GIT Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework 7 Report

Ahmet Tuğkan Ayhan

1901042692

1. System Requirements

a. Functional Requirements

Part1

- There are 3 methods that is implemented for NavigableSet using SkipList:
  1) boolean add(E e)
  2) boolean remove(E e)
  3) Iterator<E> descendindIterator()
- add method let's you add an element to the NavigableSet. Uses SkipList methods to do it.
- remove method let's you remove given parameter from the NavigableSet if the item exist in the Set. Uses SkipList methods to remove.
- descendindIterator return an Iterator object. It iterates through the items in the NavigableSet by descending order.

- There are 4 methods that is implemented for NavigableSet using AVL Tree:
  1) boolean add(E e)
  2) Iterator<E> iterator()
  3) NavigableSet<E> headSet(E toElement)
  4) NavigableSet<E> tailSet(E fromElement)
- add method let's you add an element to the NavigableSet. Uses AVL Tree methods to do it.
- iterator method returns an Iterator object. It iterates through the items in the NavigableSet in increasing order.
- headSet method returns a NavigableSet object that contains all elements in the original set that is less than given parameter.
- tailSet method returns a NavigableSet object that contains all elements in the original set that is bigger than given parameter.

Part2

- There is 1 method that is implemented for part2:
  1) String isAVLorRBT(BinarySearchTree<E> tree)
- isAVLorRBT method returns a String object that contains tree type converted into a string. It controls properties for AVL Tree and Red-Black Tree, then decides on which one is correct for this tree and assigns it. There can be 3 output:
  1) This is an AVL Tree
  2) This is a Red-Black Tree & rules are broken for AVL Tree
  3) This tree is neither an AVL Tree or a Red-Black Tree

Part3

- There is no requirement for this part. This part only implements classes that is available in the course book. Implemented classes & data structures:
  1) BinarySearchTree
  2) Red-Black Tree
  3) 2-3 Tree
  4) B-Tree
  5) SkipList

b. Non-Functional Requirements

c.

Part1

- add method shouldn’t insert same elements to the NavigableSet
- remove method should return null if there is no element in the NavigableSet or given parameter doesn’t exist in the NavigableSet
- iterator and descendingIterator methods must iterate until every element in the NavigableSet is passed at least and at most 1 time.
- returned NavigableSet from headSet method shouldn’t contain any bigger number than given parameter.
- returned NavigableSet from tailSet method shouldn’t contain any less number than given parameter.

Part2

- isAVLorRBT method must calculate the height of every node in the BinarySearchTree that is given as parameter. Then should compare left and right node’s heights to decide the tree type.

Part3

- Driver code must generate randomly generated(non-repeating) numbers with the size of 10.000, 20.000, 40.000 and 80.000.
- Then it should insert these random numbers into the 5 data structure type.
- Then driver code must generate another 100 randomly generated(non-repeating) numbers and it should measure the time while inserting these 100 number into the data structures.
- Finally, driver code should calculate average running time for each data size. Then compares increase rate, running time, etc.

2. Class Diagram

Diagram was so big, I seperated it from the report. Class diagram is located in zip file as pdf file with the name “class_diagram”.

3. Use Case Diagram

Diagram was so big, I seperated it from the report. Use Case diagram is located in zip file as pdf file with the name “useCase_diagram”.

4. Problem Solution Approach

Part1

NavigableSet – SkipList

To implement NavigableSet using skiplist I first created a class for it(NS_SkipList). Then I initialized a SkipList object inside of the class as a private data field.

Normally, you can add same numbers into a skiplist data structure but while I'm implementing the add method for NavigableSet I prevented this happening.

