heap and added that element to the heap as a first element.

- Case 2: The item can already exist in the heap.
  Solution : I added one occurrence to that item

- Case 3: Heap can be full
  Solution : Give an error message and return

- Case 4: None of these, (heap is not null and item is not in the heap)
  Solution : Just add that item to the heap and swap until it satisfies heap data structure

searchElement => Just searched through ArrayList until I find the item
merge => I used insert until I merge all the item in second heap
remove => I used Arraylist's remove method for last item after I swap that element with least important element (last item)

set => Uses ArrayList's set method to change lastItemReturned element in the heap with given value

Part2

add => I recursively searched through every node in the BTSHeapTree until I find a node with given item to increase it's occurrence. If that kind of Node doesn't exist and current node is full I created a new node, If not full then I added to that node.

remove => Remove method first calls for removeElement. If it can find a node with given element it removes it. But if the element that

method found is no element other than that element than calls removeNode which removes that node from BTSHeapTree. removeNode method looks for least bigger MaxHeap value (first go right than go left until it's null) and swaps it's all element with that node and removes that leaf node.

find => Find method searches through all nodes in the BTSHeapTree. Starts with most left child and goes until most right one. Returns null if element doesn't exist

find_mode => only returns class data field named "mode". Because while I am adding every element I look it's occurrence value and if it bigger than current biggest occurrence value, I assign that element as a mode.

4. Test Cases

**** PART 1 ****

1) Creating 2 Array with sizes 10 and 15 (data type: Integer) myHeap1 (size 10) myHeap2 (size 15)
2) Generating 30 random numbers and holding them in an array
3) Inserting 11 random number into myHeap1 and showing the error message. (Also printing the myHeap1 to the console)
4) Inserting 16 random numbers into myHeap2 and showing the error message. (showing current sizes and trees)
5) Trying to merge these two arrays. Showing error message why it can't happen
6) Creating another array(myHeap3) with size of 25 and merging these 2 array with that node. Printing out current tree of myHeap3
7) Searching for element "1234". If this element exist (low chance :c) showing it returns it's index, otherwise -1
8) Removing least important and most important nodes from myHeap3 and printing out current tree
9) Getting iterator for myHeap3 and going until last element and setting it's value as "1".

10) Removing every element from myHeap3. And trying to remove while there is no element inside

**** PART 2 ****

1) Allocating a new BSTHeapTree named “myBST” and adding 3000 random numbers into that tree. (bound:5000)
2) Printing current tree
3) Searching through 100 elements that is in the random array and returning these number’s occurrence value
4) Searching through 10 elements that is NOT in the random array and returning these number’s occurrence value
5) Printing out mode of the BSTHeapTree and it’s number of occurrence.
6) Getting occurrence value of 100 random elements that is in the array before removing them and after removing them. And after that printing these values
7) Getting occurrence value of 10 elements that is NOT in the array before and after removing them. Printing these values.
8) Finally printing current state of BSTHeapTree

