GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 6 Report

Abdurrahman BULUT
1901042258

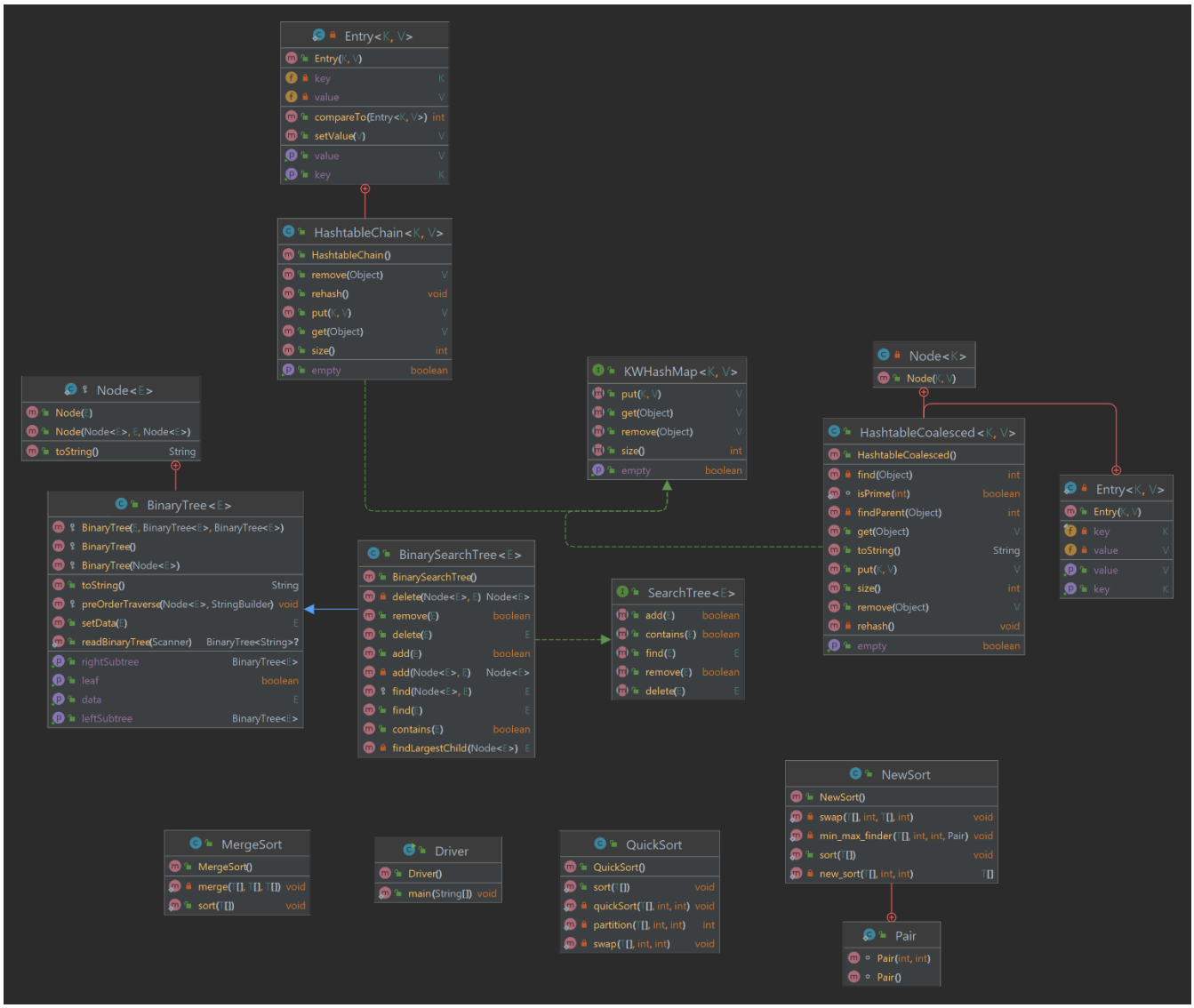
1. SYSTEM REQUIREMENTS

➢ Functional Requirements

♦ System

- openjdk 17.0.2 2022-01-18 LTS
- OpenJDK Runtime Environment Corretto-17.0.2.8.1 (build 17.0.2+8-LTS)
- OpenJDK 64-Bit Server VM Corretto-17.0.2.8.1 (build 17.0.2+8-LTS, mixed mode, sharing)