Remove method uses SkipList's remove method but I added some extra conditions where the method should return null to prevent exceptions occur.

descendingIterator method initializes an ArrayList object where SkipList object's elements are copied to. By doing so, I was able to use Collections.reverse method to reverse ArrayList object. Then I returned original iterator of the ArrayList object

NavigableSet – AVL

To implement NavigableSet using AVL Tree I first created a class for it(NS_AVLTree). Then initialized a AVL Tree object inside of the class as a private data field.

add method acts same as skiplist version

iterator method initializes an ArrayList object and sends it into InOrderAVLTreeTraversel method which I wrote inside of the NS_AVLTree. This method takes root node of the AVL Tree object. Then makes a full inorder tree traversal and adds all data into ArrayList object. Finally after traversel is done I returned the ArrayList object's original iterator.

headSet method uses InOrderAVLTreeTraversel_head method, it is as same as normal one but this time it only adds elements to the ArrayList object which is lower than given parameter.

tailSet method uses InOrderAVLTreeTraversel_tail method, it is as same as normal one but this time it only adds elements to the ArrayList object which is bigger than given parameter.

Part2

The method I wrote for determining to Tree type gets a BinarySearchTree parameter and returns a String object which holds the tree type.

To make things more easy to handle I wrote helper methods.

isAVL method returns a boolean value if given BinarySearchTree is an AVL Tree. If tree is null it s a AVL Tree, if left and right height difference of the nodes are is always between 1, -1 and 0 for every node in the tree then it is a AVL Tree.

isRBT method returns a boolean value if given BinarySearchTree is an Red-Black Tree. If tree is null then it is a AVL tree, if root node's color is not black then it is not a Red-Black Tree.

Finally, if height difference of the nodes are always between -2 and 2 for every node in the tree then it is a AVL Tree.

Part3

For the last part, I took implementations of the data structures from course book and wrote a driver code to show their speed and increase rate according to different data sizes. I also draw a graph to show "running time-problem size" relation of the data structures. Finally I showed their increase rate in an order.

5. Test Cases

** PART 1 **

NavigableSet - Skiplist

1) Inserting 20 random numbers between 0-100
2) Printing out the inserted numbers by using descendingIterator
3) Removing 10 randomly selected numbers from the set
4) Printing out the set after removing 10 numbers

NavigableSet - AVL

5) Inserting 20 random numbers between 0-100
6) Printing out the inserted numbers by using Iterator
7) Getting all numbers that is bigger than 50 by using tailSet into a new NavigableSet object
8) Printing out the inserted numbers by using Iterator
9) Getting all numbers that is less than 50 by using headSet into a new NavigableSet object
10) Printing out the inserted numbers by using Iterator