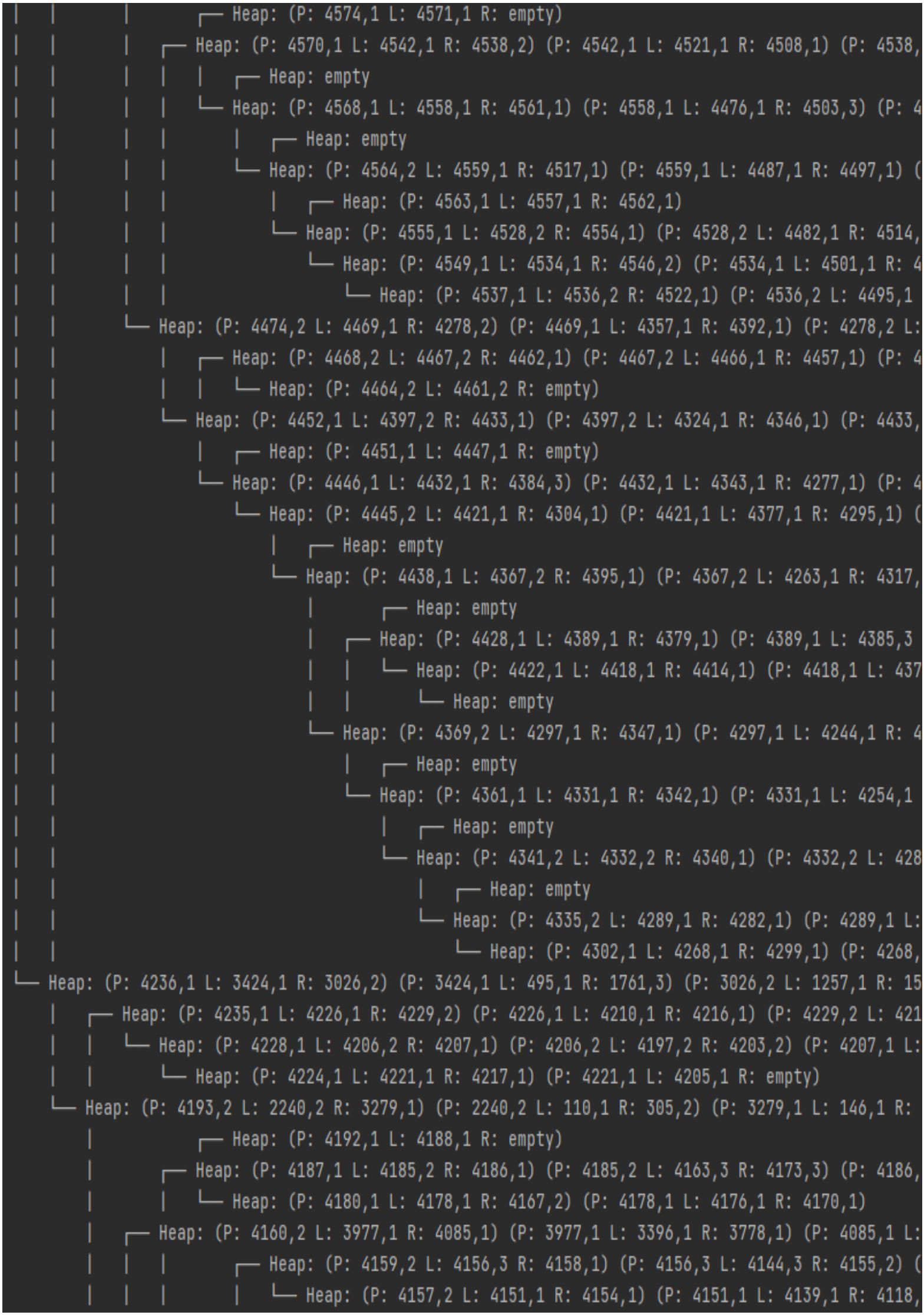
5. Running and Results

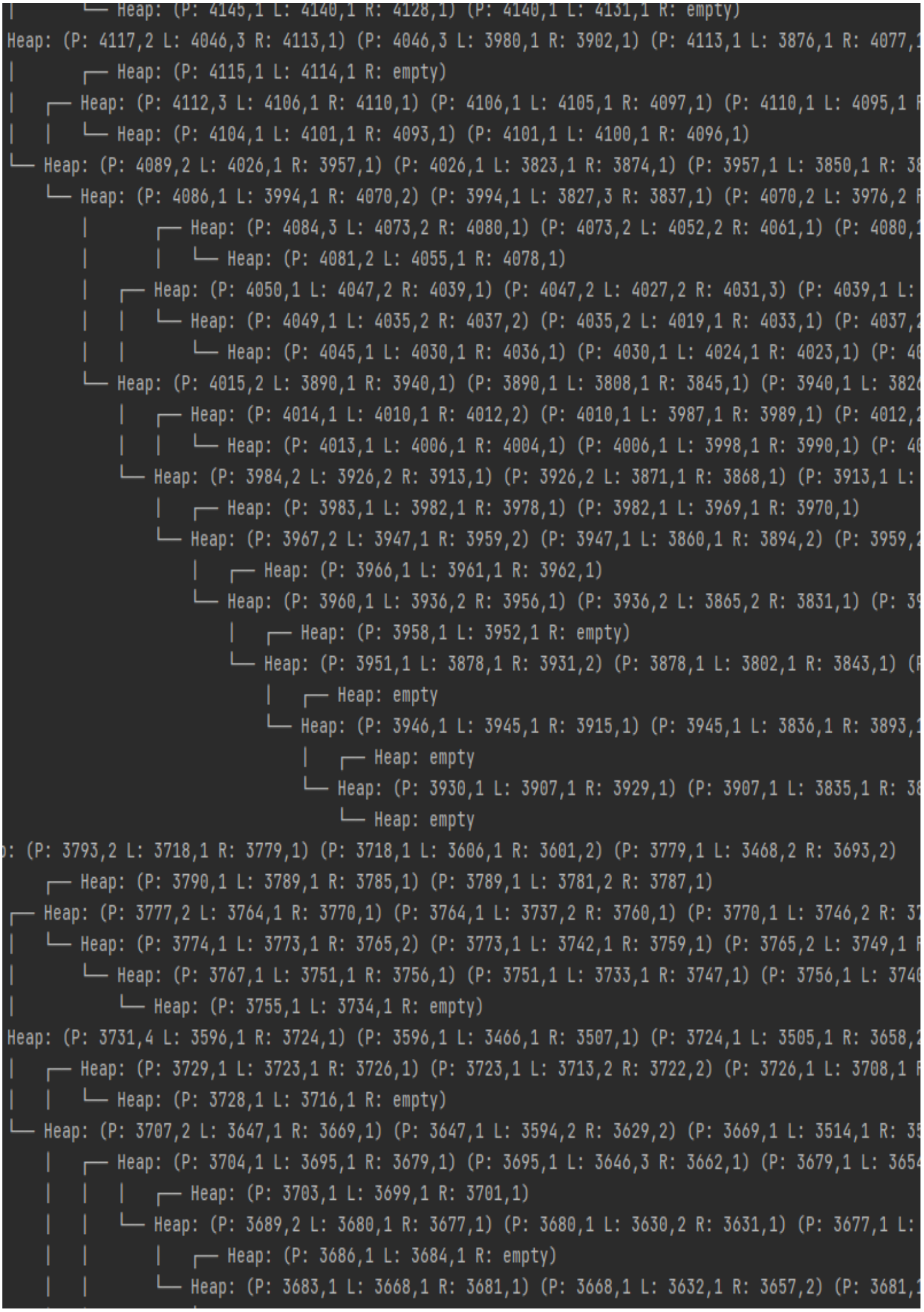
```
------------ [PART 1] ------------

You are trying to add more than you could. (Max: 10) (Value which not added: 4217)
** MyHeap1 **
1)  Parent: 8582        Left Child: 8386     Right Child: 4889
2)  Parent: 8386        Left Child: 6803     Right Child: 7487
3)  Parent: 4889        Left Child: 1388     Right Child: 1311
4)  Parent: 6803        Left Child: 2769     Right Child: 3339
5)  Parent: 7487        Left Child: 5940     Right Child: empty

Current Size(MyHeap1): 10
Current Size(MyHeap2): 0

You are trying to add more than you could. (Max: 15) (Value which not added: 9599)
** MyHeap2 **
1)  Parent: 9889        Left Child: 7446     Right Child: 9853
2)  Parent: 7446        Left Child: 7203     Right Child: 7034
3)  Parent: 9853        Left Child: 7673     Right Child: 9365
4)  Parent: 7203        Left Child: 393      Right Child: 4118
5)  Parent: 7034        Left Child: 1387     Right Child: 6090
6)  Parent: 7673        Left Child: 4217     Right Child: 1159
7)  Parent: 9365        Left Child: 2676     Right Child: 6822

Current Size(MyHeap1): 10
Current Size(MyHeap2): 15

** Merging (myHeap2 + myHeap1 --> myHeap2) **
You are trying to add more than you could. (Max: 15) (Value which not added: 8582)
You are trying to add more than you could. (Max: 15) (Value which not added: 8386)
You are trying to add more than you could. (Max: 15) (Value which not added: 4889)
You are trying to add more than you could. (Max: 15) (Value which not added: 6803)
You are trying to add more than you could. (Max: 15) (Value which not added: 7487)
You are trying to add more than you could. (Max: 15) (Value which not added: 1388)
You are trying to add more than you could. (Max: 15) (Value which not added: 1311)
You are trying to add more than you could. (Max: 15) (Value which not added: 2769)
You are trying to add more than you could. (Max: 15) (Value which not added: 3339)
You are trying to add more than you could. (Max: 15) (Value which not added: 5940)

Creating another object named myHeap3

Current size(myHeap3): 0
```

```
** Merging (myHeap3 + myHeap1 --> myHeap3)

** MyHeap3 **
1)  Parent: 8582        Left Child: 8386     Right Child: 4889
2)  Parent: 8386        Left Child: 6803     Right Child: 7487
3)  Parent: 4889        Left Child: 1388     Right Child: 1311
4)  Parent: 6803        Left Child: 2769     Right Child: 3339
5)  Parent: 7487        Left Child: 5940     Right Child: empty

Current size(myHeap3): 10

** Merging (myHeap3 + myHeap2 --> myHeap3)

** MyHeap3 **
1)  Parent: 9889        Left Child: 9365     Right Child: 9853
2)  Parent: 9365        Left Child: 8582     Right Child: 8386
3)  Parent: 9853        Left Child: 7446     Right Child: 7203
4)  Parent: 8582        Left Child: 7673     Right Child: 4118
5)  Parent: 8386        Left Child: 6090     Right Child: 7487
6)  Parent: 7446        Left Child: 6822     Right Child: 4889
7)  Parent: 7203        Left Child: 1311     Right Child: 7034
8)  Parent: 7673        Left Child: 2769     Right Child: 6803
9)  Parent: 4118        Left Child: 393      Right Child: 3339
10) Parent: 6090        Left Child: 1387     Right Child: 5940
11) Parent: 7487        Left Child: 4217     Right Child: 1159
12) Parent: 6822        Left Child: 1388     Right Child: 2676

Current size(myHeap3): 25

Index of element '1234' (-1 if it doesn't exist in the heap) : -1

Removing least important node(also a leaf node) : 1388

** MyHeap3 ** (after removing least important)
1)  Parent: 9889        Left Child: 9365     Right Child: 9853
2)  Parent: 9365        Left Child: 8582     Right Child: 8386
3)  Parent: 9853        Left Child: 7446     Right Child: 7203
4)  Parent: 8582        Left Child: 7673     Right Child: 4118
5)  Parent: 8386        Left Child: 6090     Right Child: 7487
6)  Parent: 7446        Left Child: 6822     Right Child: 4889
7)  Parent: 7203        Left Child: 1311     Right Child: 7034
8)  Parent: 7673        Left Child: 2769     Right Child: 6803
```

```
9)  Parent: 4118        Left Child: 393     Right Child: 3339
10) Parent: 6090        Left Child: 1387    Right Child: 5940
11) Parent: 7487        Left Child: 4217    Right Child: 1159
12) Parent: 6822        Left Child: 2676    Right Child: empty


Removing root node(also a two children node) : 9889

** MyHeap3 ** (after removing root)
1)  Parent: 2676        Left Child: 9365    Right Child: 9853
2)  Parent: 9365        Left Child: 8582    Right Child: 8386
3)  Parent: 9853        Left Child: 7446    Right Child: 7203
4)  Parent: 8582        Left Child: 7673    Right Child: 4118