2. CLASS DIAGRAM



S

3. PROBLEM SOLUTION APPROACH

In this assignment, we are asked to implement several HashMap and some sorting algorithms and analyse them**.** In part Q1**,**

1) I implemented Hash table with chaining technique. I implemented a Binary Search Tree with Binary tree class and Search tree interface. In default implementation has included linked list and I used binary search tree for them. My chaining implementation is not working properly. I need a iterator for Binary Search Tree. Since I cannot implement an iterator for Binary Search Tree, Chaining implementation works but not properly.

2) **Double – coalesced combination**
   a. **Coalesced hashing**

Coalesced hashing is a collision avoidance technique when there is a fixed sized data. It is a combination of both Separate chaining and Open addressing. It uses the concept of Open Addressing (linear probing) to find first empty place for colliding element from the bottom of the hash table and the concept of Separate Chaining to link the colliding elements to each other through pointers.

The basic operations of Coalesced hashing are:

INSERT (key):

The insert Operation inserts the key according to the hash value of that key if that hash value in the table is empty otherwise the key is inserted in first empty place from the bottom of the hash table and the address of this empty place is mapped in NEXT field of the previous pointing node of the chain. (Explained in example below).

DELETE(Key):

The key if present is deleted. Also, if the node to be deleted contains the address of another node in hash table, then this address is mapped in the NEXT field of the node pointing to the node which is to be deleted

SEARCH (key):

Returns True if key is present, otherwise return False.

The best-case complexity of all these operations is O (1) and the worst-case complexity is O(n) where n is the total number of keys. It is better than separate chaining because it inserts the colliding element

in the memory of hash table only instead of creating a new linked list as in separate chaining.

b. double hashing

Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Advantages of Double hashing

The advantage of Double hashing is that it is one of the best forms of probing, producing a uniform distribution of records throughout a hash table. This technique does not yield any clusters. It is one of effective method for resolving collisions.

Double hashing can be done using:

(hash1(key) + i * hash2(key)) % TABLE_SIZE

Here hash1() and hash2() are hash functions and TABLE_SIZE is size of hash table.
(We repeat by increasing i when collision occurs)

First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : hash2(key) = PRIME – (key % PRIME) where PRIME is a prime smaller than the TABLE_SIZE.

Using the double hashing function, the probe sites for the colliding item are determined during an insertion operation in the hybrid approach. Furthermore, like with the coalesced hashing technique, the colliding items are linked to each other through pointers. To calculate probe positions, I used the hash function which is already given.

Finally, I tested with different data sets

2) Sorting algorithms

I implemented Merge Sort, Quicksort and new sort algorithms and compared with 1000 random arrays for each problem size. For each problem size and problem, I used same random array and calculated run time separately.

4. TEST CASES

hash Map Created....
put(5,"test1");
put(3,"test2");
put(15,"test3");
put(12,"test4");
put(2,"test5");
put(277,"test6");
put(53,"test7");
put(232,"test7");
put(1,"test7");
put(65,"test7");
put(1010,"test7");
put(34,"test6");
put(78,"test7");
put(35888,"test7");
Added some Entries...
remove (2)
put(2,"test5")
remove (25)
get (12)
get (3)
get (277)
get (22)
isEmpty()

Part 2-3

Inserting 5 elements into hashTables

Getting 2 elements which exist in hashTables

Getting one element which not exist in hashTables

Removing 2 element that exist in hashTables

Removing... (3, 12)

Inserting 100 elements into hashTables

Getting 10 elements which exist in hashTables

Getting one element which not exist in hashTables

Removing 10 element that exist in hashTables

Inserting 2000 elements into hashTables

Getting 100 elements which exist in hashTables

Getting one element which not exist in hashTables

Removing 500 element that exist in hashTables

Removing one element that doesn't exist in hashTables

Search Algorithms Tests

Integer array with size 100 is created.

