Gebze Technical University

CSE222 – Data Structures and Algorithms

HW7 – Part3

Subject : Analyzing Several Tree structure and SkipList structure

Nevzat Seferoglu

171044024

*Problem Description :

Compare the performance of the following data structures.

- Regular binary search tree
- Red-Black tree implementation in the book
- Red Black tree implementation in java
- B-tree implementation in the book
- Skip list implementation in the book
- Skip list implementation in java
- Skip list in question Q2

For each data structure, do the following:

- Insert a collection of randomly generated numbers. Perform this operation 10 times for 10.000, 20.000, 40.000 and 80.000 random numbers (10 times for each). So, you will have 10 instances of each data structure for each 4 different sizes. There should be 240 data structure in total.
- Verify that data structures are built correctly (for example, for BST, perform an inorder traversal).
- Compare the run-time performance of the insertion operation for the data structures. Insert 10 extra random numbers into the structures you built. Measure the running time and calculate the average running time for each data structure and four different problem size. Compare the running times and their increase.
- Compare the run-time performance of the deletion operation for the data structures. Perform 10 successful deletion operations from the structures you built. Measure the running time and calculate the average running time for each data structure and four different problem size.
  Compare the running times and their increase.

*Solution Approach :

Some implementation in the book was not fully completed. Firstly, I tried to implement these missing functions.

*Note : I could not make Part2 which is related to customized skip list implementation. Therefore, I could make a measurement about that part. There is only another exception for BTree. Because I implement add functionality for BTree , I could not implement deletion part of it.

I will comment and analyze the one by one of each structure and racing each other.

*Analyzing 10K of each structure :

*BinarySearchTree :

BinarySearchTree 1. 10K insertion: 11 ms.

BinarySearchTree 2. 10K insertion: 3 ms.

BinarySearchTree 3. 10K insertion: 2 ms.

BinarySearchTree 4. 10K insertion: 2 ms.

BinarySearchTree 5. 10K insertion: 2 ms.

BinarySearchTree 6. 10K insertion: 2 ms.

BinarySearchTree 7. 10K insertion: 3 ms.

BinarySearchTree 8. 10K insertion: 3 ms.

BinarySearchTree 9. 10K insertion: 3 ms.

BinarySearchTree 10. 10K insertion: 3 ms.

Average running time for inserting subsequently: 3.4 ms

*RedBlackTree(book) :

RedBlackTree 1. 10K insertion: 9 ms.

RedBlackTree 2. 10K insertion: 7 ms.

RedBlackTree 3. 10K insertion: 9 ms.

RedBlackTree 4. 10K insertion: 7 ms.

RedBlackTree 5. 10K insertion: 4 ms.

RedBlackTree 6. 10K insertion: 3 ms.

RedBlackTree 7. 10K insertion: 3 ms.

RedBlackTree 8. 10K insertion: 4 ms.

RedBlackTree 9. 10K insertion: 4 ms.

RedBlackTree 10. 10K insertion: 4 ms.

Average running time for inserting subsequently: 5.4 ms

*RedBlackTree(java) :

RedBlackTreeJava 1. 10K insertion: 9 ms.

RedBlackTreeJava 2. 10K insertion: 7 ms.

RedBlackTreeJava 3. 10K insertion: 7 ms.

RedBlackTreeJava 4. 10K insertion: 7 ms.

RedBlackTreeJava 5. 10K insertion: 7 ms.

RedBlackTreeJava 6. 10K insertion: 7 ms.

RedBlackTreeJava 7. 10K insertion: 7 ms.

RedBlackTreeJava 8. 10K insertion: 5 ms.

RedBlackTreeJava 9. 10K insertion: 15 ms.

RedBlackTreeJava 10. 10K insertion: 19 ms.

Average running time for inserting subsequently: 9 ms

***BTree(book) :**

BTree 1. 10K insertion: 8 ms.

BTree 2. 10K insertion: 7 ms.

BTree 3. 10K insertion: 7 ms.

BTree 4. 10K insertion: 7 ms.

BTree 5. 10K insertion: 7 ms.

BTree 6. 10K insertion: 7 ms.

BTree 7. 10K insertion: 7 ms.

BTree 8. 10K insertion: 7 ms.

BTree 9. 10K insertion: 8 ms.

BTree 10. 10K insertion: 7 ms.

Average running time for inserting subsequently: 7.2 ms

***SkipListBook (book) :**

SkipListBook 1. 10K insertion: 10 ms.

SkipListBook 2. 10K insertion: 7 ms.

SkipListBook 3. 10K insertion: 7 ms.

SkipListBook 4. 10K insertion: 7 ms.

SkipListBook 5. 10K insertion: 7 ms.

SkipListBook 6. 10K insertion: 7 ms.

SkipListBook 7. 10K insertion: 7 ms.

SkipListBook 8. 10K insertion: 7 ms.

SkipListBook 9. 10K insertion: 8 ms.

SkipListBook 10. 10K insertion: 8 ms.

Average running time for inserting subsequently: 7.5 ms

***SkipListJava (java) :**

SkipListJava 1. 10K insertion: 24 ms.

SkipListJava 2. 10K insertion: 18 ms.

SkipListJava 3. 10K insertion: 17 ms.