5)  Parent: 8386        Left Child: 6090    Right Child: 7487
6)  Parent: 7446        Left Child: 6822    Right Child: 4889
7)  Parent: 7203        Left Child: 1311    Right Child: 7034
8)  Parent: 7673        Left Child: 2769    Right Child: 6803
9)  Parent: 4118        Left Child: 393     Right Child: 3339
10) Parent: 6090        Left Child: 1387    Right Child: 5940
11) Parent: 7487        Left Child: 4217    Right Child: 1159


** MyHeap3 ** (after iterating an iterator for 5 elements. It calls maxHeapify method inside so it
1)  Parent: 9853        Left Child: 9365    Right Child: 7446
2)  Parent: 9365        Left Child: 8582    Right Child: 8386
3)  Parent: 7446        Left Child: 6822    Right Child: 7203
4)  Parent: 8582        Left Child: 7673    Right Child: 4118
5)  Parent: 8386        Left Child: 6090    Right Child: 7487
6)  Parent: 6822        Left Child: 2676    Right Child: 4889
7)  Parent: 7203        Left Child: 1311    Right Child: 7034
8)  Parent: 7673        Left Child: 2769    Right Child: 6803
9)  Parent: 4118        Left Child: 393     Right Child: 3339
10) Parent: 6090        Left Child: 1387    Right Child: 5940
11) Parent: 7487        Left Child: 4217    Right Child: 1

Printing class after removing every element inside
(Empty heap)

Trying to remove while there is no element inside

------------ [PART 2] ------------
```

```
----------- [PART 2] -----------

|       ┌─ Heap: (P: 4997,3 L: 4977,1 R: 4993,2) (P: 4977,1 L: 4953,2 R: 4967,1) (P: 4993,2 L: 498
|       |  └─ Heap: (P: 4983,1 L: 4982,1 R: 4954,2) (P: 4982,1 L: 4951,1 R: 4958,1) (P: 4954,2 L:
|       |     └─ Heap: (P: 4981,1 L: 4976,1 R: 4979,1) (P: 4976,1 L: 4940,1 R: 4969,1) (P: 4979,
|   ┌─ Heap: (P: 4939,1 L: 4911,1 R: 4810,1) (P: 4911,1 L: 4826,2 R: 4833,1) (P: 4810,1 L: 4739,3
|   |   |   ┌─ Heap: (P: 4937,1 L: 4934,2 R: 4931,1) (P: 4934,2 L: 4928,2 R: 4926,2) (P: 4931,1 L:
|   |   |   |  └─ Heap: empty
|   |   └─ Heap: (P: 4925,1 L: 4796,2 R: 4906,2) (P: 4796,2 L: 4790,1 R: 4768,2) (P: 4906,2 L: 481
|   |       |   ┌─ Heap: empty
|   |       └─ Heap: (P: 4922,1 L: 4890,1 R: 4899,1) (P: 4890,1 L: 4889,1 R: 4886,3) (P: 4899,1 L:
|   |           |      ┌─ Heap: empty
|   |           |   ┌─ Heap: (P: 4920,1 L: 4882,1 R: 4908,1) (P: 4882,1 L: 4880,3 R: 4873,1) (P: 4
|   |           |   |  └─ Heap: (P: 4913,1 L: 4879,1 R: 4902,1) (P: 4879,1 L: 4867,1 R: 4878,2) (
|   |           |   |     └─ Heap: (P: 4912,1 L: 4901,1 R: 4907,1) (P: 4901,1 L: 4870,1 R: 4868,
|   |           |   |        └─ Heap: empty
|   |           └─ Heap: (P: 4862,2 L: 4840,1 R: 4835,3) (P: 4840,1 L: 4766,2 R: 4778,4) (P: 4835,
|   |              |   ┌─ Heap: (P: 4858,1 L: 4856,1 R: empty)
|   |              └─ Heap: (P: 4853,1 L: 4807,3 R: 4837,1) (P: 4807,3 L: 4774,1 R: 4776,1) (P: 48
|   |                 |   ┌─ Heap: (P: 4848,1 L: 4845,1 R: 4847,2) (P: 4845,1 L: 4824,1 R: 4832,2
|   |                 |   |  └─ Heap: (P: 4838,1 L: 4836,1 R: 4809,2) (P: 4836,1 L: 4827,1 R: en
|   |                 └─ Heap: (P: 4803,1 L: 4783,1 R: 4798,3) (P: 4783,1 L: 4721,3 R: 4764,1) (
|   |                    └─ Heap: (P: 4795,2 L: 4792,1 R: 4782,1) (P: 4792,1 L: 4770,1 R: 4720,1
|   |                       └─ Heap: (P: 4793,2 L: 4761,1 R: 4765,2) (P: 4761,1 L: 4731,1 R: 47
|   |                          └─ Heap: (P: 4788,1 L: 4775,1 R: 4785,1) (P: 4775,1 L: 4751,1
└─ Heap: (P: 4713,2 L: 1437,2 R: 2525,1) (P: 1437,2 L: 545,3 R: 1087,2) (P: 2525,1 L: 1332,2 R: 471
    |          ┌─ Heap: empty
    |       ┌─ Heap: (P: 4710,2 L: 4705,1 R: 4709,2) (P: 4705,1 L: 4697,3 R: 4700,2) (P: 4709,2 L:
    |       |  └─ Heap: (P: 4707,2 L: 4699,1 R: empty)
    |   ┌─ Heap: (P: 4695,1 L: 4401,1 R: 4669,1) (P: 4401,1 L: 4358,3 R: 4399,1) (P: 4669,1 L: 4310
    |   |   |      ┌─ Heap: (P: 4694,2 L: 4684,1 R: 4686,1) (P: 4684,1 L: 4682,1 R: 4675,1)
    |   |   |   ┌─ Heap: (P: 4674,1 L: 4602,1 R: 4647,2) (P: 4602,1 L: 4599,1 R: 4587,1) (P: 4647,2
    |   |   |   |  |   ┌─ Heap: empty
    |   |   |   └─ Heap: (P: 4672,2 L: 4667,1 R: 4653,2) (P: 4667,1 L: 4597,1 R: 4589,2) (P: 46
    |   |   |      |   ┌─ Heap: (P: 4670,1 L: 4663,1 R: 4662,1) (P: 4663,1 L: 4659,1 R: empty)
    |   |   |      └─ Heap: (P: 4654,1 L: 4621,2 R: 4651,1) (P: 4621,2 L: 4578,1 R: 4580,1) (P
    |   |   |         |   ┌─ Heap: (P: 4650,1 L: 4649,1 R: 4637,2) (P: 4649,1 L: 4644,3 R: 46
    |   |   |         └─ Heap: (P: 4636,2 L: 4616,2 R: 4623,1) (P: 4616,2 L: 4595,1 R: 4600,1
    |   |   |            └─ Heap: (P: 4634,1 L: 4627,1 R: 4632,1) (P: 4627,1 L: 4591,1 R: 46
    |   |   |               └─ Heap: (P: 4604,1 L: 4592,1 R: empty)
    |   |   └─ Heap: (P: 4577,1 L: 4550,2 R: 4539,1) (P: 4550,2 L: 4338,1 R: 4364,2) (P: 4539,1 L:
```