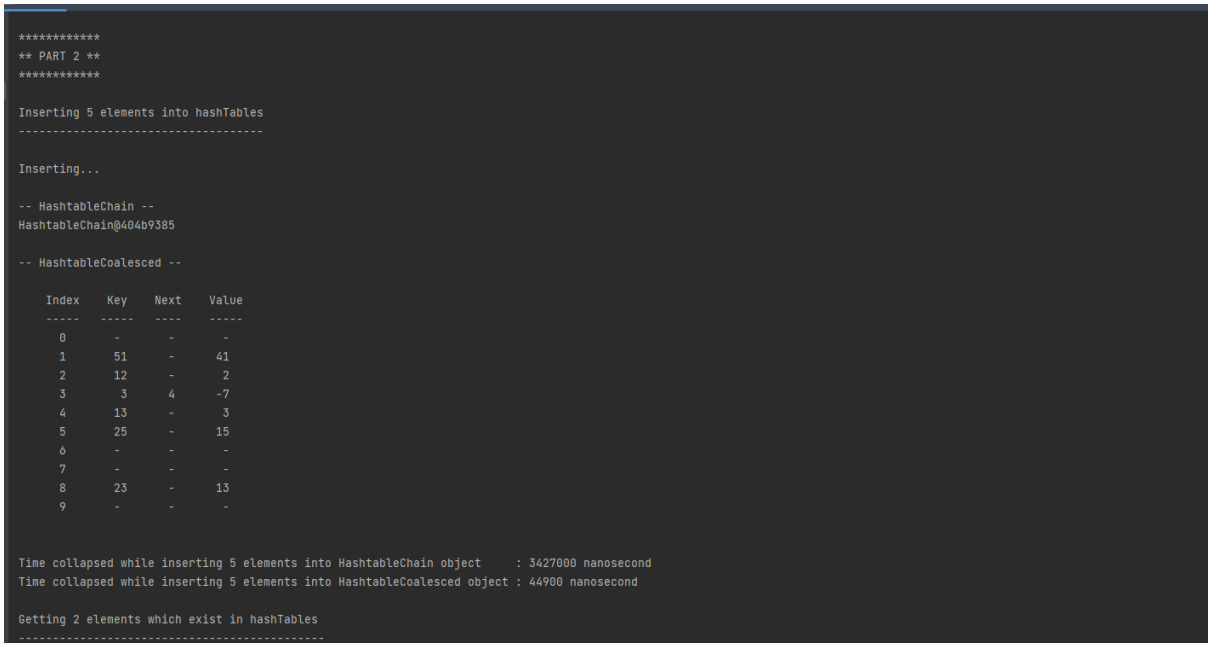
1000 arrays with size : 100 working
Average Run Time for size = 100
Quick Sort Time: 60504 ns
Merge Sort Time: 61704 ns
New Sort Time: 34297 ns

1000 arrays with size : 1000 working
Average Run Time for size = 1000
Quick Sort Time: 178772 ns
Merge Sort Time: 253635 ns
New Sort Time: 1168005 ns

