

**Question 2:**

**Problem Solution:**

* I need implement Skip-List data structure from my book first.
* It’s remove method is missing so I searched online and found remove method for it.
* Then, our task is design a data structure so each node can keep several elements instead of one element.
* To do this, I changed SLNode<E> data field to an ArrayList (I could use a default java array but I guess it needs to be shifted after add and remove methods so I thought ArrayList is a better solution)
* In constructor of MySkipList an integer is given as parameter. This integer decides how many elements will be in a single node at maximum. And any node cannot hold less than this integer’s half size except first node of the list.
* Then, I need to change add and remove methods.
* In normal skip list in the book, it first finds the predecessors of the node that it will be added and makes assignments so it can be added easily to anywhere in the list. Firstly, I found predecessors of that element and the element to node of predecessors first link. If it passes its capacity, I need to divide this node. (Example: Order is 5 -> If a node reaches capacity, It will hold 3 and 2 elements in two nodes.)
* Remove method pretty much the same. Just checks any node can not hold lesser than order/2, it will be removed and elements will be added to node’s predecessors.

**Test Cases:**

|  |  |  |
| --- | --- | --- |
| **Test** **Subject** | **Test Number** | **Pass/Fail** |
| Create a test object with order 5 | T1 | PASS |
| Add first element | T2 | PASS |
| Add more elements | T3 | PASS |
| Divide first node | T4 | PASS |
| Add more elements | T5 | PASS |
| Remove nodes | T6 | PASS |
| Remove elements and node | T7 | PASS |
| Test it again with order 4 | T8 | PASS |

**Running and Results:**

Test T1:

Test Data:

MySkipList<Integer> test = new MySkipList<>(5);

Excepted: test: “Empty”

Result: Passed

Test T2:

Test Data:

test.add(-5);

Excepted:

Head 1->-5 --> {-5}1: |->null|

Result:Passed

Test T3:

Test Data:

test.add(5);  
test.add(10);

Excepted:

Head 1->-5 --> {-5, 0, 5, 10}1: |->null|

Result: Passed

Test T4:

Test Data:

test.add(3);

Excepted:

Head 2->-5->null --> {-5, 0, 3}1: |->5| --> {5, 10}1: |->null|

Result: Passed

Test T5:

Test Data:

test.add(20);  
test.add(-10);  
test.add(9);  
test.add(7);  
test.add(17);  
test.add(25);

Excepted:

Head 2->-10->15 --> {-10, -5, 0, 3}1: |->5| --> {5, 7, 9, 10}1: |->15| --> {15, 17, 20, 25}2: |->null->null|

Result: Passed

Test T6:

Test Data:

test.remove(5);  
test.remove(10);

Excepted:

Head 2->-10->15 --> {-10, -5, 0, 3}1: |->7| --> {7, 9}1: |->15| --> {15, 17, 20, 25}2: |->null->null|

Result: Passed

Test T7:

Test Data:

test.remove(7);

Excepted:

Head 2->-10->3 --> {-10, -5, 0}1: |->3| --> {3, 9}1: |->15| --> {15, 17, 20, 25}2: |->null->null|

Result: Passed

Test T8:

Test Data:

MySkipList<Integer> test = new MySkipList<>(4);  
test.add(-5);  
test.add(0);  
test.add(5);  
test.add(10);  
test.add(3);  
test.add(15);  
test.add(20);  
test.add(-10);  
test.add(9);  
test.add(7);  
test.add(17);  
test.add(25);  
test.remove(5);  
test.remove(10);  
test.remove(7);