SkipListJava 4. 10K insertion: 38 ms.

SkipListJava 5. 10K insertion: 40 ms.

SkipListJava 6. 10K insertion: 31 ms.

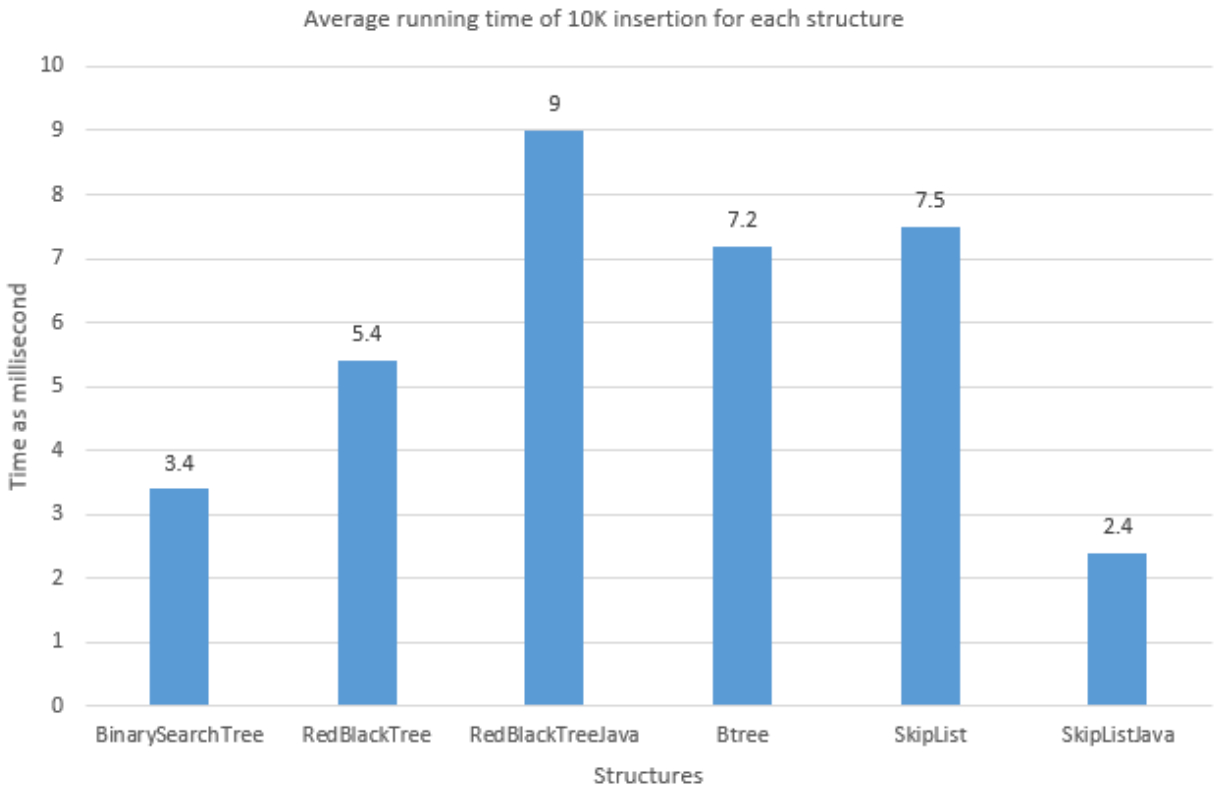
SkipListJava 7. 10K insertion: 16 ms.

SkipListJava 8. 10K insertion: 19 ms.

SkipListJava 9. 10K insertion: 17 ms.

SkipListJava 10. 10K insertion: 20 ms.

Average running time for inserting subsequently: 2.4 ms



*Analyzing 20K of each structure :

*BinarySearchTree :

BinarySearchTree 1. 20K insertion: 7 ms.

BinarySearchTree 2. 20K insertion: 6 ms.

BinarySearchTree 3. 20K insertion: 6 ms.

BinarySearchTree 4. 20K insertion: 7 ms.

BinarySearchTree 5. 20K insertion: 6 ms.

BinarySearchTree 6. 20K insertion: 6 ms.

BinarySearchTree 7. 20K insertion: 7 ms.

BinarySearchTree 8. 20K insertion: 7 ms.

BinarySearchTree 9. 20K insertion: 7 ms.

BinarySearchTree 10. 20K insertion: 41 ms.

Average running time for inserting subsequently: 16.3 ms

*RedBlackTree(book) :

RedBlackTree 1. 20K insertion: 9 ms.

RedBlackTree 2. 20K insertion: 9 ms.

RedBlackTree 3. 20K insertion: 9 ms.

RedBlackTree 4. 20K insertion: 9 ms.

RedBlackTree 5. 20K insertion: 8 ms.

RedBlackTree 6. 20K insertion: 7 ms.

RedBlackTree 7. 20K insertion: 7 ms.

RedBlackTree 8. 20K insertion: 8 ms.

RedBlackTree 9. 20K insertion: 5 ms.

RedBlackTree 10. 20K insertion: 6 ms.

Average running time for inserting subsequently: 7.7 ms

***RedBlackTree(java) :**

RedBlackTreeJava 1. 20K insertion: 34 ms.