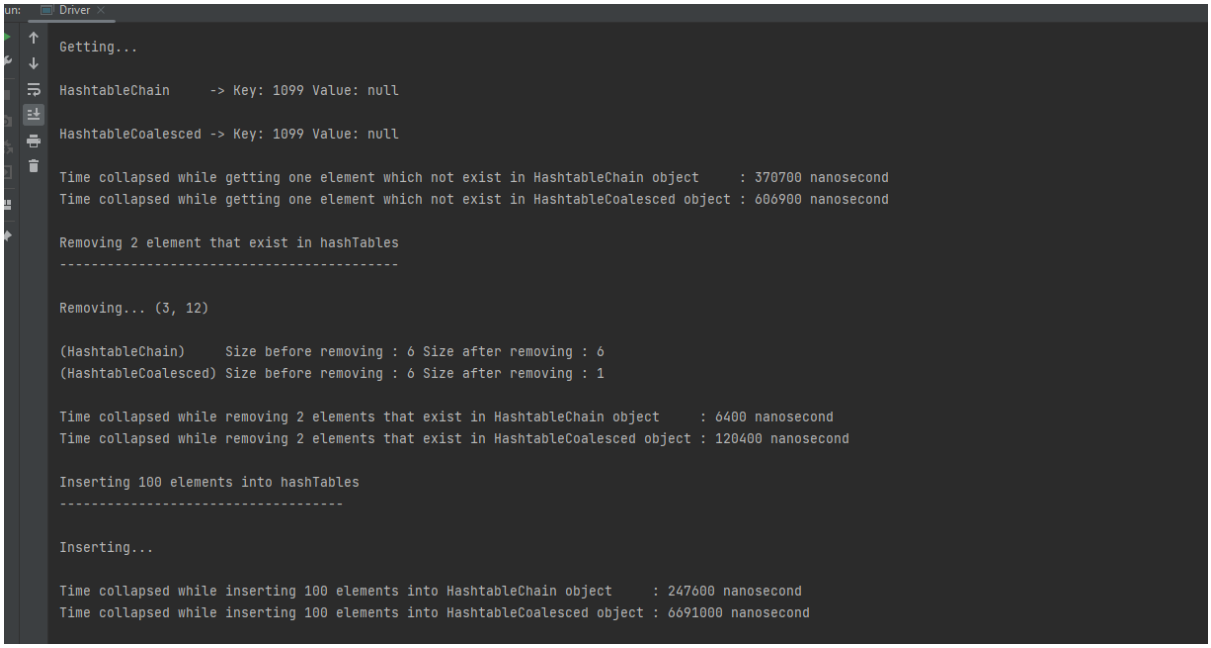
1000 arrays with size : 10000 working
Average Run Time for size = 10000
Quick Sort Time: 1325099 ns
Merge Sort Time: 1770771 ns
New Sort Time: 94736979 ns