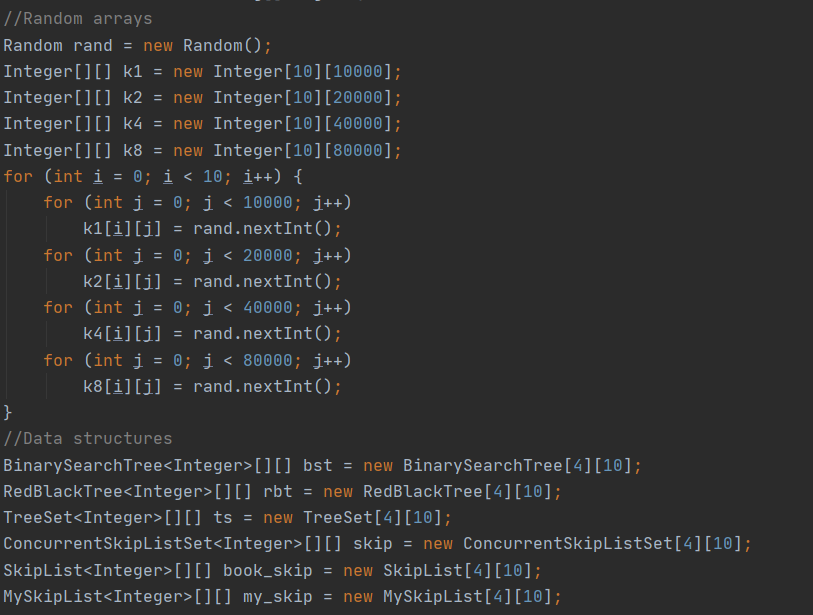
Excepted:

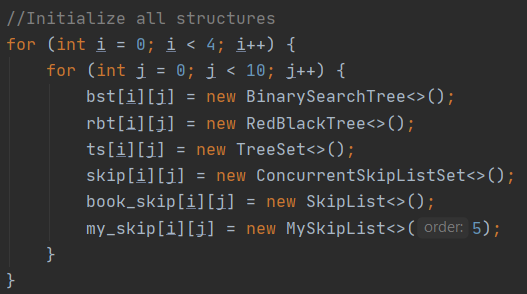
Head 3->-10->15->null --> {-10, -5}1: |->0| --> {0, 3, 9}1: |->15| --> {15, 17}2: |->20->null| --> {20, 25}1: |->null|

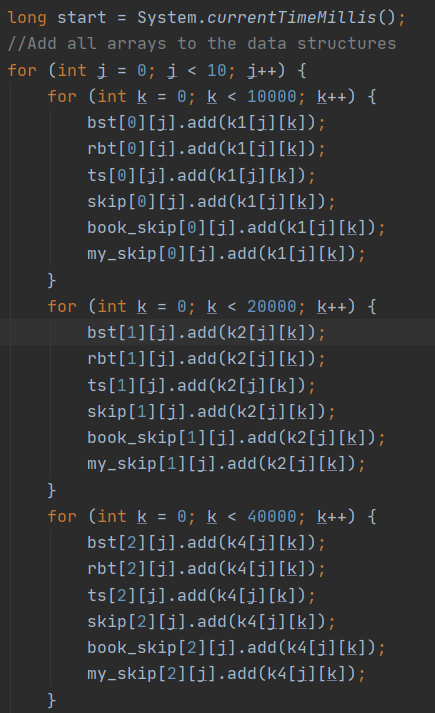
Result: Passed

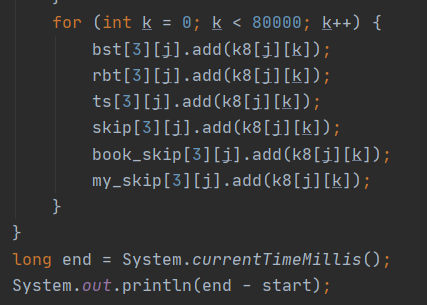
**Question 3:**

1- 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.



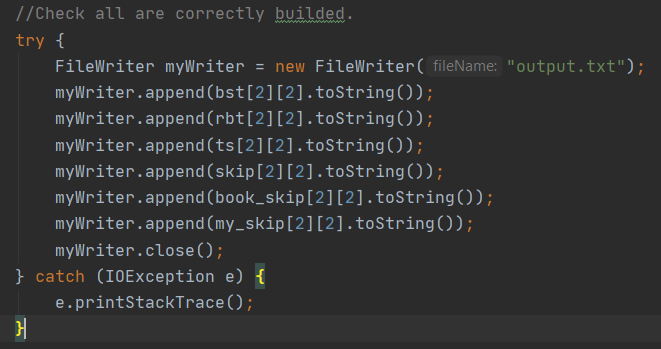






**Total Time: 7693ms**

2- Verify that data structures are built correctly (for example, for BST, perform an inorder traversal).