RedBlackTreeJava 2. 20K insertion: 21 ms.

RedBlackTreeJava 3. 20K insertion: 11 ms.

RedBlackTreeJava 4. 20K insertion: 9 ms.

RedBlackTreeJava 5. 20K insertion: 9 ms.

RedBlackTreeJava 6. 20K insertion: 8 ms.

RedBlackTreeJava 7. 20K insertion: 8 ms.

RedBlackTreeJava 8. 20K insertion: 8 ms.

RedBlackTreeJava 9. 20K insertion: 8 ms.

RedBlackTreeJava 10. 20K insertion: 9 ms.

Average running time for inserting subsequently: 12.5 ms

***BTree(book) :**

BTree 1. 20K insertion: 17 ms.

BTree 2. 20K insertion: 18 ms.

BTree 3. 20K insertion: 17 ms.

BTree 4. 20K insertion: 16 ms.

BTree 5. 20K insertion: 18 ms.

BTree 6. 20K insertion: 19 ms.

BTree 7. 20K insertion: 20 ms.

BTree 8. 20K insertion: 17 ms.

BTree 9. 20K insertion: 14 ms.

BTree 10. 20K insertion: 14 ms.

Average running time for inserting subsequently: 17 ms

***SkipListBook (book) :**

SkipListBook 1. 20K insertion: 17 ms.

SkipListBook 2. 20K insertion: 17 ms.

SkipListBook 3. 20K insertion: 18 ms.

SkipListBook 4. 20K insertion: 19 ms.

SkipListBook 5. 20K insertion: 21 ms.

SkipListBook 6. 20K insertion: 21 ms.

SkipListBook 7. 20K insertion: 21 ms.

SkipListBook 8. 20K insertion: 18 ms.

SkipListBook 9. 20K insertion: 13 ms.

SkipListBook 10. 20K insertion: 11 ms.

Average running time for inserting subsequently: 17.6 ms

***SkipListJava (java) :**

SkipListJava 1. 20K insertion: 35 ms.

SkipListJava 2. 20K insertion: 11 ms.

SkipListJava 3. 20K insertion: 14 ms.

SkipListJava 4. 20K insertion: 11 ms.

SkipListJava 5. 20K insertion: 10 ms.

SkipListJava 6. 20K insertion: 10 ms.

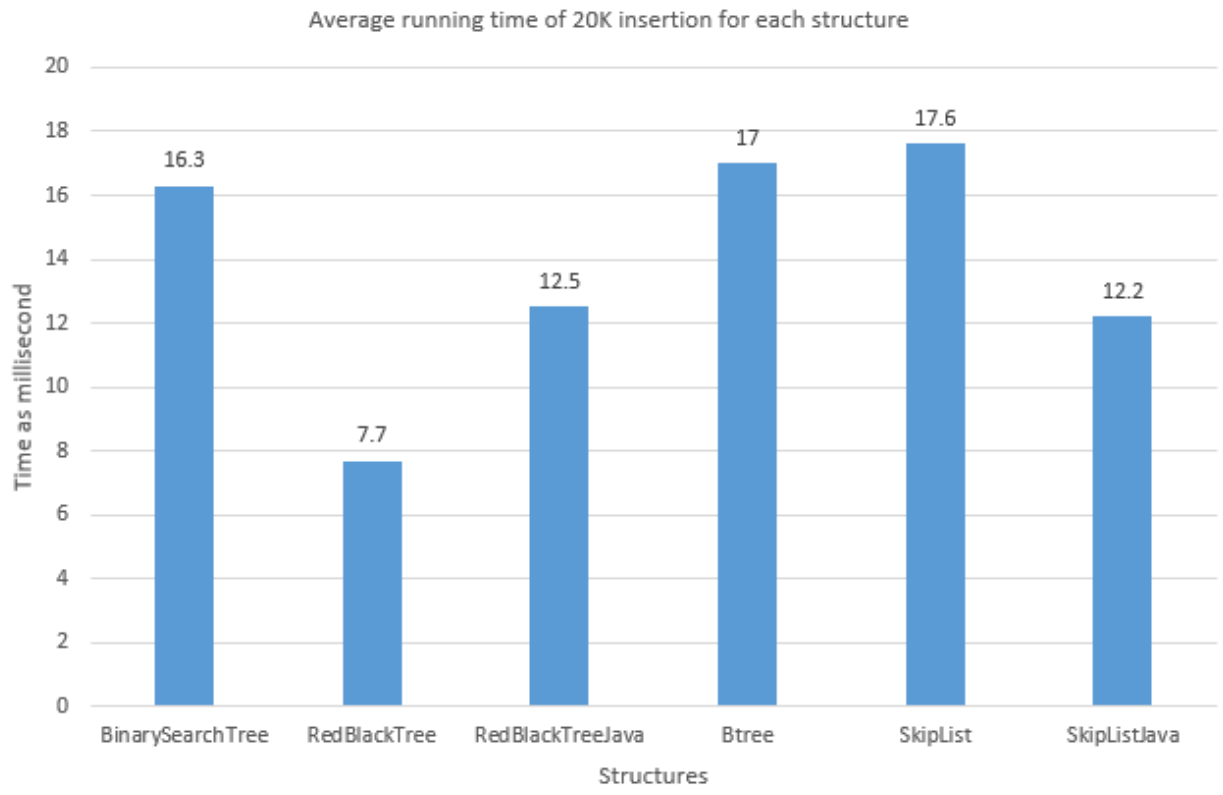
SkipListJava 7. 20K insertion: 10 ms.

