GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #6 Report

Serhat SARI
200104004028

1- SYSTEM REQUIREMENTS

a- Non-Functional System Requirements

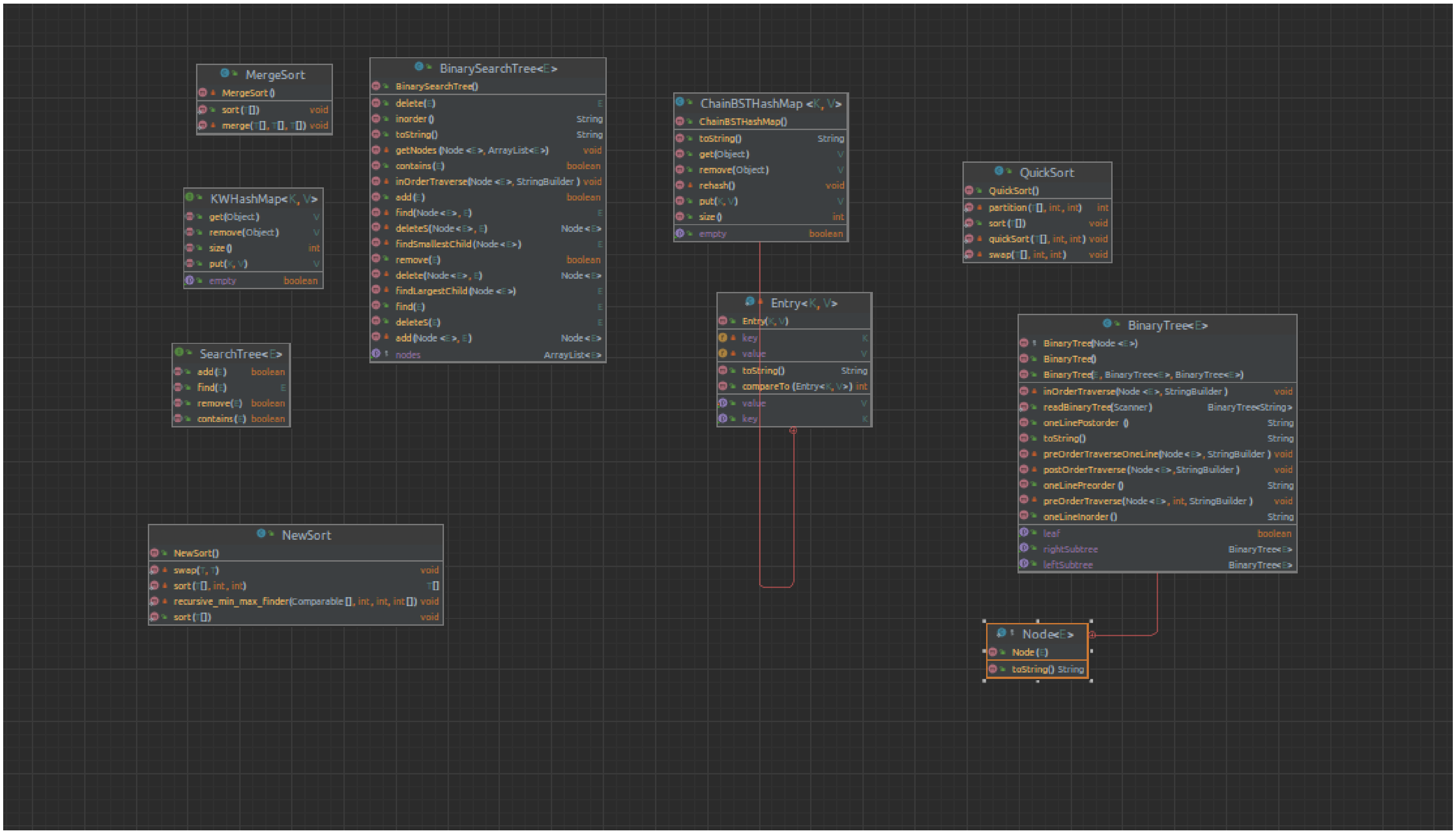
1- Back-end Software : Java 11

2- Software should be able to compile with “javac” on a linux distribution.

b-Functional System Requirements

-1.th Question, you should use Map as it supposed to be.