**Output.txt attached to zip file.**

3- 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.

**BST Insertion Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1-** | 19500 | 18600 | 20200 | 23400 |
| **2-** | 22700 | 17900 | 21700 | 35500 |
| **3-** | 17700 | 17800 | 16300 | 21400 |
| **4-** | 15200 | 17300 | 19700 | 19800 |
| **5-** | 15900 | 14700 | 19800 | 20200 |
| **6-** | 16100 | 19300 | 23500 | 25500 |
| **7-** | 15600 | 18600 | 21300 | 22500 |
| **8-** | 17700 | 17400 | 17400 | 21100 |
| **9-** | 12800 | 16600 | 22600 | 22400 |
| **10-** | 12500 | 19000 | 22800 | 30900 |
| **Average** | 16570 | 17720 | 20530 | 24.270 |

**Red-Black Tree from book Insertion Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 35200 | 23600 | 22000 | 39800 |
| **2 -** | 16900 | 17200 | 23700 | 28400 |
| **3 -** | 15800 | 15500 | 21000 | 27700 |
| **4 -** | 13400 | 13600 | 20900 | 25100 |
| **5 -** | 14100 | 16000 | 20100 | 30100 |
| **6 -** | 13400 | 13500 | 19000 | 29000 |
| **7 -** | 14000 | 14500 | 18900 | 27700 |
| **8 -** | 13500 | 20800 | 20400 | 27800 |
| **9 -** | 13800 | 15100 | 20900 | 27800 |
| **10 -** | 13000 | 13900 | 22000 | 2594500 |
| **Average** | 16310 | 16370 | 20890 | 285790 |

**Red-Black Tree Insertion in Java(nanosecond)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 17700 | 20000 | 28500 | 28600 |
| **2 -** | 18400 | 20100 | 21900 | 25500 |
| **3 -** | 17000 | 33700 | 18800 | 22600 |
| **4 -** | 15000 | 18400 | 23100 | 19300 |
| **5 -** | 17000 | 19000 | 17900 | 22500 |
| **6 -** | 15700 | 21000 | 20000 | 23100 |
| **7 -** | 23300 | 16900 | 20400 | 19300 |
| **8 -** | 16300 | 18700 | 21200 | 27700 |
| **9 -** | 14500 | 17900 | 17400 | 26500 |
| **10 -** | 13000 | 18200 | 21800 | 37200 |
| **Average** | 16790 | 20390 | 21100 | 25230 |

**Skip-List in Java Insertion(nanosecond)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 27000 | 36500 | 35600 | 42600 |
| **2 -** | 26600 | 41300 | 43400 | 41700 |
| **3 -** | 25900 | 34200 | 35700 | 38300 |
| **4 -** | 25800 | 44500 | 28100 | 30500 |
| **5 -** | 25500 | 34400 | 31000 | 36800 |
| **6 -** | 24100 | 33700 | 33700 | 34300 |
| **7 -** | 22900 | 31600 | 30100 | 34500 |
| **8 -** | 21800 | 24700 | 38900 | 37700 |
| **9 -** | 22100 | 28300 | 32600 | 90300 |
| **10 -** | 21300 | 26800 | 60500 | 32200 |
| **Average** | 24300 | 33600 | 36960 | 41890 |

**Skip-List from my book Insertion(nanosecond)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 24800 | 52600 | 34200 | 44500 |
| **2 -** | 27300 | 32100 | 49700 | 53000 |
| **3 -** | 24000 | 40200 | 28200 | 39800 |
| **4 -** | 25900 | 33200 | 27000 | 33800 |
| **5 -** | 26000 | 49300 | 27200 | 35200 |
| **6 -** | 23300 | 31800 | 36300 | 63200 |
| **7 -** | 19900 | 29200 | 24200 | 36800 |
| **8 -** | 20100 | 25300 | 33200 | 32800 |
| **9 -** | 23800 | 36100 | 25800 | 30600 |
| **10 -** | 25300 | 26600 | 34800 | 52800 |
| **Average** | 24040 | 35640 | 32060 | 42250 |