** PART 2 **

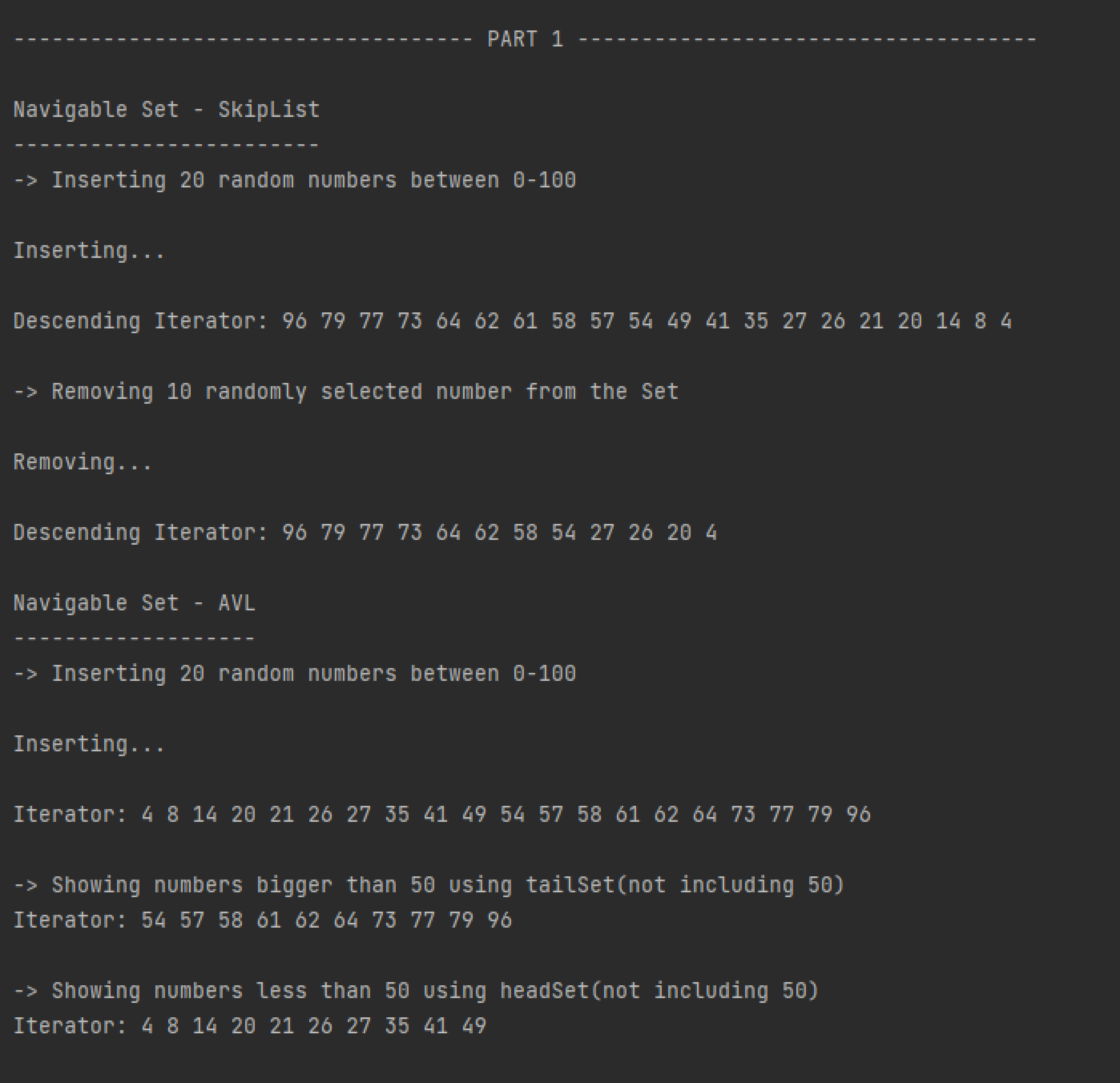
11) Initialize 3 BinarySearchTree object
12) Insert 10 numbers into each of them
13) Print out the current look of the Trees.
14) Decide which Tree type is correct for the trees and print out their tree types.