2- CLASS DIAGRAM

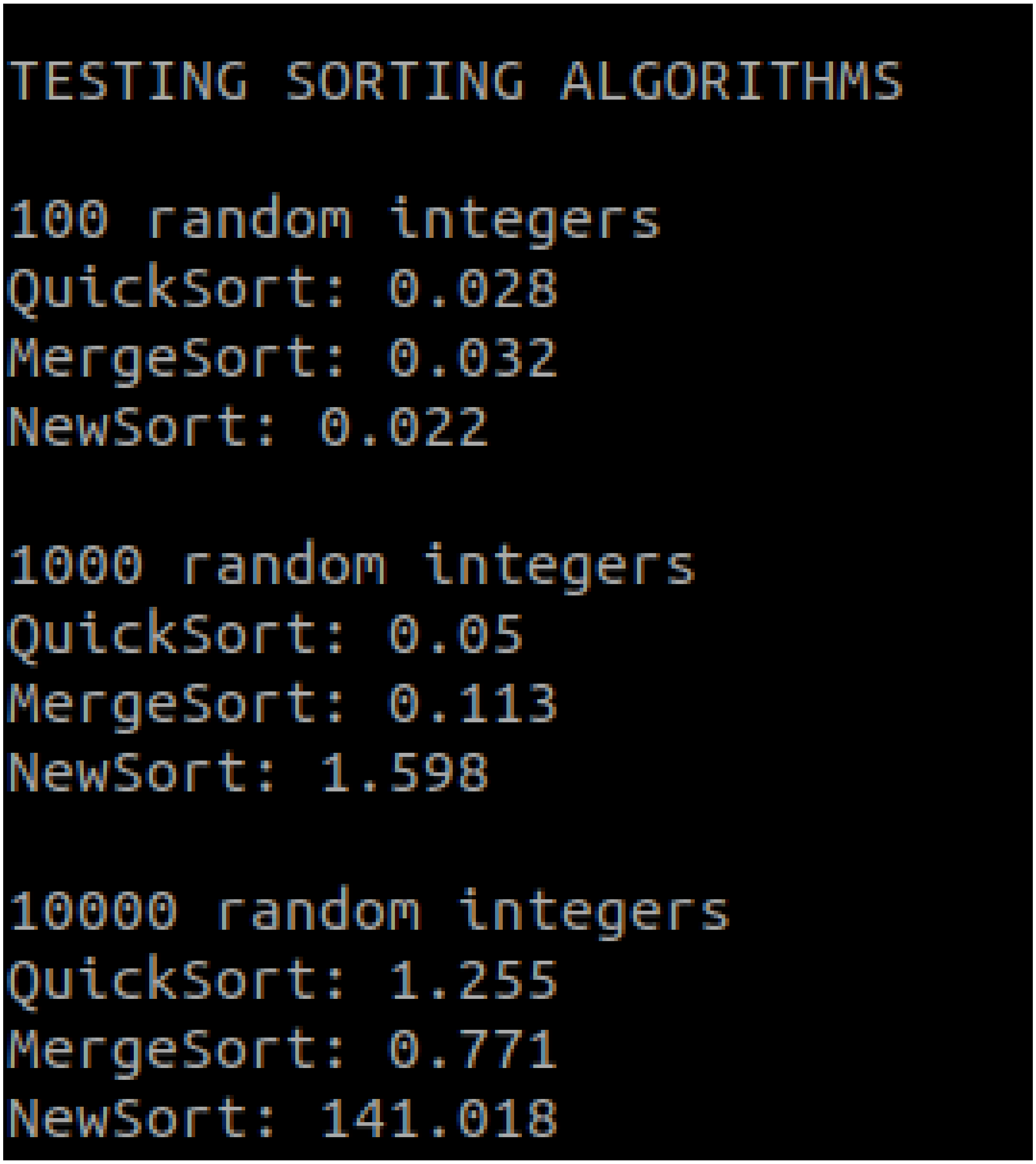


3- Problem Solution Approach

For the first problem, first i have read the book.
Then i started to implement the first one. Q1.1 was easy.
I used BST instead of LinkedList.

For the second problem, i took sorting algorithms from book.
Then i implement the new_sort class which was hard for me to implement.
Because recursive max min finder part was hard.
But eventually i did it.

SORTING ALGORITHM COMPARISON



```
TESTING SORTING ALGORITHMS

100 random integers
QuickSort: 0.028
MergeSort: 0.032
NewSort: 0.022

1000 random integers
QuickSort: 0.05
MergeSort: 0.113
NewSort: 1.598

10000 random integers
QuickSort: 1.255
MergeSort: 0.771
NewSort: 141.018
```