Heap: (P: 4574,1 L: 4571,1 R: empty)
Heap: (P: 4570,1 L: 4542,1 R: 4538,2) (P: 4542,1 L: 4521,1 R: 4508,1) (P: 4538,
Heap: empty
Heap: (P: 4568,1 L: 4558,1 R: 4561,1) (P: 4558,1 L: 4476,1 R: 4503,3) (P: 4
Heap: empty
Heap: (P: 4564,2 L: 4559,1 R: 4517,1) (P: 4559,1 L: 4487,1 R: 4497,1) (
Heap: (P: 4563,1 L: 4557,1 R: 4562,1)
Heap: (P: 4555,1 L: 4528,2 R: 4554,1) (P: 4528,2 L: 4482,1 R: 4514,
Heap: (P: 4549,1 L: 4534,1 R: 4546,2) (P: 4534,1 L: 4501,1 R: 4
Heap: (P: 4537,1 L: 4536,2 R: 4522,1) (P: 4536,2 L: 4495,1
Heap: (P: 4474,2 L: 4469,1 R: 4278,2) (P: 4469,1 L: 4357,1 R: 4392,1) (P: 4278,2 L:
Heap: (P: 4468,2 L: 4467,2 R: 4462,1) (P: 4467,2 L: 4466,1 R: 4457,1) (P: 4
Heap: (P: 4464,2 L: 4461,2 R: empty)
Heap: (P: 4452,1 L: 4397,2 R: 4433,1) (P: 4397,2 L: 4324,1 R: 4346,1) (P: 4433,
Heap: (P: 4451,1 L: 4447,1 R: empty)
Heap: (P: 4446,1 L: 4432,1 R: 4384,3) (P: 4432,1 L: 4343,1 R: 4277,1) (P: 4
Heap: (P: 4445,2 L: 4421,1 R: 4304,1) (P: 4421,1 L: 4377,1 R: 4295,1) (
Heap: empty
Heap: (P: 4438,1 L: 4367,2 R: 4395,1) (P: 4367,2 L: 4263,1 R: 4317,
Heap: empty
Heap: (P: 4428,1 L: 4389,1 R: 4379,1) (P: 4389,1 L: 4385,3
Heap: (P: 4422,1 L: 4418,1 R: 4414,1) (P: 4418,1 L: 437
Heap: empty
Heap: (P: 4369,2 L: 4297,1 R: 4347,1) (P: 4297,1 L: 4244,1 R: 4
Heap: empty
Heap: (P: 4361,1 L: 4331,1 R: 4342,1) (P: 4331,1 L: 4254,1
Heap: empty
Heap: (P: 4341,2 L: 4332,2 R: 4340,1) (P: 4332,2 L: 428
Heap: empty
Heap: (P: 4335,2 L: 4289,1 R: 4282,1) (P: 4289,1 L:
Heap: (P: 4302,1 L: 4268,1 R: 4299,1) (P: 4268,
Heap: (P: 4236,1 L: 3424,1 R: 3026,2) (P: 3424,1 L: 495,1 R: 1761,3) (P: 3026,2 L: 1257,1 R: 15
Heap: (P: 4235,1 L: 4226,1 R: 4229,2) (P: 4226,1 L: 4210,1 R: 4216,1) (P: 4229,2 L: 421
Heap: (P: 4228,1 L: 4206,2 R: 4207,1) (P: 4206,2 L: 4197,2 R: 4203,2) (P: 4207,1 L:
Heap: (P: 4224,1 L: 4221,1 R: 4217,1) (P: 4221,1 L: 4205,1 R: empty)
Heap: (P: 4193,2 L: 2240,2 R: 3279,1) (P: 2240,2 L: 110,1 R: 305,2) (P: 3279,1 L: 146,1 R:
Heap: (P: 4192,1 L: 4188,1 R: empty)
Heap: (P: 4187,1 L: 4185,2 R: 4186,1) (P: 4185,2 L: 4163,3 R: 4173,3) (P: 4186,
Heap: (P: 4180,1 L: 4178,1 R: 4167,2) (P: 4178,1 L: 4176,1 R: 4170,1)
Heap: (P: 4160,2 L: 3977,1 R: 4085,1) (P: 3977,1 L: 3396,1 R: 3778,1) (P: 4085,1 L:
Heap: (P: 4159,2 L: 4156,3 R: 4158,1) (P: 4156,3 L: 4144,3 R: 4155,2) (
Heap: (P: 4157,2 L: 4151,1 R: 4154,1) (P: 4151,1 L: 4139,1 R: 4118,