SkipListJava 8. 20K insertion: 9 ms.

SkipListJava 9. 20K insertion: 6 ms.

SkipListJava 10. 20K insertion: 6 ms.

Average running time for inserting subsequently: 12.2 ms



*Analyzing 40K of each structure :

*BinarySearchTree :

BinarySearchTree 1. 40K insertion: 27 ms.
BinarySearchTree 2. 40K insertion: 23 ms.
BinarySearchTree 3. 40K insertion: 28 ms.
BinarySearchTree 4. 40K insertion: 17 ms.
BinarySearchTree 5. 40K insertion: 17 ms.
BinarySearchTree 6. 40K insertion: 35 ms.
BinarySearchTree 7. 40K insertion: 15 ms.
BinarySearchTree 8. 40K insertion: 17 ms.
BinarySearchTree 9. 40K insertion: 17 ms.
BinarySearchTree 10. 40K insertion: 41 ms.

Average running time for inserting subsequently: 23,7 ms

*RedBlackTree(book) :

RedBlackTree 1. 40K insertion: 14 ms.
RedBlackTree 2. 40K insertion: 30 ms.
RedBlackTree 3. 40K insertion: 13 ms.
RedBlackTree 4. 40K insertion: 13 ms.
RedBlackTree 5. 40K insertion: 12 ms.
RedBlackTree 6. 40K insertion: 13 ms.
RedBlackTree 7. 40K insertion: 13 ms.
RedBlackTree 8. 40K insertion: 16 ms.
RedBlackTree 9. 40K insertion: 15 ms.
RedBlackTree 10. 40K insertion: 22 ms.

Average running time for inserting subsequently: 14.9 ms

*RedBlackTree(java) :

RedBlackTreeJava 1. 40K insertion: 14 ms.
RedBlackTreeJava 2. 40K insertion: 12 ms.
RedBlackTreeJava 3. 40K insertion: 12 ms.
RedBlackTreeJava 4. 40K insertion: 10 ms.
RedBlackTreeJava 5. 40K insertion: 10 ms.
RedBlackTreeJava 6. 40K insertion: 12 ms.
RedBlackTreeJava 7. 40K insertion: 18 ms.
RedBlackTreeJava 8. 40K insertion: 42 ms.
RedBlackTreeJava 9. 40K insertion: 14 ms.
RedBlackTreeJava 10. 40K insertion: 10 ms.

Average running time for inserting subsequently: 15.4 ms

***BTree(book) :**

BTree 1. 40K insertion: 34 ms.

BTree 2. 40K insertion: 28 ms.

BTree 3. 40K insertion: 27 ms.

BTree 4. 40K insertion: 26 ms.

BTree 5. 40K insertion: 26 ms.

BTree 6. 40K insertion: 31 ms.

BTree 7. 40K insertion: 23 ms.

BTree 8. 40K insertion: 23 ms.

BTree 9. 40K insertion: 20 ms.

BTree 10. 40K insertion: 20 ms.

Average running time for inserting subsequently: 25.8 ms

***SkipListBook (book) :**

SkipListBook 1. 40K insertion: 38 ms.

SkipListBook 2. 40K insertion: 23 ms.

SkipListBook 3. 40K insertion: 23 ms.

SkipListBook 4. 40K insertion: 26 ms.

SkipListBook 5. 40K insertion: 24 ms.

SkipListBook 6. 40K insertion: 33 ms.

SkipListBook 7. 40K insertion: 22 ms.

SkipListBook 8. 40K insertion: 21 ms.

SkipListBook 9. 40K insertion: 19 ms.

SkipListBook 10. 40K insertion: 18 ms.

Average running time for inserting subsequently: 24.7 ms

***SkipListJava (java) :**

SkipListJava 1. 40K insertion: 30 ms.

SkipListJava 2. 40K insertion: 27 ms.

SkipListJava 3. 40K insertion: 20 ms.

SkipListJava 4. 40K insertion: 21 ms.

SkipListJava 5. 40K insertion: 18 ms.

SkipListJava 6. 40K insertion: 18 ms.