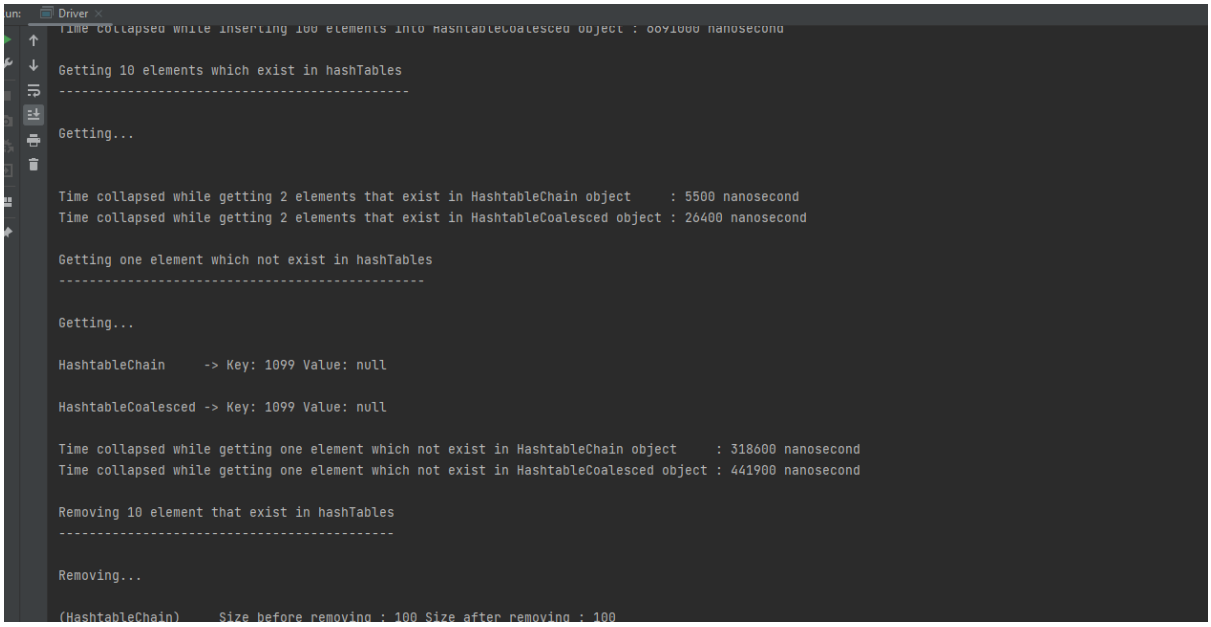
5. **RUNNING AND RESULTS**

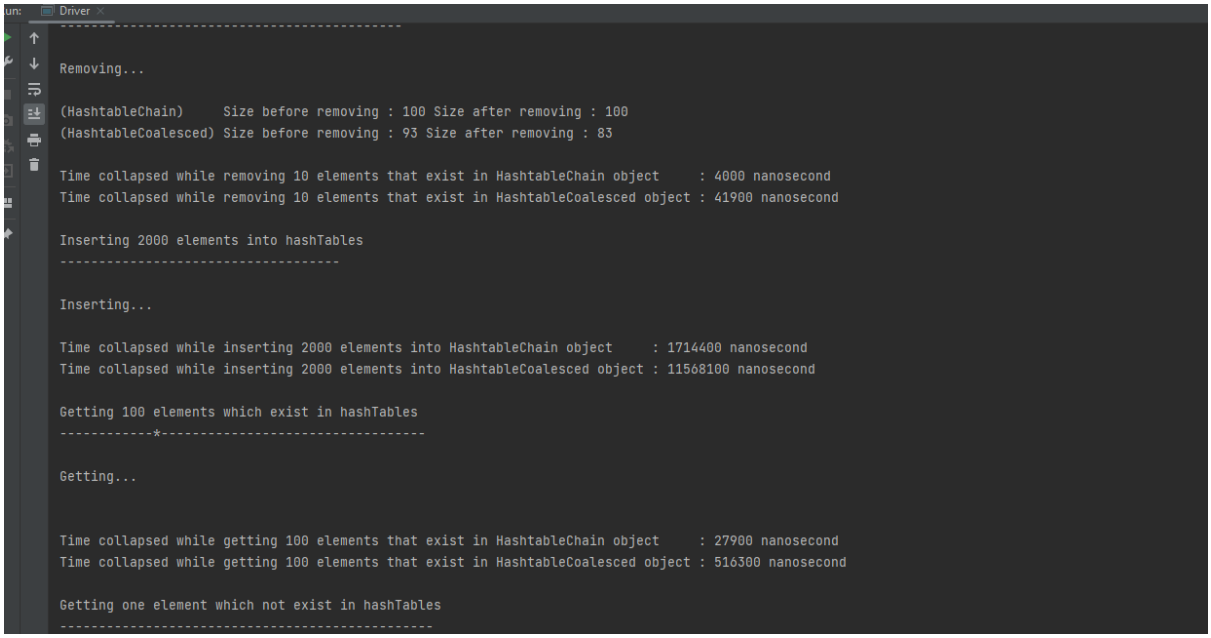
HashtableChain and HashtableCoalesced

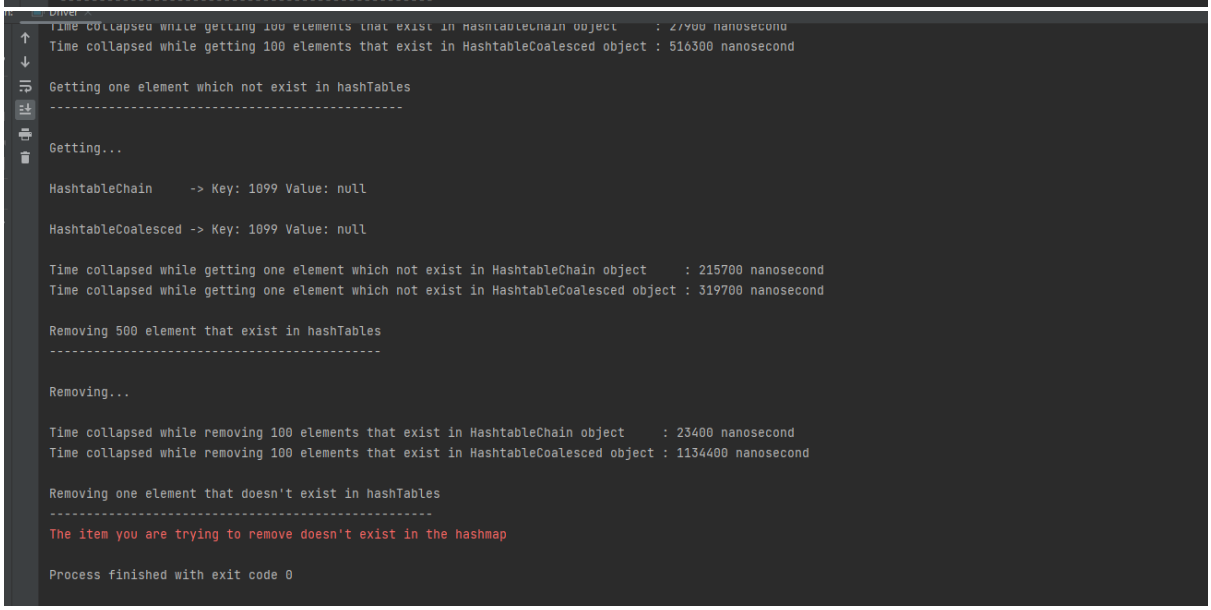


```
************
** PART 2 **
************

Inserting 5 elements into hashTables
------------------------------------

Inserting...

-- HashtableChain --
HashtableChain@404b9385

-- HashtableCoalesced --

    Index    Key    Next    Value
    -----   -----   ----    -----
      0       -       -       -
      1       51      -      41
      2       12      -       2
      3        3      4      -7
      4       13      -       3
      5       25      -      15
      6       -       -       -
      7       -       -       -
      8       23      -      13
      9       -       -       -


Time collapsed while inserting 5 elements into HashtableChain object     : 3427000 nanosecond
Time collapsed while inserting 5 elements into HashtableCoalesced object : 44900 nanosecond

Getting 2 elements which exist in hashTables
--------------------------------------------
```


Run:
Driver
Getting...
HashtableChain -> Key: 1099 Value: null
HashtableCoalesced -> Key: 1099 Value: null
Time collapsed while getting one element which not exist in HashtableChain object : 370700 nanosecond
Time collapsed while getting one element which not exist in HashtableCoalesced object : 606900 nanosecond