Q1. 2.a

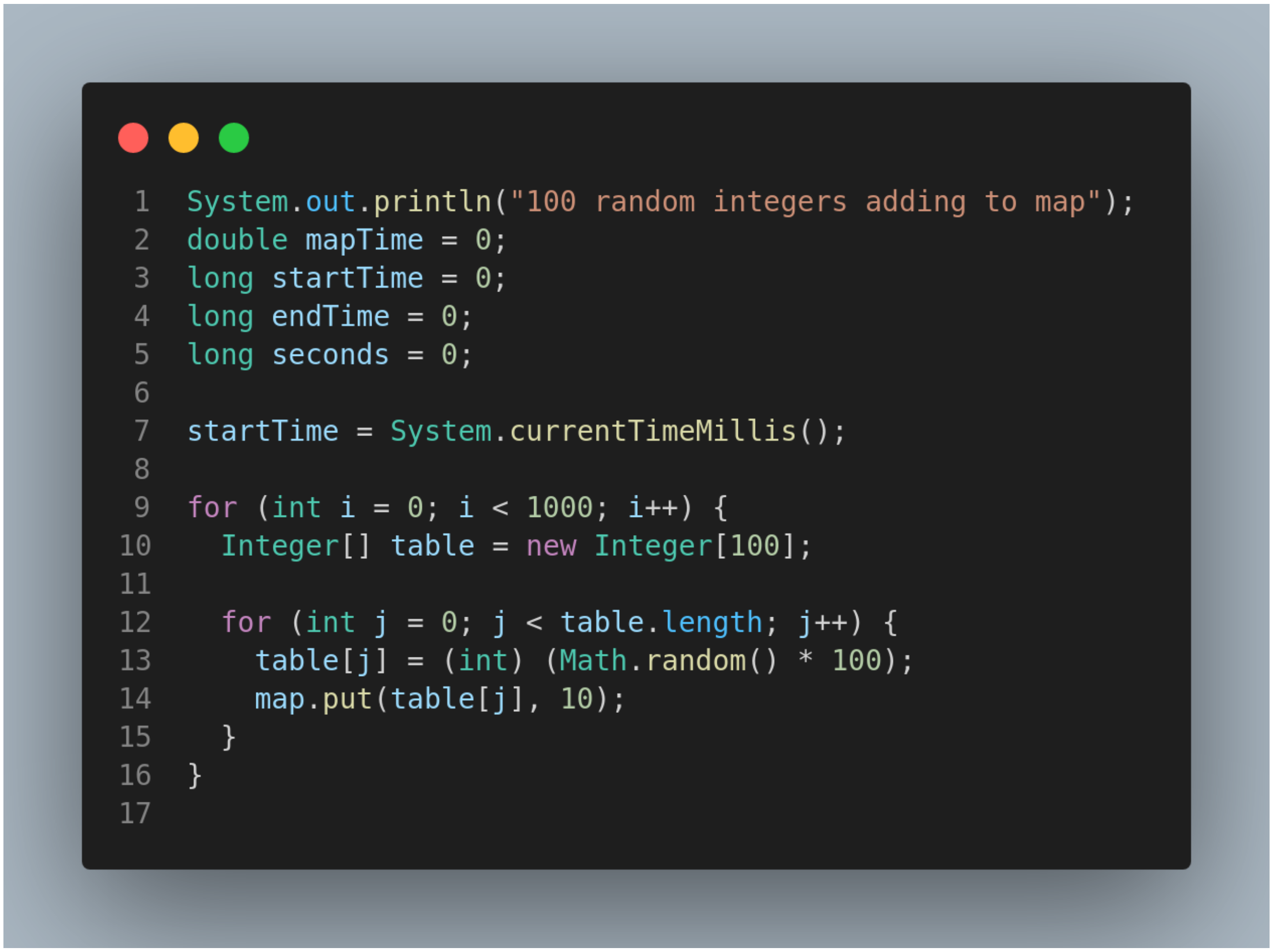
Coalesced chaining protects us from the effects of primary and secondary clustering,
If the chains are short, this strategy is so efficient.
Deletion from a coalesced hash table is expensive like open addressing way.
Resizing the table is very expensive and should be done so rarily.

Q1.2.b

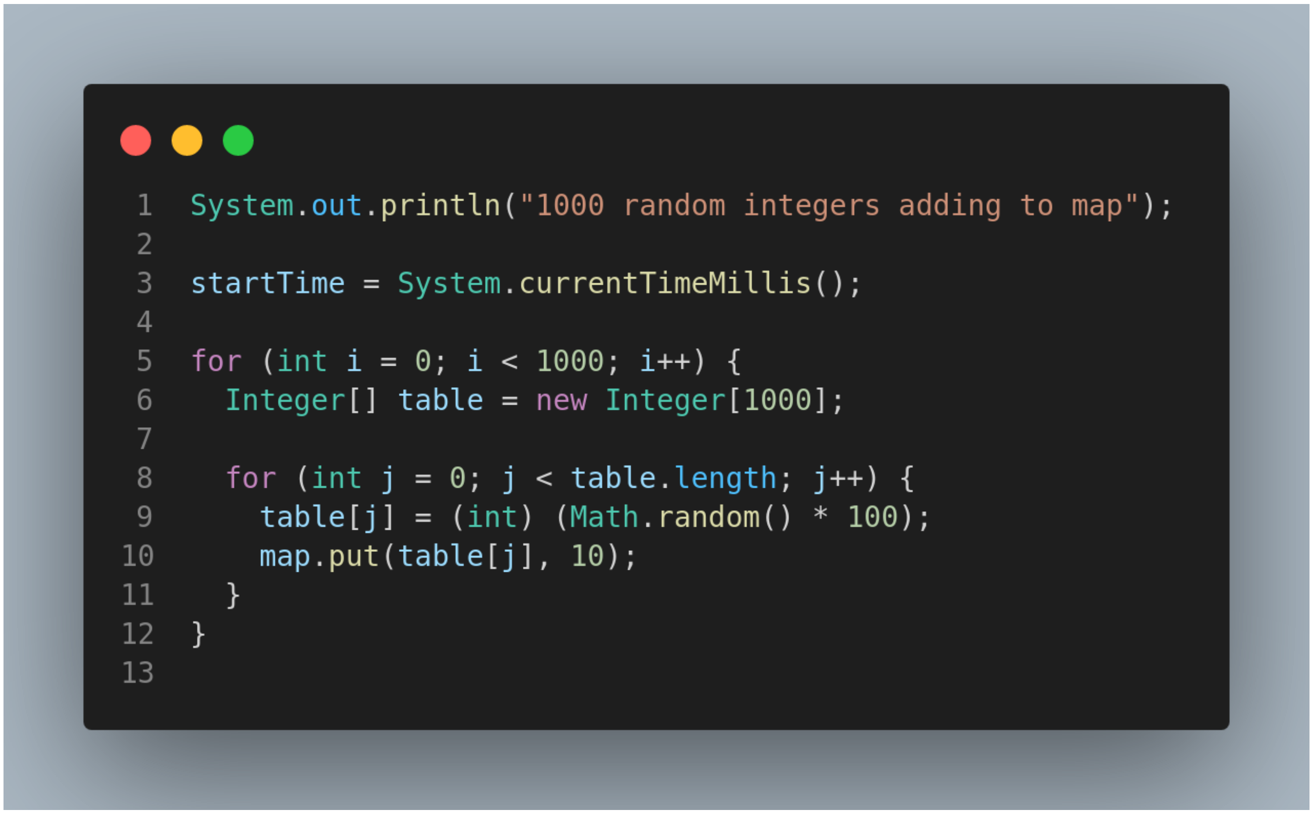
This technique overcomes the cluster issue.
It is very effective method for resolving collisions.
Double hashing is useful if an application requires a smaller
hash table because it can find a empty place in an effective way.