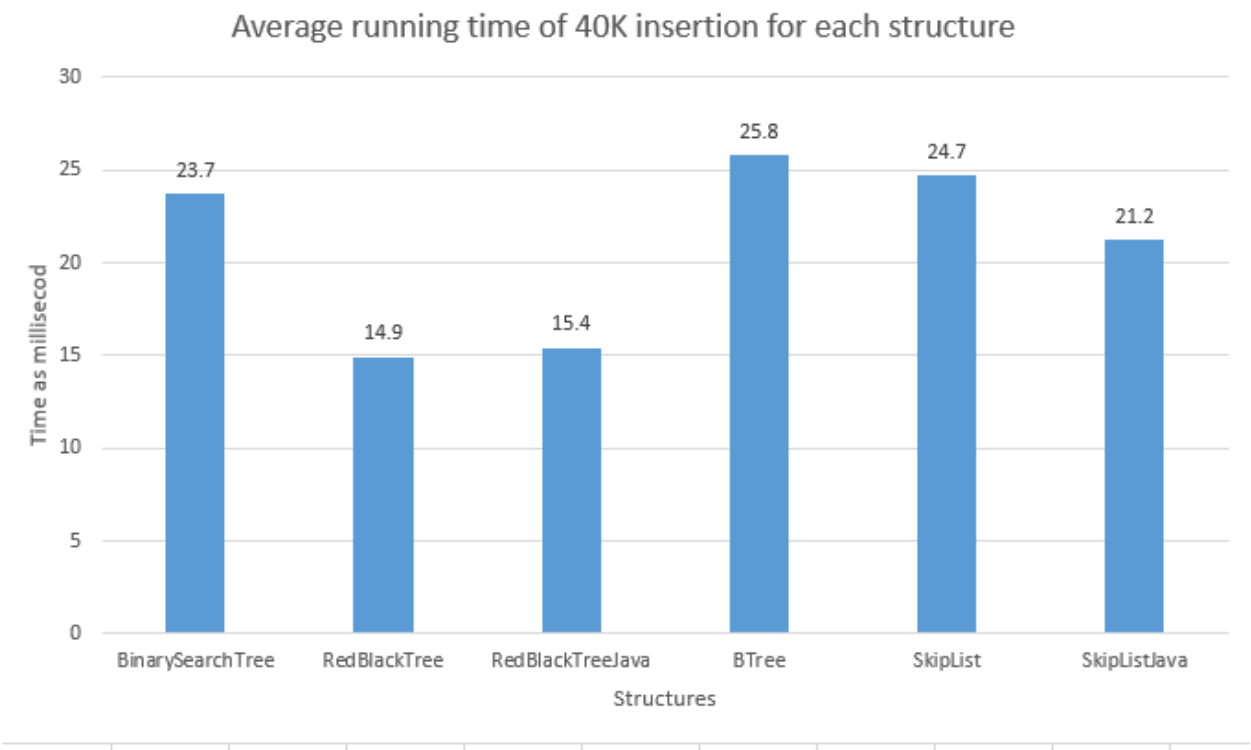
SkipListJava 7. 40K insertion: 21 ms.

SkipListJava 8. 40K insertion: 19 ms.

SkipListJava 9. 40K insertion: 18 ms.

SkipListJava 10. 40K insertion: 20 ms.

Average running time for inserting subsequently: 21.2 ms



*Analyzing 80K of each structure :

*BinarySearchTree :

BinarySearchTree 1. 80K insertion: 61 ms.

BinarySearchTree 2. 80K insertion: 127 ms.

BinarySearchTree 3. 80K insertion: 36 ms.

BinarySearchTree 4. 80K insertion: 33 ms.

BinarySearchTree 5. 80K insertion: 29 ms.

BinarySearchTree 6. 80K insertion: 32 ms.

BinarySearchTree 7. 80K insertion: 32 ms.

BinarySearchTree 8. 80K insertion: 33 ms.

BinarySearchTree 9. 80K insertion: 35 ms.

BinarySearchTree 10. 80K insertion: 28 ms.

Average running time for inserting subsequently: 44.6 ms

*RedBlackTree(book) :

RedBlackTree 1. 80K insertion: 61 ms.

RedBlackTree 2. 80K insertion: 43 ms.

RedBlackTree 3. 80K insertion: 66 ms.

RedBlackTree 4. 80K insertion: 37 ms.

RedBlackTree 5. 80K insertion: 33 ms.

RedBlackTree 6. 80K insertion: 38 ms.

RedBlackTree 7. 80K insertion: 35 ms.

RedBlackTree 8. 80K insertion: 41 ms.

RedBlackTree 9. 80K insertion: 44 ms.

RedBlackTree 10. 80K insertion: 30 ms.

Average running time for inserting subsequently: 42.8 ms

***RedBlackTree(java) :**

RedBlackTreeJava 1. 80K insertion: 23 ms.

RedBlackTreeJava 2. 80K insertion: 26 ms.

RedBlackTreeJava 3. 80K insertion: 29 ms.

RedBlackTreeJava 4. 80K insertion: 27 ms.

RedBlackTreeJava 5. 80K insertion: 31 ms.

RedBlackTreeJava 6. 80K insertion: 25 ms.

RedBlackTreeJava 7. 80K insertion: 25 ms.

RedBlackTreeJava 8. 80K insertion: 24 ms.

RedBlackTreeJava 9. 80K insertion: 20 ms.

RedBlackTreeJava 10. 80K insertion: 19 ms.

Average running time for inserting subsequently: 24.9 ms

***BTree(book) :**

BTree 1. 80K insertion: 44 ms.

BTree 2. 80K insertion: 44 ms.

BTree 3. 80K insertion: 50 ms.

BTree 4. 80K insertion: 43 ms.

BTree 5. 80K insertion: 47 ms.

BTree 6. 80K insertion: 35 ms.

BTree 7. 80K insertion: 35 ms.

BTree 8. 80K insertion: 34 ms.

BTree 9. 80K insertion: 40 ms.

BTree 10. 80K insertion: 38 ms.