Removing 2 element that exist in hashTables
Removing... (3, 12)
(HashtableChain) Size before removing : 6 Size after removing : 6
(HashtableCoalesced) Size before removing : 6 Size after removing : 1
Time collapsed while removing 2 elements that exist in HashtableChain object : 6400 nanosecond
Time collapsed while removing 2 elements that exist in HashtableCoalesced object : 120400 nanosecond
Inserting 100 elements into hashTables
Inserting...
Time collapsed while inserting 100 elements into HashtableChain object : 247600 nanosecond
Time collapsed while inserting 100 elements into HashtableCoalesced object : 6691000 nanosecond


Run:
Driver
Getting 10 elements which exist in hashTables
Getting...
Time collapsed while getting 2 elements that exist in HashtableChain object : 5500 nanosecond
Time collapsed while getting 2 elements that exist in HashtableCoalesced object : 26400 nanosecond
Getting one element which not exist in hashTables
Getting...
HashtableChain -> Key: 1099 Value: null
HashtableCoalesced -> Key: 1099 Value: null
Time collapsed while getting one element which not exist in HashtableChain object : 318600 nanosecond
Time collapsed while getting one element which not exist in HashtableCoalesced object : 441900 nanosecond
Removing 10 element that exist in hashTables
Removing...
(HashtableChain) Size before removing : 100 Size after removing : 100