Double hashing can find the next free place faster than the linear probing.
But the computational cost may is higher than linear probing.
Also implementation of double hashing is harder than the others.

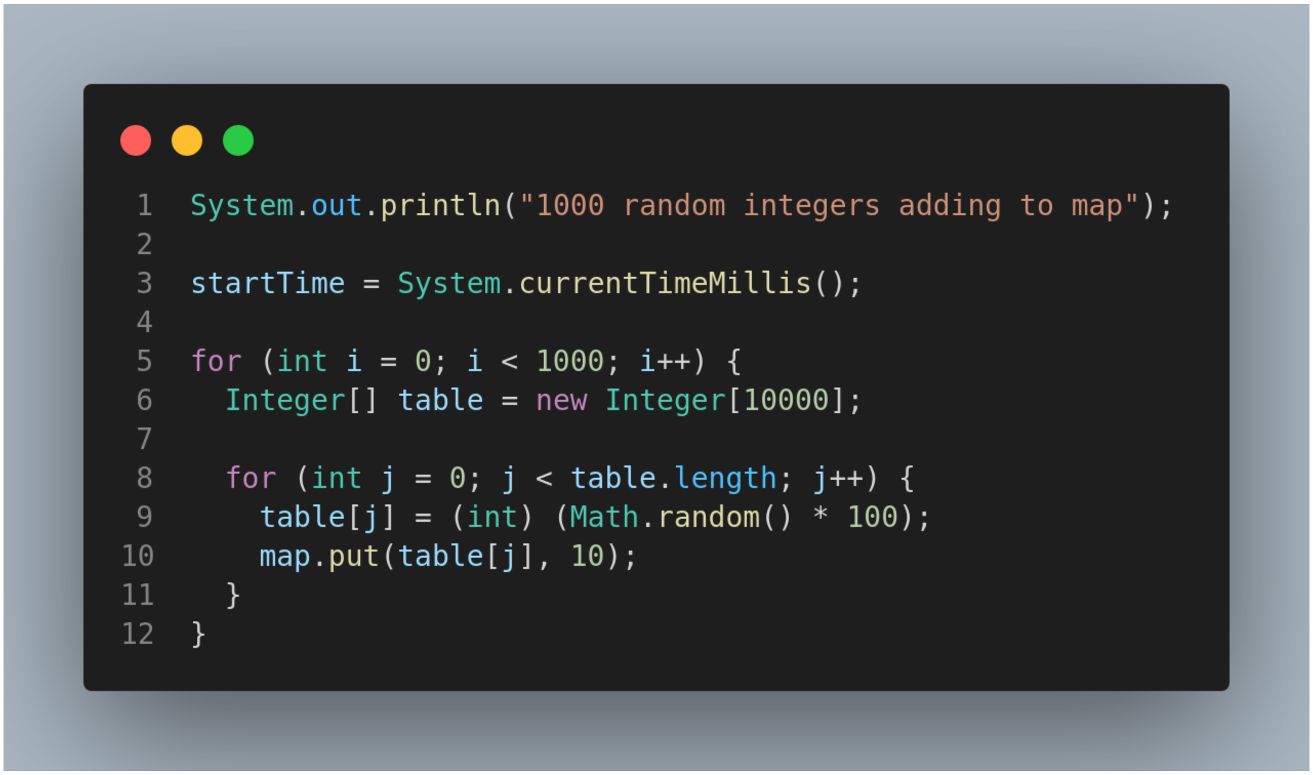
TEST CASES



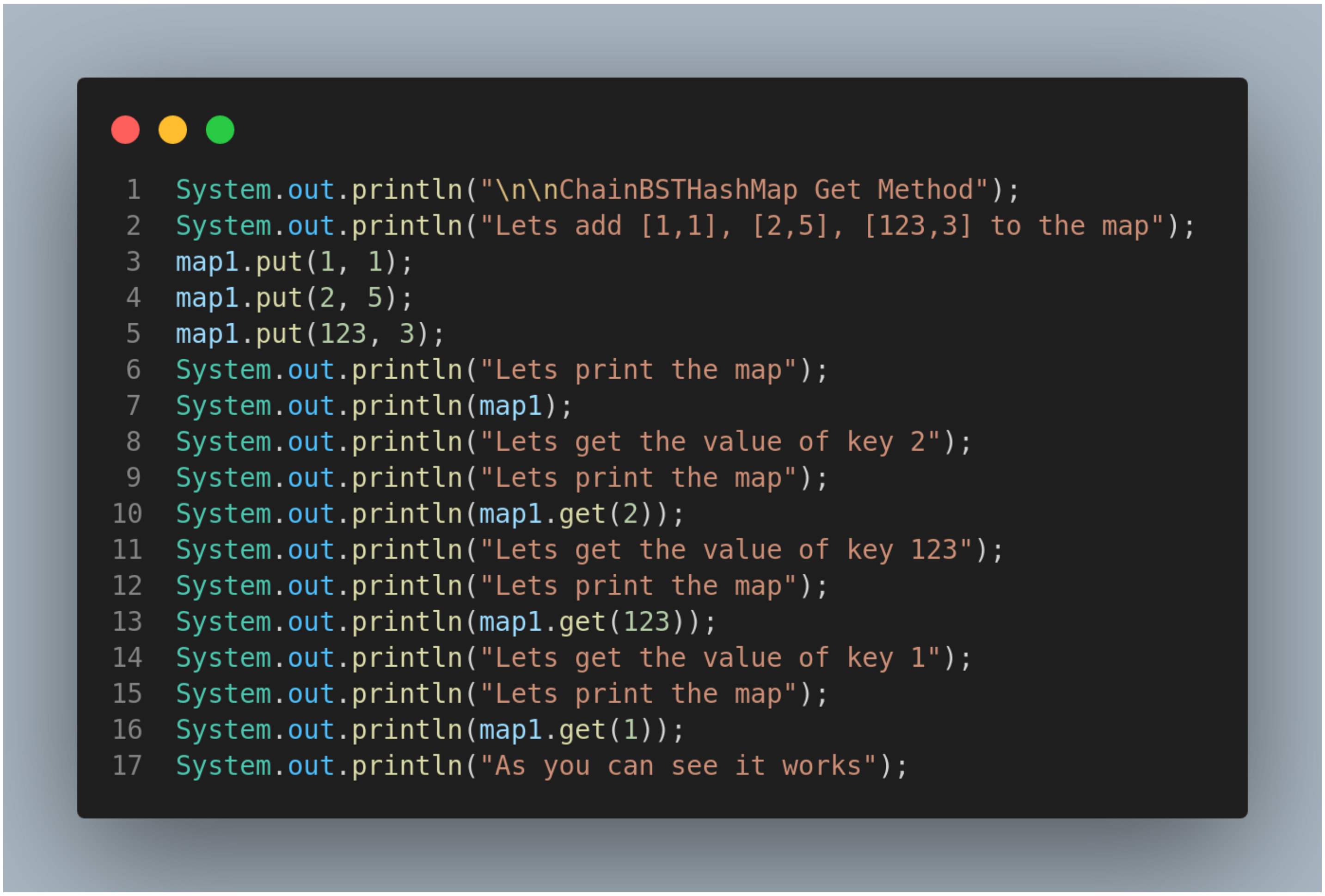
```java
System.out.println("100 random integers adding to map");
double mapTime = 0;
long startTime = 0;
long endTime = 0;
long seconds = 0;

startTime = System.currentTimeMillis();