└─ Heap: (P: 4145,1 L: 4140,1 R: 4128,1) (P: 4140,1 L: 4131,1 R: empty)
Heap: (P: 4117,2 L: 4046,3 R: 4113,1) (P: 4046,3 L: 3980,1 R: 3902,1) (P: 4113,1 L: 3876,1 R: 4077,1
┌─ Heap: (P: 4115,1 L: 4114,1 R: empty)
┌─ Heap: (P: 4112,3 L: 4106,1 R: 4110,1) (P: 4106,1 L: 4105,1 R: 4097,1) (P: 4110,1 L: 4095,1 R
└─ Heap: (P: 4104,1 L: 4101,1 R: 4093,1) (P: 4101,1 L: 4100,1 R: 4096,1)
└─ Heap: (P: 4089,2 L: 4026,1 R: 3957,1) (P: 4026,1 L: 3823,1 R: 3874,1) (P: 3957,1 L: 3850,1 R: 38
└─ Heap: (P: 4086,1 L: 3994,1 R: 4070,2) (P: 3994,1 L: 3827,3 R: 3837,1) (P: 4070,2 L: 3976,2 R
┌─ Heap: (P: 4084,3 L: 4073,2 R: 4080,1) (P: 4073,2 L: 4052,2 R: 4061,1) (P: 4080,1
└─ Heap: (P: 4081,2 L: 4055,1 R: 4078,1)
┌─ Heap: (P: 4050,1 L: 4047,2 R: 4039,1) (P: 4047,2 L: 4027,2 R: 4031,3) (P: 4039,1 L:
└─ Heap: (P: 4049,1 L: 4035,2 R: 4037,2) (P: 4035,2 L: 4019,1 R: 4033,1) (P: 4037,2
└─ Heap: (P: 4045,1 L: 4030,1 R: 4036,1) (P: 4030,1 L: 4024,1 R: 4023,1) (P: 40
└─ Heap: (P: 4015,2 L: 3890,1 R: 3940,1) (P: 3890,1 L: 3808,1 R: 3845,1) (P: 3940,1 L: 3826
┌─ Heap: (P: 4014,1 L: 4010,1 R: 4012,2) (P: 4010,1 L: 3987,1 R: 3989,1) (P: 4012,2
└─ Heap: (P: 4013,1 L: 4006,1 R: 4004,1) (P: 4006,1 L: 3998,1 R: 3990,1) (P: 40
└─ Heap: (P: 3984,2 L: 3926,2 R: 3913,1) (P: 3926,2 L: 3871,1 R: 3868,1) (P: 3913,1 L:
┌─ Heap: (P: 3983,1 L: 3982,1 R: 3978,1) (P: 3982,1 L: 3969,1 R: 3970,1)
└─ Heap: (P: 3967,2 L: 3947,1 R: 3959,2) (P: 3947,1 L: 3860,1 R: 3894,2) (P: 3959,2
┌─ Heap: (P: 3966,1 L: 3961,1 R: 3962,1)
└─ Heap: (P: 3960,1 L: 3936,2 R: 3956,1) (P: 3936,2 L: 3865,2 R: 3831,1) (P: 39
┌─ Heap: (P: 3958,1 L: 3952,1 R: empty)
└─ Heap: (P: 3951,1 L: 3878,1 R: 3931,2) (P: 3878,1 L: 3802,1 R: 3843,1) (P
┌─ Heap: empty
└─ Heap: (P: 3946,1 L: 3945,1 R: 3915,1) (P: 3945,1 L: 3836,1 R: 3893,1
┌─ Heap: empty
└─ Heap: (P: 3930,1 L: 3907,1 R: 3929,1) (P: 3907,1 L: 3835,1 R: 38
└─ Heap: empty
): (P: 3793,2 L: 3718,1 R: 3779,1) (P: 3718,1 L: 3606,1 R: 3601,2) (P: 3779,1 L: 3468,2 R: 3693,2)
┌─ Heap: (P: 3790,1 L: 3789,1 R: 3785,1) (P: 3789,1 L: 3781,2 R: 3787,1)
┌─ Heap: (P: 3777,2 L: 3764,1 R: 3770,1) (P: 3764,1 L: 3737,2 R: 3760,1) (P: 3770,1 L: 3746,2 R: 37
└─ Heap: (P: 3774,1 L: 3773,1 R: 3765,2) (P: 3773,1 L: 3742,1 R: 3759,1) (P: 3765,2 L: 3749,1 R
└─ Heap: (P: 3767,1 L: 3751,1 R: 3756,1) (P: 3751,1 L: 3733,1 R: 3747,1) (P: 3756,1 L: 3740
└─ Heap: (P: 3755,1 L: 3734,1 R: empty)
Heap: (P: 3731,4 L: 3596,1 R: 3724,1) (P: 3596,1 L: 3466,1 R: 3507,1) (P: 3724,1 L: 3505,1 R: 3658,2
┌─ Heap: (P: 3729,1 L: 3723,1 R: 3726,1) (P: 3723,1 L: 3713,2 R: 3722,2) (P: 3726,1 L: 3708,1 R
└─ Heap: (P: 3728,1 L: 3716,1 R: empty)
└─ Heap: (P: 3707,2 L: 3647,1 R: 3669,1) (P: 3647,1 L: 3594,2 R: 3629,2) (P: 3669,1 L: 3514,1 R: 35
┌─ Heap: (P: 3704,1 L: 3695,1 R: 3679,1) (P: 3695,1 L: 3646,3 R: 3662,1) (P: 3679,1 L: 3654
┌─ Heap: (P: 3703,1 L: 3699,1 R: 3701,1)
└─ Heap: (P: 3689,2 L: 3680,1 R: 3677,1) (P: 3680,1 L: 3630,2 R: 3631,1) (P: 3677,1 L:
┌─ Heap: (P: 3686,1 L: 3684,1 R: empty)
└─ Heap: (P: 3683,1 L: 3668,1 R: 3681,1) (P: 3668,1 L: 3632,1 R: 3657,2) (P: 3681,1

```
  |   |           └─ Heap: (P: 3667,1 L: 3644,1 R: 3635,1)
  └─ Heap: (P: 3627,2 L: 3530,2 R: 3617,2) (P: 3530,2 L: 3429,2 R: 3508,1) (P: 3617,2 L: 3394,2 R:
      |   ┌─ Heap: (P: 3626,1 L: 3623,1 R: 3621,1) (P: 3623,1 L: 3622,1 R: empty)
      └─ Heap: (P: 3613,1 L: 3602,4 R: 3532,1) (P: 3602,4 L: 3440,2 R: 3526,2) (P: 3532,1 L: 3441,1
          |   ┌─ Heap: (P: 3612,1 L: 3610,2 R: 3604,1) (P: 3610,2 L: 3609,1 R: 3603,1)
          └─ Heap: (P: 3593,1 L: 3551,2 R: 3568,2) (P: 3551,2 L: 3403,1 R: 3491,1) (P: 3568,2 L: 34
              |       ┌─ Heap: empty
              |   ┌─ Heap: (P: 3580,1 L: 3571,1 R: 3567,1) (P: 3571,1 L: 3563,2 R: 3570,2) (P: 3567
              |   |   └─ Heap: (P: 3574,1 L: 3556,1 R: 3560,1)
              └─ Heap: (P: 3555,3 L: 3538,1 R: 3553,1) (P: 3538,1 L: 3463,2 R: 3432,1) (P: 3553,1 L
                  |   ┌─ Heap: (P: 3550,2 L: 3544,2 R: 3549,1) (P: 3544,2 L: 3537,1 R: 3541,1) (P:
                  └─ Heap: (P: 3534,1 L: 3462,1 R: 3519,1) (P: 3462,1 L: 3446,1 R: 3448,1) (P: 3519
                      └─ Heap: (P: 3527,2 L: 3524,1 R: 3513,1) (P: 3524,1 L: 3456,1 R: 3395,1) (P:
                          └─ Heap: (P: 3517,1 L: 3480,1 R: 3490,1) (P: 3480,1 L: 3421,1 R: 3474,1)
                              └─ Heap: (P: 3512,1 L: 3500,1 R: 3475,1) (P: 3500,1 L: 3402,1 R: 3415
                                  └─ Heap: (P: 3458,1 L: 3457,1 R: 3413,1) (P: 3457,1 L: 3410,2 R:
,1 L: 3089,2 R: 2910,3) (P: 3089,2 L: 283,1 R: 982,1) (P: 2910,3 L: 178,2 R: 996,1)
─ Heap: (P: 3389,1 L: 3388,2 R: 3387,1) (P: 3388,2 L: 3373,1 R: 3378,2) (P: 3387,1 L: 3376,1 R: 3366
  └─ Heap: (P: 3382,1 L: 3381,1 R: 3367,1) (P: 3381,1 L: 3380,1 R: empty)
ap: (P: 3365,2 L: 3357,2 R: 3350,2) (P: 3357,2 L: 3300,1 R: 3321,1) (P: 3350,2 L: 3349,1 R: 3339,2)
      ┌─ Heap: empty
  ┌─ Heap: (P: 3360,1 L: 3353,1 R: 3356,1) (P: 3353,1 L: 3340,1 R: 3345,1) (P: 3356,1 L: 3343,2 R:
  |   └─ Heap: empty
─ Heap: (P: 3336,1 L: 3312,1 R: 3316,1) (P: 3312,1 L: 3292,1 R: 3295,1) (P: 3316,1 L: 3299,1 R: 3305
  └─ Heap: (P: 3328,1 L: 3313,1 R: 3314,1) (P: 3313,1 L: 3297,1 R: 3306,1) (P: 3314,1 L: 3301,2 R:
      └─ Heap: (P: 3326,1 L: 3319,1 R: 3322,1) (P: 3319,1 L: 3291,1 R: 3304,1) (P: 3322,1 L: 3302,1
          └─ Heap: (P: 3308,1 L: 3307,1 R: empty)
(P: 3290,1 L: 3117,2 R: 3195,2) (P: 3117,2 L: 2965,2 R: 2983,3) (P: 3195,2 L: 3044,2 R: 2878,1)
  ┌─ Heap: (P: 3287,1 L: 3286,1 R: empty)
─ Heap: (P: 3282,1 L: 3272,2 R: 3249,1) (P: 3272,2 L: 3217,1 R: 3245,1) (P: 3249,1 L: 3239,2 R: 3213
  |   ┌─ Heap: empty
  └─ Heap: (P: 3278,1 L: 3274,1 R: 3218,1) (P: 3274,1 L: 3196,2 R: 3221,1) (P: 3218,1 L: 3194,1 R:
      |   ┌─ Heap: (P: 3276,1 L: 3267,1 R: 3271,1) (P: 3267,1 L: 3260,1 R: 3263,1) (P: 3271,1 L: 32
      └─ Heap: (P: 3258,2 L: 3254,1 R: 3256,1) (P: 3254,1 L: 3228,2 R: 3241,1) (P: 3256,1 L: 3201,1
          └─ Heap: (P: 3250,2 L: 3236,2 R: 3212,1) (P: 3236,2 L: 3222,1 R: 3200,1) (P: 3212,1 L: 32
              └─ Heap: (P: 3231,1 L: 3219,1 R: empty)
ap: (P: 3192,1 L: 3101,1 R: 3094,1) (P: 3101,1 L: 2918,2 R: 2811,2) (P: 3094,1 L: 3081,1 R: 2841,3)
      ┌─ Heap: (P: 3187,1 L: 3185,2 R: empty)
  ┌─ Heap: (P: 3183,1 L: 3182,1 R: 3173,2) (P: 3182,1 L: 3150,3 R: 3147,1) (P: 3173,2 L: 3164,1 R:
  |   └─ Heap: (P: 3162,1 L: 3157,2 R: 3151,1) (P: 3157,2 L: 3145,1 R: 3153,1) (P: 3151,1 L: 3146,1
─ Heap: (P: 3144,1 L: 3139,2 R: 2986,3) (P: 3139,2 L: 2900,3 R: 3001,4) (P: 2986,3 L: 2809,3 R: 2870
  |   ┌─ Heap: (P: 3141,1 L: 3136,2 R: 3140,2)
```

```
Printing occurrences of 100 values that is in the array

Number of occurrence of value 1: 1
Number of occurrence of value 47: 3
Number of occurrence of value 78: 1
Number of occurrence of value 143: 2
Number of occurrence of value 195: 1
Number of occurrence of value 259: 1
Number of occurrence of value 297: 2
Number of occurrence of value 326: 3
Number of occurrence of value 399: 2
Number of occurrence of value 449: 2
Number of occurrence of value 505: 1
Number of occurrence of value 553: 1
Number of occurrence of value 603: 2
Number of occurrence of value 659: 1
Number of occurrence of value 712: 1
Number of occurrence of value 754: 1
Number of occurrence of value 812: 1
Number of occurrence of value 865: 2
Number of occurrence of value 905: 2
Number of occurrence of value 955: 2
Number of occurrence of value 1003: 1
Number of occurrence of value 1056: 1
Number of occurrence of value 1088: 2
Number of occurrence of value 1126: 1
Number of occurrence of value 1166: 1
Number of occurrence of value 1219: 1
Number of occurrence of value 1265: 2
Number of occurrence of value 1327: 1
Number of occurrence of value 1392: 2
Number of occurrence of value 1437: 2
Number of occurrence of value 1497: 1
Number of occurrence of value 1544: 2
Number of occurrence of value 1586: 3
Number of occurrence of value 1640: 2
Number of occurrence of value 1702: 1
Number of occurrence of value 1755: 2
Number of occurrence of value 1797: 1
```

```
Number of occurrence of value 1844: 1
Number of occurrence of value 1908: 1
Number of occurrence of value 1965: 2
Number of occurrence of value 2018: 3
Number of occurrence of value 2059: 2
Number of occurrence of value 2100: 1
Number of occurrence of value 2151: 1
Number of occurrence of value 2209: 1
Number of occurrence of value 2258: 2
Number of occurrence of value 2309: 1
Number of occurrence of value 2344: 1
Number of occurrence of value 2399: 2
Number of occurrence of value 2442: 2
Number of occurrence of value 2494: 1
Number of occurrence of value 2536: 2
Number of occurrence of value 2595: 1
Number of occurrence of value 2653: 3
Number of occurrence of value 2701: 1
Number of occurrence of value 2758: 2
Number of occurrence of value 2807: 1
Number of occurrence of value 2845: 2
Number of occurrence of value 2903: 1
Number of occurrence of value 2945: 1
Number of occurrence of value 2986: 3
Number of occurrence of value 3024: 1
Number of occurrence of value 3084: 2
Number of occurrence of value 3119: 1
Number of occurrence of value 3183: 1
Number of occurrence of value 3239: 2
Number of occurrence of value 3295: 1
Number of occurrence of value 3343: 2
Number of occurrence of value 3394: 2
Number of occurrence of value 3462: 1
Number of occurrence of value 3526: 2
Number of occurrence of value 3563: 2
Number of occurrence of value 3617: 2
Number of occurrence of value 3667: 1
Number of occurrence of value 3723: 1
Number of occurrence of value 3767: 1
Number of occurrence of value 3840: 1
Number of occurrence of value 3913: 1
Number of occurrence of value 3956: 1
```

```
Number of occurrence of value 4004: 1
Number of occurrence of value 4037: 2
Number of occurrence of value 4089: 2
Number of occurrence of value 4144: 3
Number of occurrence of value 4176: 1
Number of occurrence of value 4224: 1
Number of occurrence of value 4282: 1
Number of occurrence of value 4331: 1
Number of occurrence of value 4369: 2
Number of occurrence of value 4428: 1
Number of occurrence of value 4478: 1
Number of occurrence of value 4531: 1
Number of occurrence of value 4569: 1
Number of occurrence of value 4628: 2
Number of occurrence of value 4663: 1
Number of occurrence of value 4712: 1
Number of occurrence of value 4759: 2
Number of occurrence of value 4798: 3
Number of occurrence of value 4840: 1
Number of occurrence of value 4898: 2
Number of occurrence of value 4939: 1

Printing occurrences of 10 values that is not in the array

Number of occurrence of value 5001: 0
Number of occurrence of value 5002: 0
Number of occurrence of value 5003: 0
Number of occurrence of value 5004: 0
Number of occurrence of value 5005: 0
Number of occurrence of value 5006: 0
Number of occurrence of value 5007: 0
Number of occurrence of value 5008: 0
Number of occurrence of value 5009: 0
Number of occurrence of value 5010: 0

Mode of the BSTHeapTree: 974
It's occurrence value  : 4

Occurrence values of 100 elements that is in the array before removing and after removing them
1) Value: 1 Occurrence before removing: 1 Occurrence after removing: 0
2) Value: 37 Occurrence before removing: 1 Occurrence after removing: 0
3) Value: 57 Occurrence before removing: 1 Occurrence after removing: 0
```