** PART 3 **

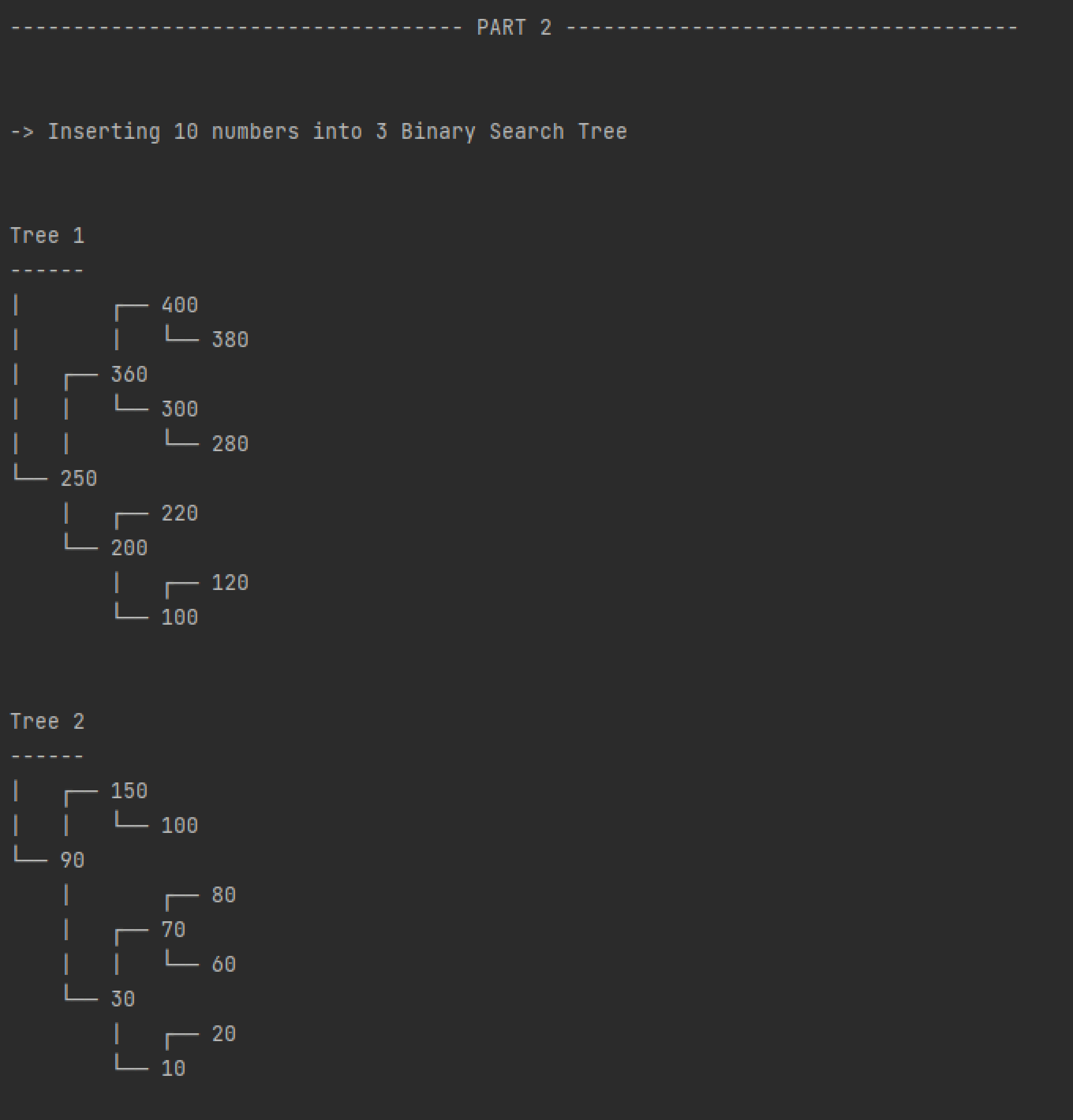
15) Initialize 40 instance for each data structure type. (200 total)
16) Insert 10.000, 20.000, 40.000, 80.000 random(non-repeating) numbers to these instances in an order.
    ! (0-10th instances 10.000 data, 10-20th instances 20.000 data, 20-30th instances 40.000 data, 30-40th instances 80.000 data)
17) Insert another 100 random(non-repeating) number into the data structures and this time measure the time consumed by each data structure.
18) Calculate the average time consumption for every data size and data structure
19) Print out the average time consumtion for every data size and data structure
20) Make the comparison for time consumption

21) Print out data structure with the highest total time consumption
22) Print out data structure with the lowest total time consumption