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[100];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
    map.put(table[j], 10);
  }
}

```



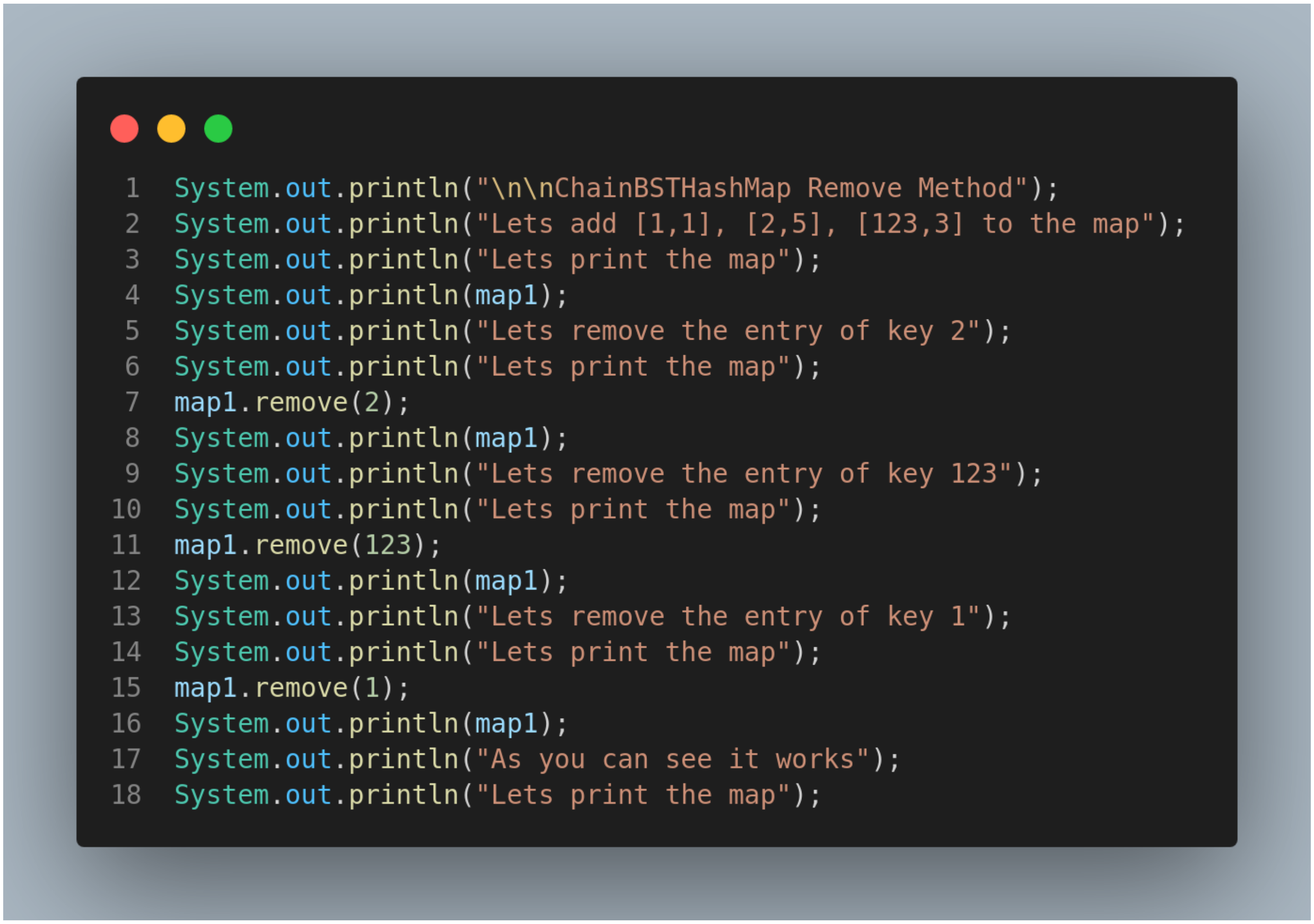
```java
System.out.println("1000 random integers adding to map");

startTime = System.currentTimeMillis();

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[1000];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
    map.put(table[j], 10);
  }
}