Average running time for inserting subsequently: 41 ms

***SkipListBook (book) :**

SkipListBook 1. 80K insertion: 56 ms.

SkipListBook 2. 80K insertion: 92 ms.

SkipListBook 3. 80K insertion: 75 ms.

SkipListBook 4. 80K insertion: 64 ms.

SkipListBook 5. 80K insertion: 51 ms.

SkipListBook 6. 80K insertion: 55 ms.

SkipListBook 7. 80K insertion: 53 ms.

SkipListBook 8. 80K insertion: 50 ms.

SkipListBook 9. 80K insertion: 49 ms.

SkipListBook 10. 80K insertion: 55 ms.

Average running time for inserting subsequently: 60 ms

***SkipListJava (java) :**

SkipListJava 1. 80K insertion: 45 ms.

SkipListJava 2. 80K insertion: 52 ms.

SkipListJava 3. 80K insertion: 52 ms.

SkipListJava 4. 80K insertion: 49 ms.

SkipListJava 5. 80K insertion: 44 ms.

SkipListJava 6. 80K insertion: 38 ms.

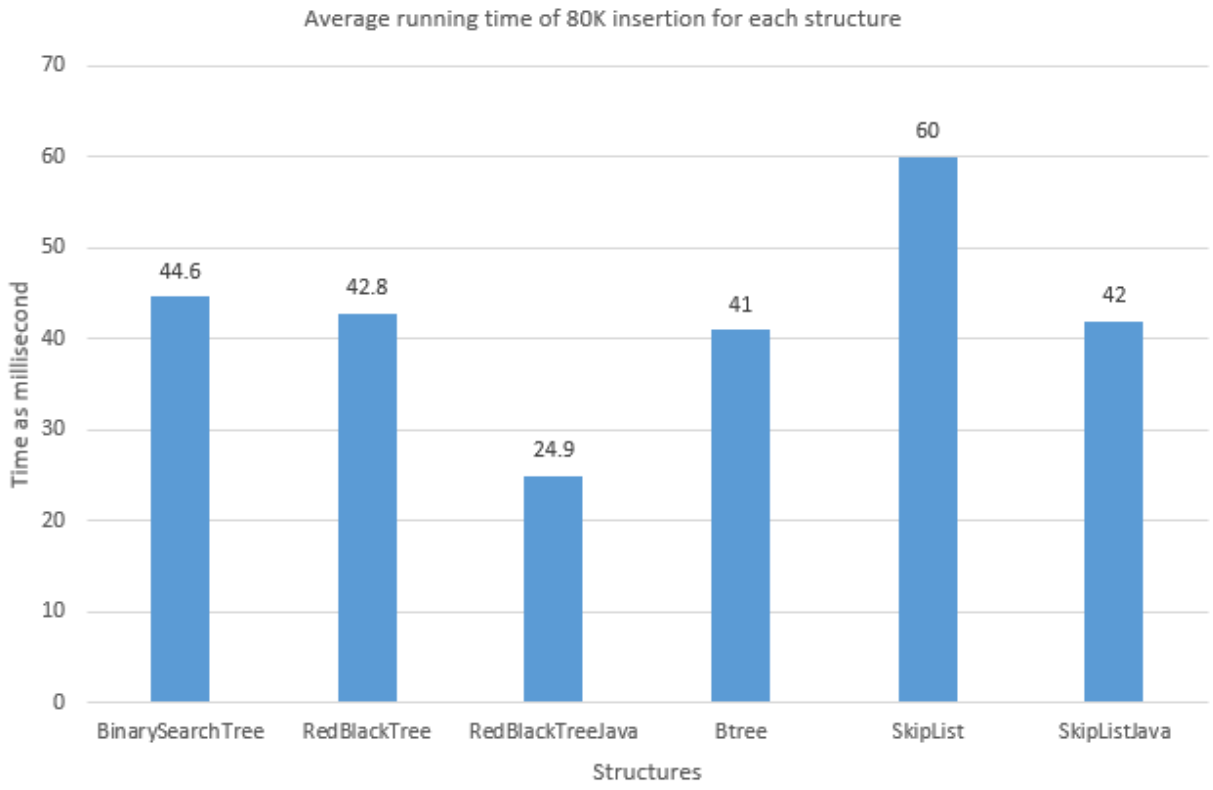
SkipListJava 7. 80K insertion: 38 ms.

SkipListJava 8. 80K insertion: 34 ms.

SkipListJava 9. 80K insertion: 34 ms.

SkipListJava 10. 80K insertion: 34 ms.

Average running time for inserting subsequently: 42 ms



*Non-existing item insertion running time for each structure :

***BinarySearchTree :**

10 extra random non-existing number insertion for 10K insertion : 48000 ns.

10 extra random non-existing number insertion for 20K insertion : 186000 ns.

10 extra random non-existing number insertion for 40K insertion : 66000 ns.

10 extra random non-existing number insertion for 80K insertion : 67000 ns.

Average running time for inserting subsequently: 91750 ns

***RedBlackTree(book) :**

10 extra random non-existing number insertion for 10K insertion : 44000 ns.

10 extra random non-existing number insertion for 20K insertion : 25000 ns.

10 extra random non-existing number insertion for 40K insertion : 33000 ns.