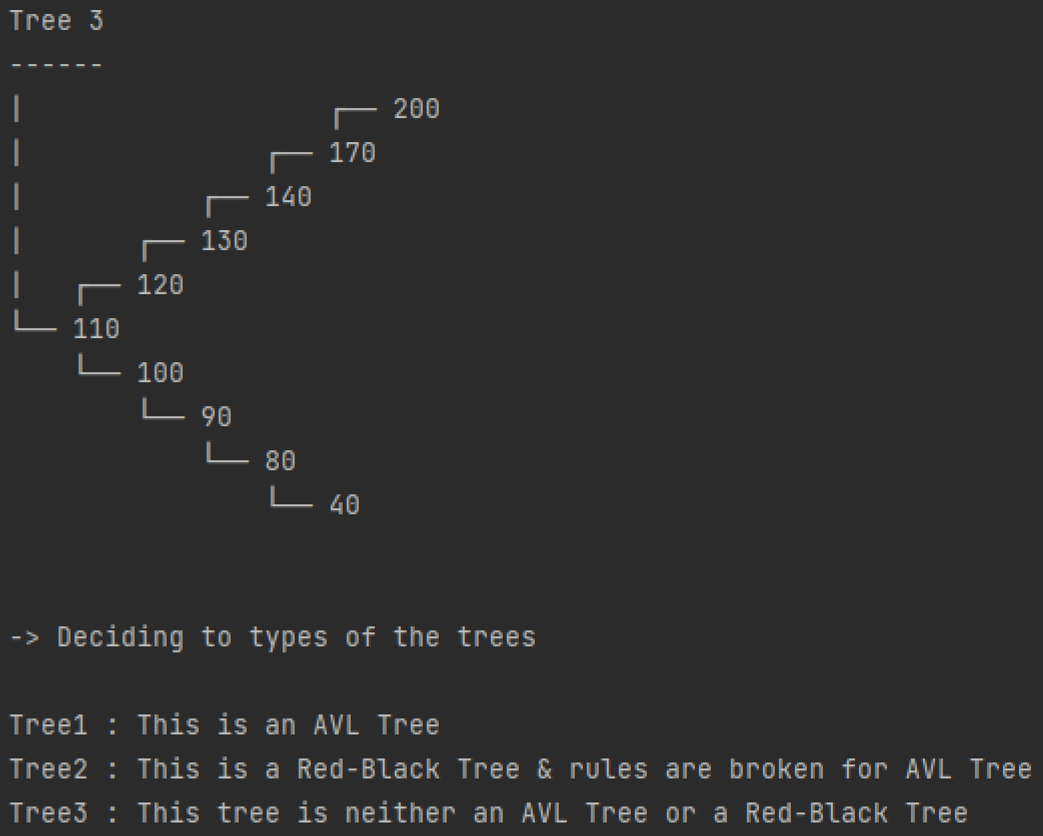
6. Running command and Results



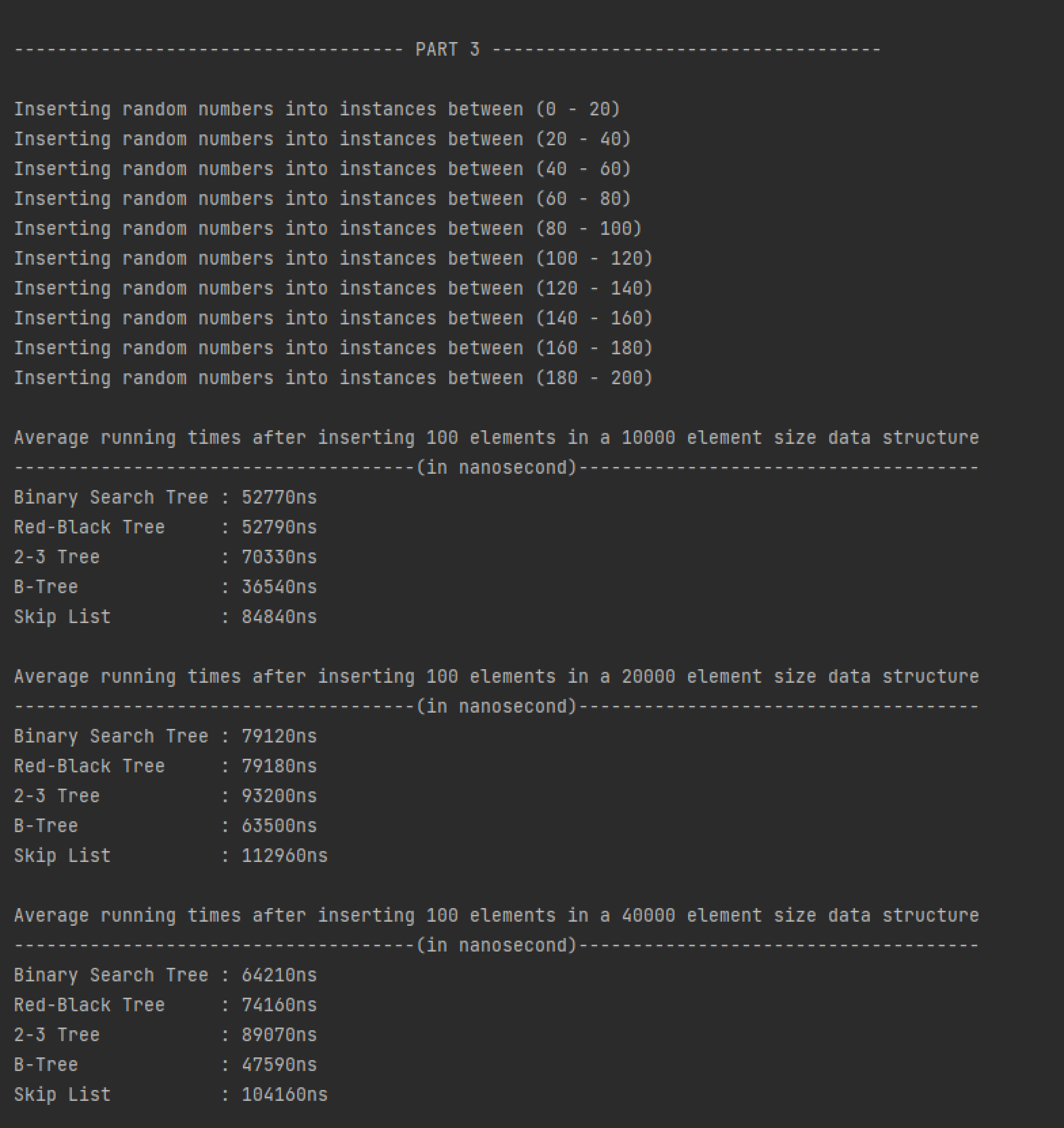
```
------------------------------------ PART 1 ------------------------------------

Navigable Set - SkipList
------------------------
-> Inserting 20 random numbers between 0-100

Inserting...

Descending Iterator: 96 79 77 73 64 62 61 58 57 54 49 41 35 27 26 21 20 14 8 4

-> Removing 10 randomly selected number from the Set

Removing...

Descending Iterator: 96 79 77 73 64 62 58 54 27 26 20 4

Navigable Set - AVL
-------------------
-> Inserting 20 random numbers between 0-100

Inserting...

Iterator: 4 8 14 20 21 26 27 35 41 49 54 57 58 61 62 64 73 77 79 96

-> Showing numbers bigger than 50 using tailSet(not including 50)
Iterator: 54 57 58 61 62 64 73 77 79 96

-> Showing numbers less than 50 using headSet(not including 50)
Iterator: 4 8 14 20 21 26 27 35 41 49
```