```



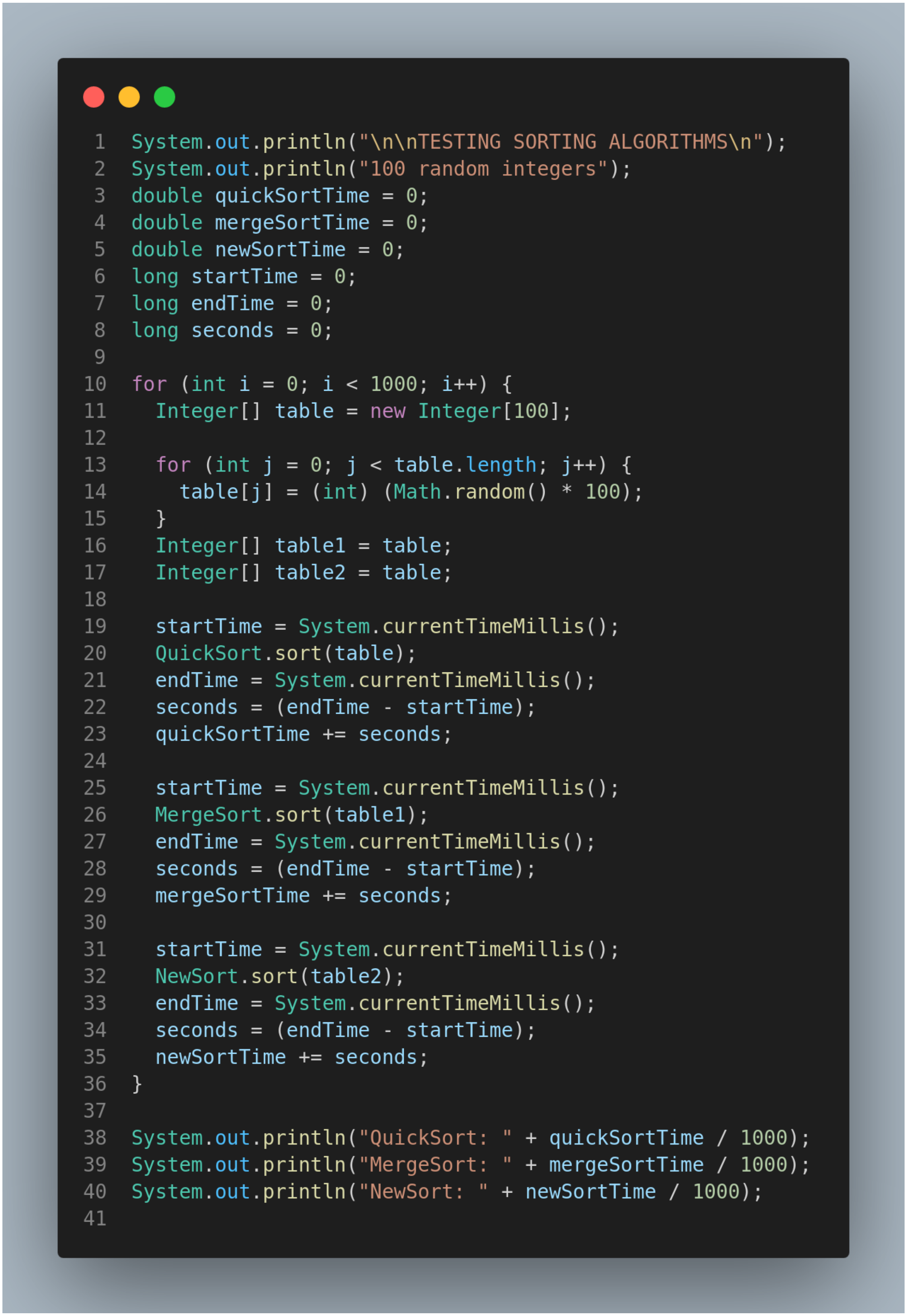
```java
System.out.println("1000 random integers adding to map");

startTime = System.currentTimeMillis();

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[10000];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
    map.put(table[j], 10);
  }
}
```