**Skip-List from Question 2(nanosecond)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 36000 | 36000 | 48100 | 69100 |
| **2 -** | 28600 | 32800 | 51500 | 68300 |
| **3 -** | 31300 | 37800 | 39600 | 56200 |
| **4 -** | 30200 | 41400 | 38500 | 44400 |
| **5 -** | 33200 | 38100 | 42100 | 52500 |
| **6 -** | 25800 | 38000 | 35200 | 50000 |
| **7 -** | 38100 | 35200 | 36700 | 39800 |
| **8 -** | 27800 | 32800 | 42000 | 46200 |
| **9 -** | 30900 | 33800 | 38400 | 37800 |
| **10 -** | 24300 | 31600 | 42100 | 55200 |
| **Average** | 30620 | 35750 | 41420 | 51950 |

4-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.

**BST Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 51700 | 51400 | 60100 | 62800 |
| **2 -** | 44000 | 47500 | 44200 | 66500 |
| **3 -** | 58000 | 44400 | 42100 | 45600 |
| **4 -** | 43800 | 43100 | 44500 | 51900 |
| **5 -** | 43800 | 56400 | 48400 | 54400 |
| **6 -** | 38800 | 47600 | 43800 | 58300 |
| **7 -** | 42500 | 46100 | 46100 | 71700 |
| **8 -** | 37300 | 46700 | 40300 | 57900 |
| **9 -** | 41300 | 47200 | 48500 | 60900 |
| **10 -** | 41000 | 42100 | 42000 | 54100 |
| **Average** | 44220 | 47250 | 46000 | 58410 |

**Red-Black Tree from my book Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 66600 | 56400 | 67700 | 61800 |
| **2 -** | 44500 | 54500 | 55700 | 55800 |
| **3 -** | 43900 | 46700 | 49600 | 51600 |
| **4 -** | 50000 | 50100 | 50000 | 52400 |
| **5 -** | 42500 | 46900 | 48800 | 53600 |
| **6 -** | 40600 | 49300 | 50300 | 59100 |
| **7 -** | 55900 | 47700 | 51100 | 56800 |
| **8 -** | 42400 | 47500 | 47100 | 60300 |
| **9 -** | 44100 | 48400 | 50300 | 54900 |
| **10 -** | 55600 | 49700 | 46600 | 56200 |
| **Average** | 48610 | 49720 | 51720 | 56250 |

**Red-Black Tree in Java Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 69000 | 46500 | 47300 | 61500 |
| **2 -** | 37300 | 46500 | 49200 | 59200 |
| **3 -** | 37200 | 46400 | 45900 | 52500 |
| **4 -** | 36300 | 41800 | 45500 | 42400 |
| **5 -** | 38700 | 38600 | 46900 | 42700 |
| **6 -** | 36100 | 39800 | 53300 | 51500 |
| **7 -** | 34100 | 38500 | 59700 | 51000 |
| **8 -** | 37000 | 36400 | 85900 | 54700 |
| **9 -** | 34600 | 37400 | 51700 | 58800 |
| **10 -** | 33700 | 55400 | 75200 | 62500 |
| **Average** | 39400 | 42730 | 56060 | 53680 |

**Skip-List in Java Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 292800 | 122800 | 152000 | 138800 |
| **2 -** | 111700 | 133600 | 122400 | 145100 |
| **3 -** | 112600 | 138000 | 119800 | 144100 |
| **4 -** | 102600 | 123400 | 111400 | 110300 |
| **5 -** | 94400 | 112400 | 115600 | 163000 |
| **6 -** | 105700 | 119200 | 115400 | 168900 |
| **7 -** | 116700 | 130900 | 107800 | 167200 |
| **8 -** | 105200 | 119400 | 103600 | 166400 |
| **9 -** | 99000 | 122000 | 116700 | 126000 |
| **10 -** | 97900 | 120100 | 110000 | 117300 |
| **Average** | 123860 | 124180 | 117470 | 144710 |