10 extra random non-existing number insertion for 80K insertion : 28000 ns.

Average running time for inserting subsequently: 32500 ns

***RedBlackTree(java) :**

10 extra random non-existing number insertion for 10K insertion : 52000 ns.

10 extra random non-existing number insertion for 20K insertion : 28000 ns.

10 extra random non-existing number insertion for 40K insertion : 25000 ns.

10 extra random non-existing number insertion for 80K insertion : 23000 ns.

Average running time for inserting subsequently: 32000 ns

***BTree(book) :**

10 extra random non-existing number insertion for 10K insertion : 45000 ns.

10 extra random non-existing number insertion for 20K insertion : 38000 ns.

10 extra random non-existing number insertion for 40K insertion : 41000 ns.

10 extra random non-existing number insertion for 80K insertion : 32000 ns.

Average running time for inserting subsequently: 39000 ns

***SkipListBook (book) :**

10 extra random non-existing number insertion for 10K insertion : 44000 ns.

10 extra random non-existing number insertion for 20K insertion : 26000 ns.

10 extra random non-existing number insertion for 40K insertion : 25000 ns.

10 extra random non-existing number insertion for 80K insertion : 39000 ns.

Average running time for inserting subsequently: 26187 ns

***SkipListJava (java) :**

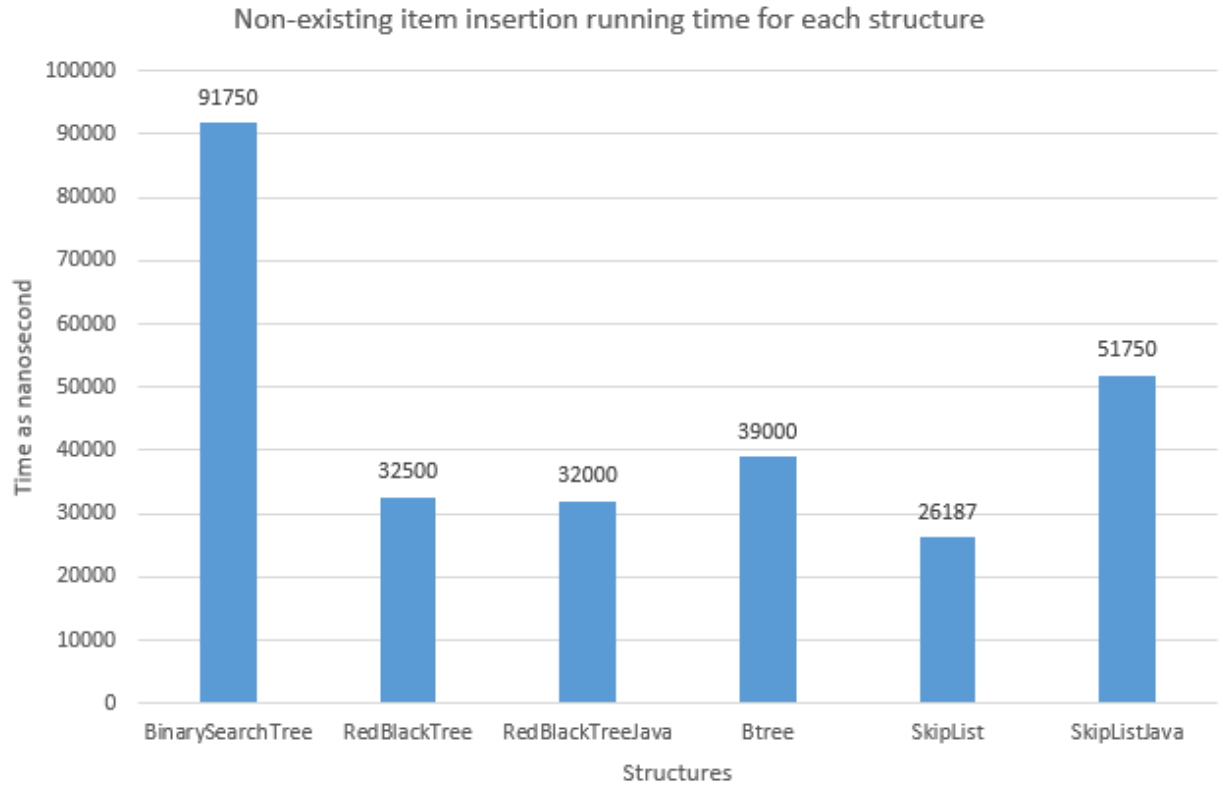
10 extra random non-existing number insertion for 10K insertion : 68000 ns.

10 extra random non-existing number insertion for 20K insertion : 48000 ns.

10 extra random non-existing number insertion for 40K insertion : 42000 ns.

10 extra random non-existing number insertion for 80K insertion : 49000 ns.

Average running time for inserting subsequently: 51750 ns



*Existing item deletion running time for each structure :

***BinarySearchTree :**

10 random existing number deletion from 10K size: 652 ms.

10 random existing number deletion from 20K size: 408 ms.

10 random existing number deletion from 40K size: 219 ms.

10 random existing number deletion from 80K size: 142 ms.