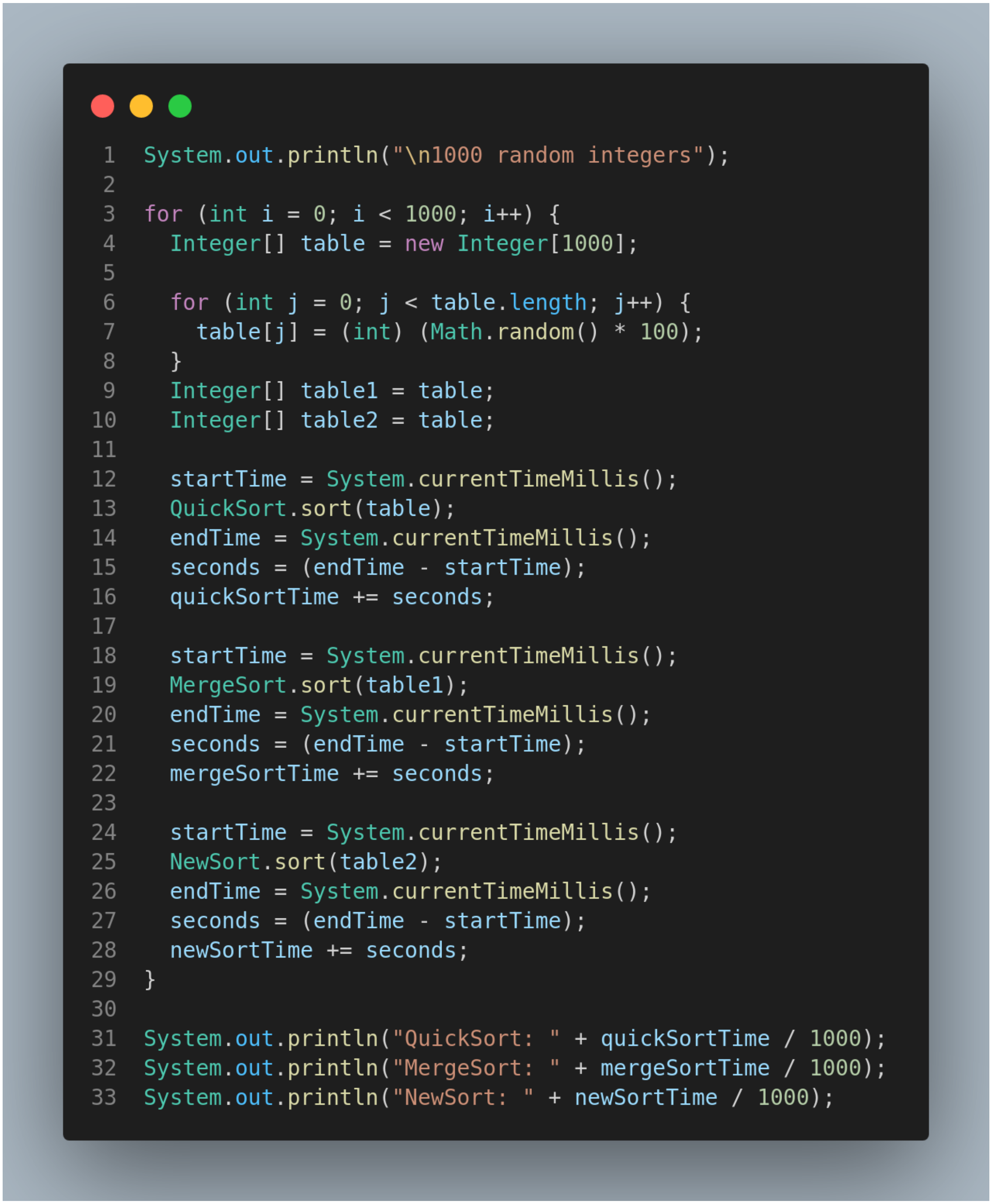
```java
System.out.println("\n\nChainBSTHashMap Get Method");
System.out.println("Lets add [1,1], [2,5], [123,3] to the map");
map1.put(1, 1);
map1.put(2, 5);
map1.put(123, 3);
System.out.println("Lets print the map");
System.out.println(map1);
System.out.println("Lets get the value of key 2");
System.out.println("Lets print the map");
System.out.println(map1.get(2));
System.out.println("Lets get the value of key 123");
System.out.println("Lets print the map");
System.out.println(map1.get(123));
System.out.println("Lets get the value of key 1");
System.out.println("Lets print the map");
System.out.println(map1.get(1));
System.out.println("As you can see it works");
```