**Skip-List from my Book Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 45100 | 49800 | 40900 | 49500 |
| **2 -** | 35300 | 36600 | 37700 | 40000 |
| **3 -** | 37800 | 31000 | 30500 | 43200 |
| **4 -** | 46200 | 28500 | 30600 | 32700 |
| **5 -** | 35800 | 26200 | 55900 | 40300 |
| **6 -** | 30400 | 31300 | 34600 | 37700 |
| **7 -** | 30800 | 32000 | 28100 | 32600 |
| **8 -** | 33500 | 27600 | 34200 | 31400 |
| **9 -** | 36200 | 29200 | 29200 | 41900 |
| **10 -** | 35000 | 27800 | 26900 | 29900 |
| **Average** | 36610 | 32000 | 34860 | 37920 |

**Skip-List in Q2 Remove Test(nanoseconds)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N = 10000** | **N = 20000** | **N = 40000** | **N = 80000** |
| **1 -** | 139600 | 118400 | 146100 | 125100 |
| **2 -** | 115000 | 170000 | 131800 | 129300 |
| **3 -** | 92800 | 141300 | 141700 | 118600 |
| **4 -** | 84600 | 128600 | 149000 | 113700 |
| **5 -** | 93200 | 133500 | 140300 | 99800 |
| **6 -** | 96700 | 145000 | 3647300 | 126200 |
| **7 -** | 100900 | 170000 | 126800 | 112900 |
| **8 -** | 94900 | 109000 | 112700 | 116400 |
| **9 -** | 90300 | 117600 | 116100 | 128700 |
| **10 -** | 86600 | 117300 | 108000 | 116800 |
| **Average** | 99460 | 135070 | 481980 | 118750 |

**Question 4**:

Problem Solution:

* I need to implement a menu-driven program for managing a software store.
* System will have two types of user:
* Administrators and users
* Administrators enters the system with a password, to be able to add, delete, update information.
* Users who browse software.
* To able to store, I should use a search tree interface for the table.
* Add, get, set methods use table’s add, get, set methods
* Two types of users will be nested class in the system.
* There is method to instantiate admin and user classes.
* All methods will be in these two classes.
* There is a toString method override to print all table

Test Cases:

|  |  |  |
| --- | --- | --- |
| Test Subject | Test Number | Pass/Fail |
| Create Admin | T1 | Pass |
| Adds software | T2 | Pass |
| Prints store | T3 | Pass |
| Create User | T4 | Pass |
| Search By Name | T5 | Pass |
| Buy | T6 | Pass |
| Admin delete | T7 | Pass |
| Admin update | T8 | Pass |

Running and Results:

Test T1:

Test Data:

SoftwareStore st = new SoftwareStore(new RedBlackTree<>());  
SoftwareStore.Admin admin = st.createAdmin("password");

Excepted: No Error

Result: Passed

Test T2:

Test Data:

admin.add("Adobe Flash 4.0","FREE");  
admin.add("Windows 10", "10$");  
admin.add("Adobe Photoshop 6.0", "120$");

Excepted: No errors

Result: Passed

Test T3:

Test Data:

System.*out*.println(st.toString());

Excepted:

Black: Software: Adobe Photoshop 6.2, Price: 40$, Quantity: 1

Black: Software: Adobe Flash 4.0, Price: FREE, Quantity: 2

Red: Software: Adobe Flash 3.3, Price: FREE, Quantity: 1

null

null

Red: Software: Adobe Photoshop 6.0, Price: 120$, Quantity: 2

null

null

Black: Software: Norton 5.5, Price: 20₺, Quantity: 1

Red: Software: Norton 4.5, Price: 30$, Quantity: 1

null

null

Red: Software: Windows 10, Price: 10$, Quantity: 1

null

null

Result: Passed

Test T4:

Test Data:

SoftwareStore.User user = st.createUser();

Excepted: No errors

Result: Passed

Test T5:

Test Data:

user.searchByName("Windows 10");

Excepted:

Software: Windows 10, Price: 10$, Quantity: 1

Result: Passed

Test T6:

Test Data:

user.buy("Windows 10", "10$");  
user.buy("Norton 5.5", "20₺");

Excepted: No errors

Result: Passed

Test T7:

Test Data:

admin.delete("Adobe Photoshop 6.0", "120$");

Excepted: No errors

Result: Passed

Test T8:

Test Data:

admin.update("Adobe Flash 4.0", "FREE" ,"1$");

Excepted: No errors

Result: Passed