Average running time for inserting subsequently: 355.25 ms

***RedBlackTree(book) :**

10 random existing number deletion from 10K size: 463 ms.

10 random existing number deletion from 20K size: 416 ms.

10 random existing number deletion from 40K size: 220 ms.

10 random existing number deletion from 80K size: 121 ms.

Average running time for inserting subsequently: 305 ms

***RedBlackTree(java) :**

10 random existing number deletion from 10K size: 774 ms.

10 random existing number deletion from 20K size: 192 ms.

10 random existing number deletion from 40K size: 225 ms.

10 random existing number deletion from 80K size: 252 ms.

Average running time for inserting subsequently: 360 ms

***BTree(book) :**

*There is no implementation ☹

Average running time for inserting subsequently: ms

***SkipListBook (book) :**

10 random existing number deletion from 10K size: 749 ms.

10 random existing number deletion from 20K size: 704 ms.

10 random existing number deletion from 40K size: 232 ms.

10 random existing number deletion from 80K size: 361 ms.

Average running time for inserting subsequently: 511 .5 ms

***SkipListJava (java) :**

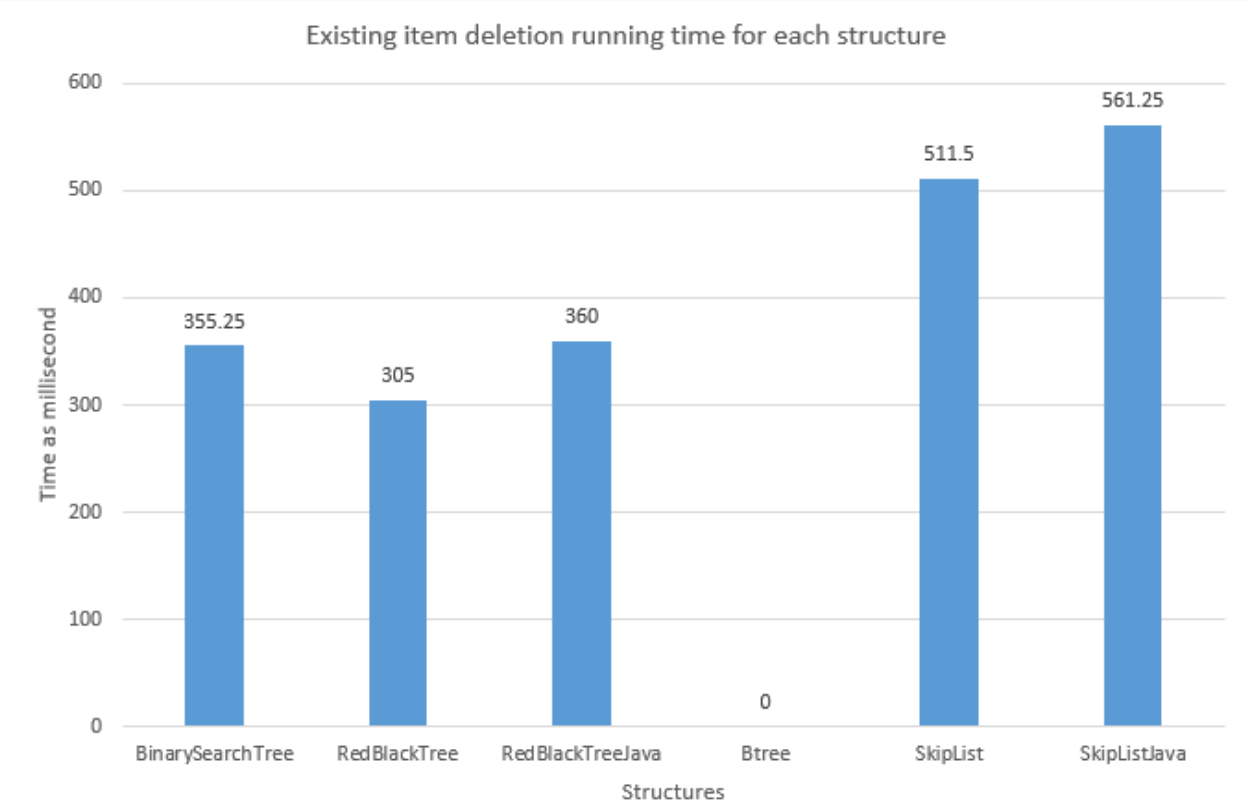
10 random existing number deletion from 10K size: 1119 ms.

10 random existing number deletion from 20K size: 712 ms.

10 random existing number deletion from 40K size: 191 ms.

10 random existing number deletion from 80K size: 223 ms.

Average running time for inserting subsequently: 561.25 ms



Result :

We can say that each structure has unique advantages.

*These results depend on a lot of variable . We cannot say this structure is better than this confidently. But when we come to concurrency and threading , **SkipList** has a list structure and allow us to concurrent programming thanks to its list structure. But the other structure such as RedBlackTree or BinarySearchTree do not allow concurrency. Because structure of building tree does now permit that.

*I observe that insertion and deletion time complexity of each structures are so close to each other. Because according the theoretical result , inserting and deleting takes **O(logn)** time for each in **average case**.

*Class Diagram:

*Attached to the assignment file as png.

*Running Command and Result:

*Attached to the assignment file as txt.