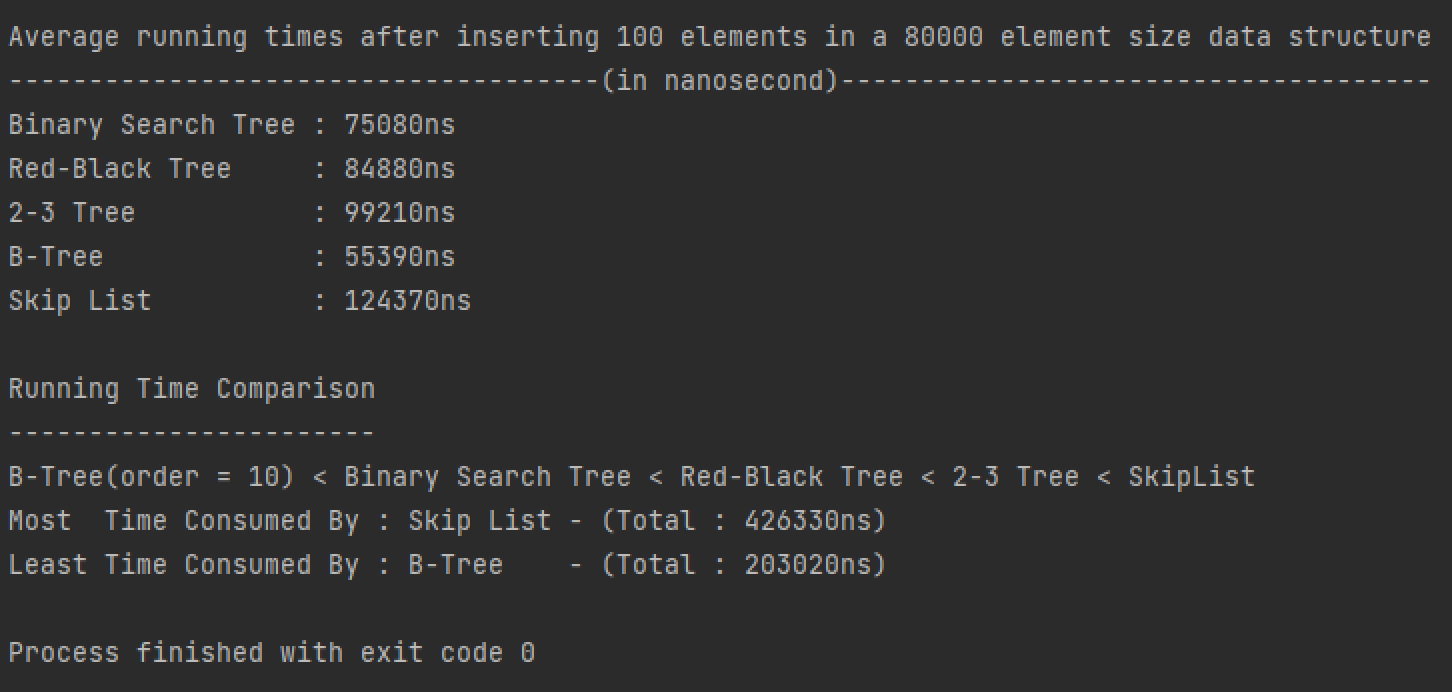
```
------------------------------------ PART 2 ------------------------------------



-> Inserting 10 numbers into 3 Binary Search Tree



Tree 1
------
│       ┌── 400
│       │   └── 380
│   ┌── 360
│   │   └── 300
│   │       └── 280
└── 250
    │   ┌── 220
    └── 200
        │   ┌── 120
        └── 100



Tree 2
------
│   ┌── 150
│   │   └── 100
└── 90
    │       ┌── 80
    │   ┌── 70
    │   │   └── 60
    └── 30
        │   ┌── 20
        └── 10
```