4) Value: 86 Occurrence before removing: 2 Occurrence after removing: 1
5) Value: 132 Occurrence before removing: 2 Occurrence after removing: 1
6) Value: 170 Occurrence before removing: 2 Occurrence after removing: 1
7) Value: 204 Occurrence before removing: 1 Occurrence after removing: 0
8) Value: 246 Occurrence before removing: 2 Occurrence after removing: 1
9) Value: 287 Occurrence before removing: 1 Occurrence after removing: 0
10) Value: 306 Occurrence before removing: 1 Occurrence after removing: 0
11) Value: 326 Occurrence before removing: 3 Occurrence after removing: 2
12) Value: 386 Occurrence before removing: 3 Occurrence after removing: 2
13) Value: 425 Occurrence before removing: 1 Occurrence after removing: 0
14) Value: 451 Occurrence before removing: 3 Occurrence after removing: 2
15) Value: 498 Occurrence before removing: 2 Occurrence after removing: 1
16) Value: 535 Occurrence before removing: 1 Occurrence after removing: 0
17) Value: 568 Occurrence before removing: 2 Occurrence after removing: 1
18) Value: 599 Occurrence before removing: 3 Occurrence after removing: 2
19) Value: 628 Occurrence before removing: 2 Occurrence after removing: 1
20) Value: 675 Occurrence before removing: 2 Occurrence after removing: 1
21) Value: 712 Occurrence before removing: 1 Occurrence after removing: 0
22) Value: 741 Occurrence before removing: 1 Occurrence after removing: 0
23) Value: 772 Occurrence before removing: 1 Occurrence after removing: 0
24) Value: 814 Occurrence before removing: 3 Occurrence after removing: 2
25) Value: 843 Occurrence before removing: 1 Occurrence after removing: 0
26) Value: 885 Occurrence before removing: 1 Occurrence after removing: 0
27) Value: 916 Occurrence before removing: 2 Occurrence after removing: 1
28) Value: 944 Occurrence before removing: 3 Occurrence after removing: 2
29) Value: 985 Occurrence before removing: 1 Occurrence after removing: 0
30) Value: 1014 Occurrence before removing: 1 Occurrence after removing: 0
31) Value: 1056 Occurrence before removing: 1 Occurrence after removing: 0
32) Value: 1079 Occurrence before removing: 3 Occurrence after removing: 2
33) Value: 1100 Occurrence before removing: 3 Occurrence after removing: 2
34) Value: 1129 Occurrence before removing: 1 Occurrence after removing: 0
35) Value: 1160 Occurrence before removing: 1 Occurrence after removing: 0
36) Value: 1189 Occurrence before removing: 1 Occurrence after removing: 0
37) Value: 1229 Occurrence before removing: 1 Occurrence after removing: 0
38) Value: 1261 Occurrence before removing: 1 Occurrence after removing: 0
39) Value: 1296 Occurrence before removing: 3 Occurrence after removing: 2
40) Value: 1343 Occurrence before removing: 2 Occurrence after removing: 1
41) Value: 1392 Occurrence before removing: 2 Occurrence after removing: 1
42) Value: 1418 Occurrence before removing: 2 Occurrence after removing: 1
43) Value: 1468 Occurrence before removing: 1 Occurrence after removing: 0
44) Value: 1499 Occurrence before removing: 1 Occurrence after removing: 0
45) Value: 1534 Occurrence before removing: 2 Occurrence after removing: 1

```
81) Value: 2807 Occurrence before removing: 1 Occurrence after removing: 0
82) Value: 2834 Occurrence before removing: 2 Occurrence after removing: 1
83) Value: 2872 Occurrence before removing: 1 Occurrence after removing: 0
84) Value: 2906 Occurrence before removing: 1 Occurrence after removing: 0
85) Value: 2939 Occurrence before removing: 1 Occurrence after removing: 0
86) Value: 2965 Occurrence before removing: 2 Occurrence after removing: 1
87) Value: 2993 Occurrence before removing: 1 Occurrence after removing: 0
88) Value: 3020 Occurrence before removing: 1 Occurrence after removing: 0
89) Value: 3053 Occurrence before removing: 1 Occurrence after removing: 0
90) Value: 3095 Occurrence before removing: 2 Occurrence after removing: 1
91) Value: 3119 Occurrence before removing: 1 Occurrence after removing: 0
92) Value: 3157 Occurrence before removing: 2 Occurrence after removing: 1
93) Value: 3201 Occurrence before removing: 1 Occurrence after removing: 0
94) Value: 3249 Occurrence before removing: 1 Occurrence after removing: 0
95) Value: 3282 Occurrence before removing: 1 Occurrence after removing: 0
96) Value: 3309 Occurrence before removing: 1 Occurrence after removing: 0
97) Value: 3354 Occurrence before removing: 1 Occurrence after removing: 0
98) Value: 3389 Occurrence before removing: 1 Occurrence after removing: 0
99) Value: 3432 Occurrence before removing: 1 Occurrence after removing: 0
100) Value: 3476 Occurrence before removing: 1 Occurrence after removing: 0

Occurrence values of 10 elements that is NOT in the array before removing and after removing them
1) Value: 5001 Occurrence before removing: 0 Occurrence after removing: 0
2) Value: 5002 Occurrence before removing: 0 Occurrence after removing: 0
3) Value: 5003 Occurrence before removing: 0 Occurrence after removing: 0
4) Value: 5004 Occurrence before removing: 0 Occurrence after removing: 0
5) Value: 5005 Occurrence before removing: 0 Occurrence after removing: 0
6) Value: 5006 Occurrence before removing: 0 Occurrence after removing: 0
7) Value: 5007 Occurrence before removing: 0 Occurrence after removing: 0
8) Value: 5008 Occurrence before removing: 0 Occurrence after removing: 0
9) Value: 5009 Occurrence before removing: 0 Occurrence after removing: 0
10) Value: 5010 Occurrence before removing: 0 Occurrence after removing: 0

After removing 100 elements from BSTHeapTree this is the last shape of it

|       ┌── Heap: (P: 4997,3 L: 4977,1 R: 4993,2) (P: 4977,1 L: 4953,2 R: 4967,1) (P: 4993,2 L: 498
|       |   └── Heap: (P: 4983,1 L: 4982,1 R: 4954,2) (P: 4982,1 L: 4951,1 R: 4958,1) (P: 4954,2 L:
|       |       └── Heap: (P: 4981,1 L: 4976,1 R: 4979,1) (P: 4976,1 L: 4940,1 R: 4969,1) (P: 4979,
|   ┌── Heap: (P: 4939,1 L: 4911,1 R: 4810,1) (P: 4911,1 L: 4826,2 R: 4833,1) (P: 4810,1 L: 4739,3
|   |   |   ┌── Heap: (P: 4937,1 L: 4934,2 R: 4931,1) (P: 4934,2 L: 4928,2 R: 4926,2) (P: 4931,1 L:
|   |   |   |   └── Heap: empty
|   |   └── Heap: (P: 4925,1 L: 4796,2 R: 4906,2) (P: 4796,2 L: 4790,1 R: 4768,2) (P: 4906,2 L: 481
```