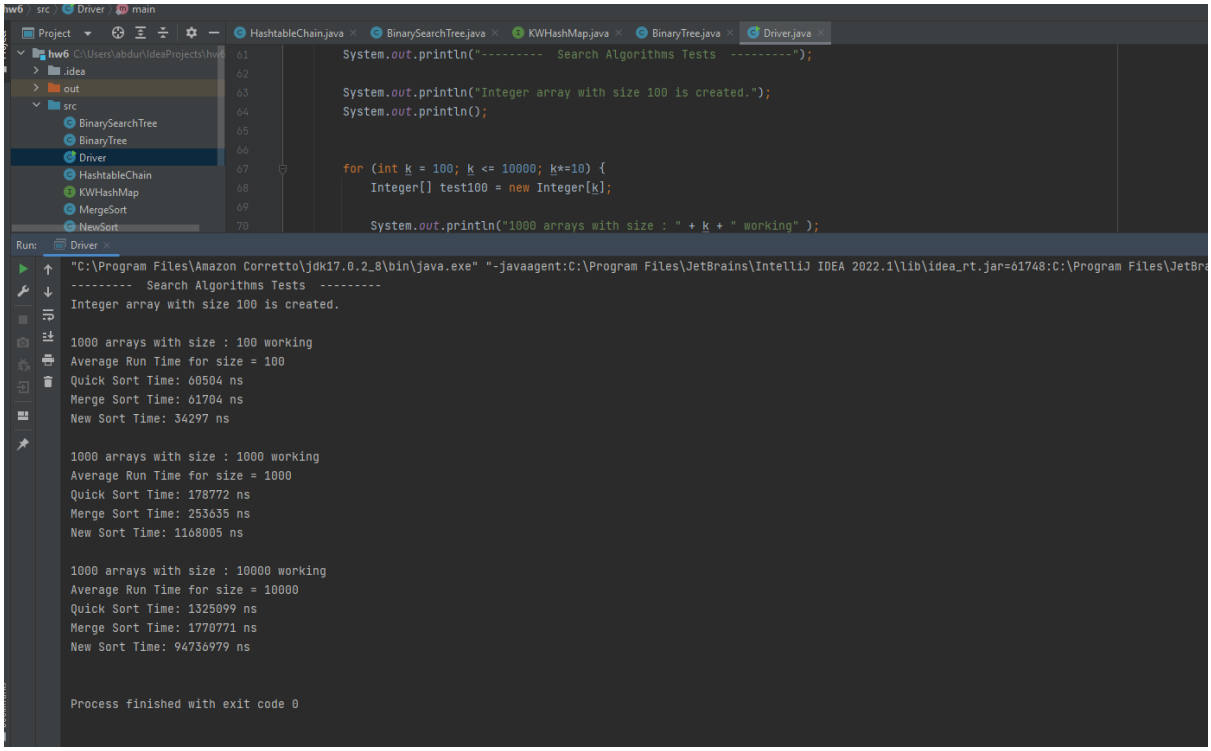


Search Algorithms:

Quick Sort has Ω(n log(n)) for best case and O(n^2) for the worst case. So, It has θ(n log(n)) for average.

Merge Sort has Ω(n log(n)) for best case and O(n log(n)) for the worst case. So, It has θ(n log(n)) for average.

For big arrays, our algorithm works slowly but for small arrays our array works faster than others.


hw6 / src / Driver / main
Project
HashtableChain.java
BinarySearchTree.java
KWHashMap.java
BinaryTree.java
Driver.java
hw6 C:\Users\abdur\IdeaProjects\hw6
.idea
out
src
BinarySearchTree
BinaryTree
Driver
HashtableChain
KWHashMap
MergeSort
NewSort
System.out.println("--------- Search Algorithms Tests ---------");
System.out.println("Integer array with size 100 is created.");
System.out.println();
for (int k = 100; k <= 10000; k*=10) {
Integer[] test100 = new Integer[k];
System.out.println("1000 arrays with size : " + k + " working");
Run:
Driver
"C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.1\lib\idea_rt.jar=61748:C:\Program Files\JetBra
--------- Search Algorithms Tests ---------
Integer array with size 100 is created.
1000 arrays with size : 100 working
Average Run Time for size = 100
Quick Sort Time: 60504 ns
Merge Sort Time: 61704 ns
New Sort Time: 34297 ns
1000 arrays with size : 1000 working
Average Run Time for size = 1000
Quick Sort Time: 178772 ns
Merge Sort Time: 253635 ns
New Sort Time: 1168005 ns
1000 arrays with size : 10000 working
Average Run Time for size = 10000
Quick Sort Time: 1325099 ns
Merge Sort Time: 1770771 ns
New Sort Time: 94736979 ns
Process finished with exit code 0