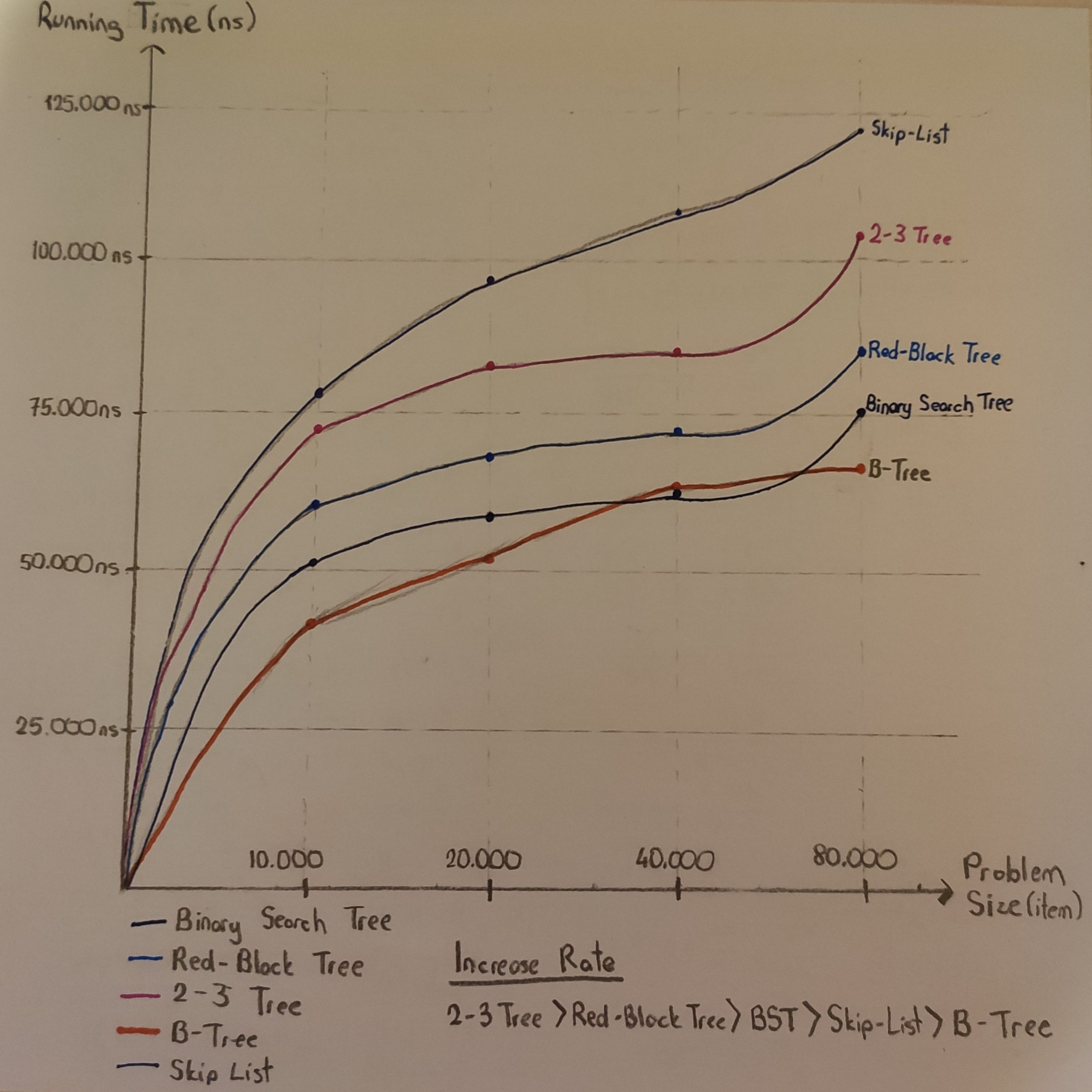
```
Tree 3
------
|                   ┌── 200
|               ┌── 170
|           ┌── 140
|       ┌── 130
|   ┌── 120
└── 110
    └── 100
        └── 90
            └── 80
                └── 40


-> Deciding to types of the trees

Tree1 : This is an AVL Tree
Tree2 : This is a Red-Black Tree & rules are broken for AVL Tree
Tree3 : This tree is neither an AVL Tree or a Red-Black Tree
```



```
----------------------------------- PART 3 -----------------------------------

Inserting random numbers into instances between (0 - 20)
Inserting random numbers into instances between (20 - 40)
Inserting random numbers into instances between (40 - 60)
Inserting random numbers into instances between (60 - 80)
Inserting random numbers into instances between (80 - 100)
Inserting random numbers into instances between (100 - 120)
Inserting random numbers into instances between (120 - 140)
Inserting random numbers into instances between (140 - 160)
Inserting random numbers into instances between (160 - 180)
Inserting random numbers into instances between (180 - 200)

Average running times after inserting 100 elements in a 10000 element size data structure
-------------------------------------(in nanosecond)-------------------------------------
Binary Search Tree : 52770ns
Red-Black Tree     : 52790ns
2-3 Tree           : 70330ns
B-Tree             : 36540ns
Skip List          : 84840ns

Average running times after inserting 100 elements in a 20000 element size data structure
-------------------------------------(in nanosecond)-------------------------------------
Binary Search Tree : 79120ns
Red-Black Tree     : 79180ns
2-3 Tree           : 93200ns
B-Tree             : 63500ns
Skip List          : 112960ns

Average running times after inserting 100 elements in a 40000 element size data structure
-------------------------------------(in nanosecond)-------------------------------------
Binary Search Tree : 64210ns
Red-Black Tree     : 74160ns
2-3 Tree           : 89070ns
B-Tree             : 47590ns
Skip List          : 104160ns
```



```
Average running times after inserting 100 elements in a 80000 element size data structure
-------------------------------------(in nanosecond)-------------------------------------
Binary Search Tree : 75080ns
Red-Black Tree     : 84880ns
2-3 Tree           : 99210ns
B-Tree             : 55390ns
Skip List          : 124370ns

Running Time Comparison
-----------------------
B-Tree(order = 10) < Binary Search Tree < Red-Black Tree < 2-3 Tree < SkipList
Most  Time Consumed By : Skip List - (Total : 426330ns)
Least Time Consumed By : B-Tree    - (Total : 203020ns)

Process finished with exit code 0
```



Increase Rate

2-3 Tree > Red-Block Tree > BST > Skip-List > B-Tree