6. Analyzing Time Complexity

```
private void maxHeapify(int index) {
    // temp data ** nodes **
    E node, leftChild, rightChild;Θ(1)

    try { node       = data.get(index); }       catch (IndexOutOfBoundsException i) { node = getMin(); } Θ(1)
    try { leftChild  = data.get(left(index)); } catch (IndexOutOfBoundsException i) { leftChild  = getMax(); } Θ(1)
    try { rightChild = data.get(right(index)); } catch (IndexOutOfBoundsException i) { rightChild = getMax(); } Θ(1)


    // check if given index is leaf node
    if(!isLeaf(index)) { Θ(1)
        // check if given node's children are bigger than their parent
        if( (node.compareTo(leftChild) < 0) || (node.compareTo(rightChild) < 0) ) { Θ(1)
            // if first one is bigger swap with it's parent and call maxHeapify with that index
            if(leftChild.compareTo(rightChild) > 0) { Θ(1)
                swap(index, left(index)); Θ(1)
                maxHeapify(left(index));
            }
            else {
                swap(index, right(index)); Θ(1)
                maxHeapify(right(index));
            }
        }
    }
}
```

$$T(n) = k + T(n/2)$$
$$T(n/2) = k + T(n/4)$$
$$\dots$$
$$T(n) = O(\log n)$$

```
private void swap(int index1, int index2) {
    // Backup first data and occurrence by using temp value
    E   tempData       = null; Θ(1)
    int tempOccurrence = 0;    Θ(1)

    try {
        tempData       = data.get(index1); Θ(1)
        tempOccurrence = occurrences.get(index1); Θ(1)
    } catch (IndexOutOfBoundsException e) {
        System.out.println("index: " + index1 + "size: " + data.size()); Θ(1)
    }

    // Inserting second data and occurrence into first index
    data.set(index1, data.get(index2)); Θ(1)
    occurrences.set(index1, occurrences.get(index2)); Θ(1)

    // Inserting first data and occurrence into second index by using temp
    data.set(index2, tempData); Θ(1)
    occurrences.set(index2, tempOccurrence); Θ(1)
}
```

Θ(1)

$$T(n) = \Theta(1)$$

```java
/**
 * Swaps all data in MaxHeap class according to Max Heap data structure's rule
 */
public void builtMaxHeap() {
    for(int i = (size/2); i > 0; i--)
        maxHeapify(i);
}
```
O(logn)

O(n.logn)

```java
/**
 * Returns root element
 * @return Root element of Max Heap
 */
public E getMax() { return data.get(1); }
```
Θ(1)

```java
/**
 * Changes Max value of the heap with given parameter
 * @param item New value of max
 */
public void setMax(E item) { data.set(1, item); }
```
Θ(1)

```java
/**
 * Returns least important element
 * @return Least important element of Max Heap
 */
public E getMin() { return data.get(size); }
```
Θ(1)

```java
/**
 * Returns size of the current MaxHeap tree
 * @return Size of the current MaxHeap tree
 */
public int getSize() { return size; }
```
Θ(1)

```java
public void print() {
    if(getSize() < 2)
        System.out.println("(Empty heap)");
    StringBuilder string = new StringBuilder();
    for(int i=1; i <= size/2; i++) {
        string.append(i).append(")\tParent: ");
        try { string.append(data.get(i)); } catch (Exception a) { string.append("empty"); }
        string.append("\t\t Left Child: ");
        try { string.append(data.get(left(i))); } catch (Exception a) { string.append("empty"); }
        string.append("\t  Right Child: ");
        try { string.append( data.get(right(i))); } catch (Exception a) { string.append("empty"); }
        string.append("\n");
    }
    System.out.println(string);
}
```
Θ(1)

Θ(1)

Θ(1)

Θ(n)

Θ(1)

Θ(1)

Θ(1)

Θ(1)

Θ(1)

Θ(1)

$T(n) = \Theta(n)$

```java
/**
 * Returns index of parent of given node
 * @param index Index of the node
 * @return Parent's index of the node
 */
private int parent(int index) { return (index / 2 == 0) ? 1 : (index / 2); }
```
Θ(1)

```java
/**
 * Returns index of left child of given node
 * @param index Index of the node
 * @return Left child's index of the node
 */
private int left(int index) { return 2 * index; }
```
Θ(1)

```java
/**
 * Returns index of right child of given node
 * @param index Index of the node
 * @return Right child's index of the node
 */
private int right(int index) { return (2 * index) + 1; }
```
Θ(1)

```java
/**
 * Returns true if selected node is a leaf(has no children) node
 * @param index Selected node
 * @return True when selected node is a leaf node
 */
private boolean isLeaf(int index) { return index >= (size / 2) && index <= size; }
```
Θ(1)

```java
/**
 * Inserts every data in given parameter into Max Heap
 * @param secondHeap Inserted data
 */
public void merge(MaxHeap<E> secondHeap) {
    E   data; Θ(1)
    int index;Θ(1)

    for(int i=1; i<=secondHeap.size; i++) {
        data  = secondHeap.data.get(i); Θ(1)
        index = searchElement(data); O(n)

        if(index != -1)
            occurrences.set(index, occurrences.get(index) + secondHeap.occurrences.get(i));Θ(1)
        else
            this.insert(secondHeap.data.get(i)); O(logn)
    }
}
```
n times

T(n) = O(n.logn)

O(n.logn)

```java
public int insert(E e) {
    // index of given element if it already exist
    int index; Θ(1)

    // when heap is empty
    if(size == 0) { Θ(1)
        data.add(e); Θ(1)
        occurrences.add(1); Θ(1)
    }                                                          } Θ(1)
    // when given element already exist in the heap
    else if(searchElement(e) != -1) { Θ(1)
        index = searchElement(e); O(n)
        occurrences.set(index, occurrences.get(index) + 1); Θ(1)   } O(n)
        return occurrences.get(index); Θ(1)
    }
    // if heap is full
    else if(size == MAX_SIZE) { Θ(1)
    Θ(1) System.out.println("You are trying to add more than you could. (Max: " + MAX_SIZE + ") (Value which not added: " + e + ")");
    Θ(1) return 0;
    }

    // inserting data and it's occurence
    data.add(e); Θ(1)
    occurrences.add(1); Θ(1)
    // increasing current size of data array
    size++; Θ(1)
    // getting temp size
    int tmpSize = size; Θ(1)
                                        Θ(1)
    while( data.get(tmpSize).compareTo(data.get(parent(tmpSize))) > 0 ) {
        swap(tmpSize, parent(tmpSize)); Θ(1)                   } O(logn)
        tmpSize = parent(tmpSize); Θ(1)
    }

    return 1; Θ(1)
}
```

$$T(n)_w = \Theta(\log n)$$
$$T(n)_b = \Theta(1)$$
$$T(n) = O(\log n)$$

```java
/**
 * Search given element in the Max Heap. Returns element's index if it is present. (-1 when eleme
 * @param e Element that method looks for
 * @return Index of that element. Returns -1 when element does not exist.
 */
public int searchElement(E e) {
    // searching through every element in the heap
    for(int i=1; i<=size; i++)
        if(e.compareTo(data.get(i)) == 0) Θ(1)   } O(n)
            return i; Θ(1)                                  } O(n)

    return -1; Θ(1)
}
```

```java
public E removeWithPriority(int priority) {
    int lastIndex = size;Θ(1)
    E parameter = data.get(priority);Θ(1)
    E minData   = getMin();Θ(1)

    // if number of occurrences of given data is not 1, then reduce it by 1
    if(occurrences.get(priority) > 1) {Θ(1)
        occurrences.set(priority, occurrences.get(priority) - 1);Θ(1)
        return parameter;Θ(1)
    }

    // swapping least important element with given element if it is not least important element
    if(parameter.compareTo(minData) != 0)Θ(1)
        swap(priority, lastIndex);Θ(1)

    // removing last element
    data.remove(lastIndex);O(n)

    // decreasing array size by 1
    size--;Θ(1)

    return parameter;Θ(1)
}
```

$$T(n)_w = \Theta(n)$$

$$T(n)_b = \Theta(1)$$

$$T(n) = O(n)$$

```java
public E setLast(E element) throws NullPointerException {
    if(lastItemReturned == null)
        throw new NullPointerException();Θ(1)

    // changing value of last returned element
    data.set(lastItemReturnedIndex, element);Θ(1)


    int currIndex = lastItemReturnedIndex;Θ(1)
    // if it is bigger than parent node then swap it until it is not
    if(data.get(currIndex).compareTo(data.get(parent(currIndex))) > 0) {Θ(1)
        while( data.get(currIndex).compareTo(data.get(parent(currIndex))) > 0 ) {Θ(1)
            swap(currIndex, parent(currIndex));Θ(1)
            currIndex = parent(currIndex);Θ(1)
        }
    }

    builtMaxHeap();O(n.logn)

    return lastItemReturned;
}
```

O(logn)

$$T(n) = O(\log n)$$

```java
/**
 * Returns mode(most occurred) element of the BSTHeapTree
 * @return Element which has biggest occurrence value
 */
public E find_mode() { return mode; } Θ(1)

/**
 * Return occurrence value of the mode element
 * @return occurrence value of the mode element
 */
public int find_mode_occurrence() { return modeOccurrence; } Θ(1)
```

```java
public int getOccurrence(E element) {
    int number; Θ(1)

    try {
        number = occurrences.get(searchElement(element)); Θ(1)
    } catch (IndexOutOfBoundsException exception) {
        number = 0; Θ(1)
    }

    return number; Θ(1)
}
```

$$T(n) = \Theta(n)$$