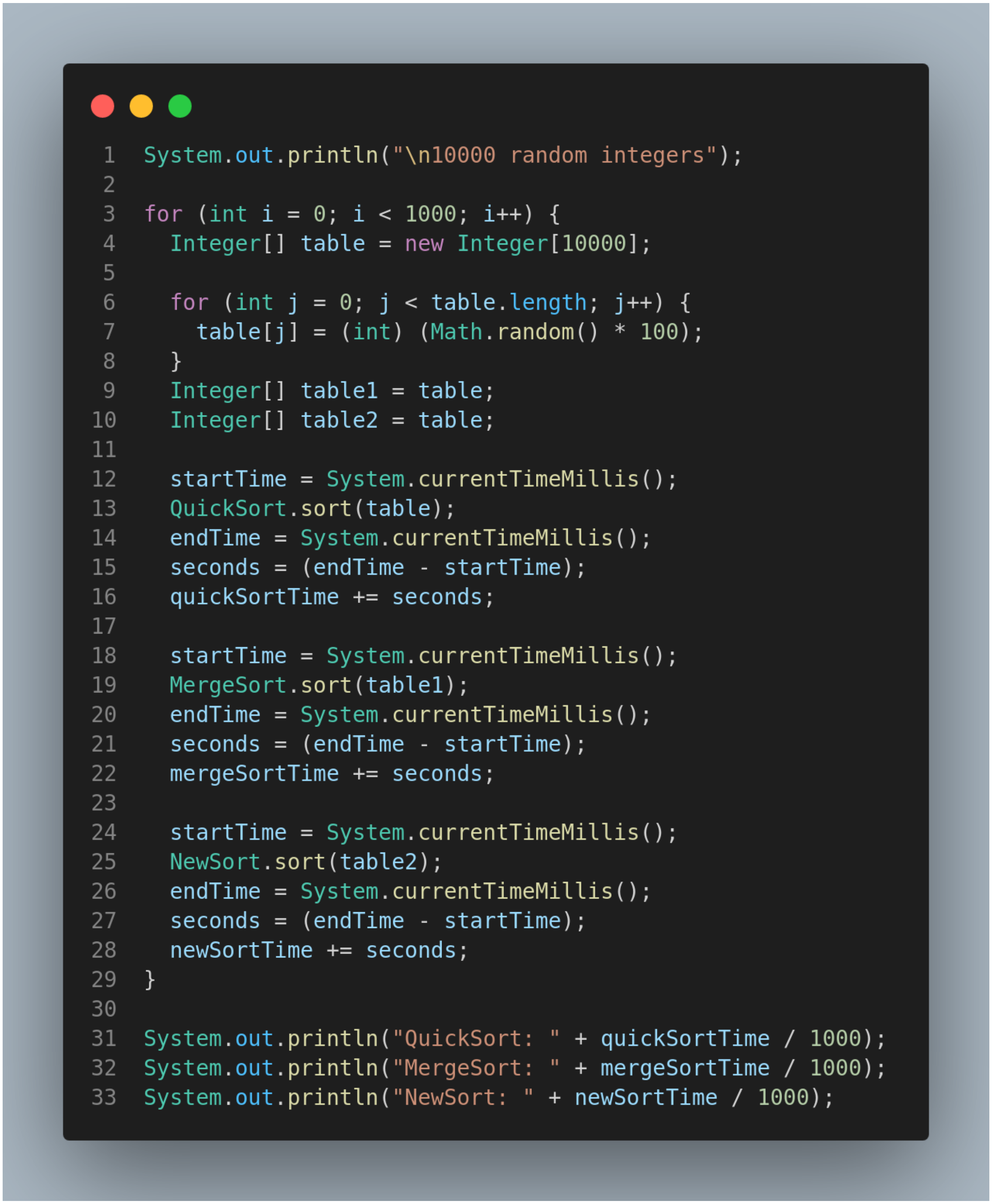
```java
System.out.println("\n\nChainBSTHashMap Remove Method");
System.out.println("Lets add [1,1], [2,5], [123,3] to the map");
System.out.println("Lets print the map");
System.out.println(map1);
System.out.println("Lets remove the entry of key 2");
System.out.println("Lets print the map");
map1.remove(2);
System.out.println(map1);
System.out.println("Lets remove the entry of key 123");
System.out.println("Lets print the map");
map1.remove(123);
System.out.println(map1);
System.out.println("Lets remove the entry of key 1");
System.out.println("Lets print the map");
map1.remove(1);
System.out.println(map1);
System.out.println("As you can see it works");
System.out.println("Lets print the map");
```



```java
System.out.println("\n\nTESTING SORTING ALGORITHMS\n");
System.out.println("100 random integers");
double quickSortTime = 0;
double mergeSortTime = 0;
double newSortTime = 0;
long startTime = 0;
long endTime = 0;
long seconds = 0;

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[100];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
  }
  Integer[] table1 = table;
  Integer[] table2 = table;

  startTime = System.currentTimeMillis();
  QuickSort.sort(table);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  quickSortTime += seconds;

  startTime = System.currentTimeMillis();
  MergeSort.sort(table1);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  mergeSortTime += seconds;

  startTime = System.currentTimeMillis();
  NewSort.sort(table2);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  newSortTime += seconds;
}

System.out.println("QuickSort: " + quickSortTime / 1000);
System.out.println("MergeSort: " + mergeSortTime / 1000);
System.out.println("NewSort: " + newSortTime / 1000);
```

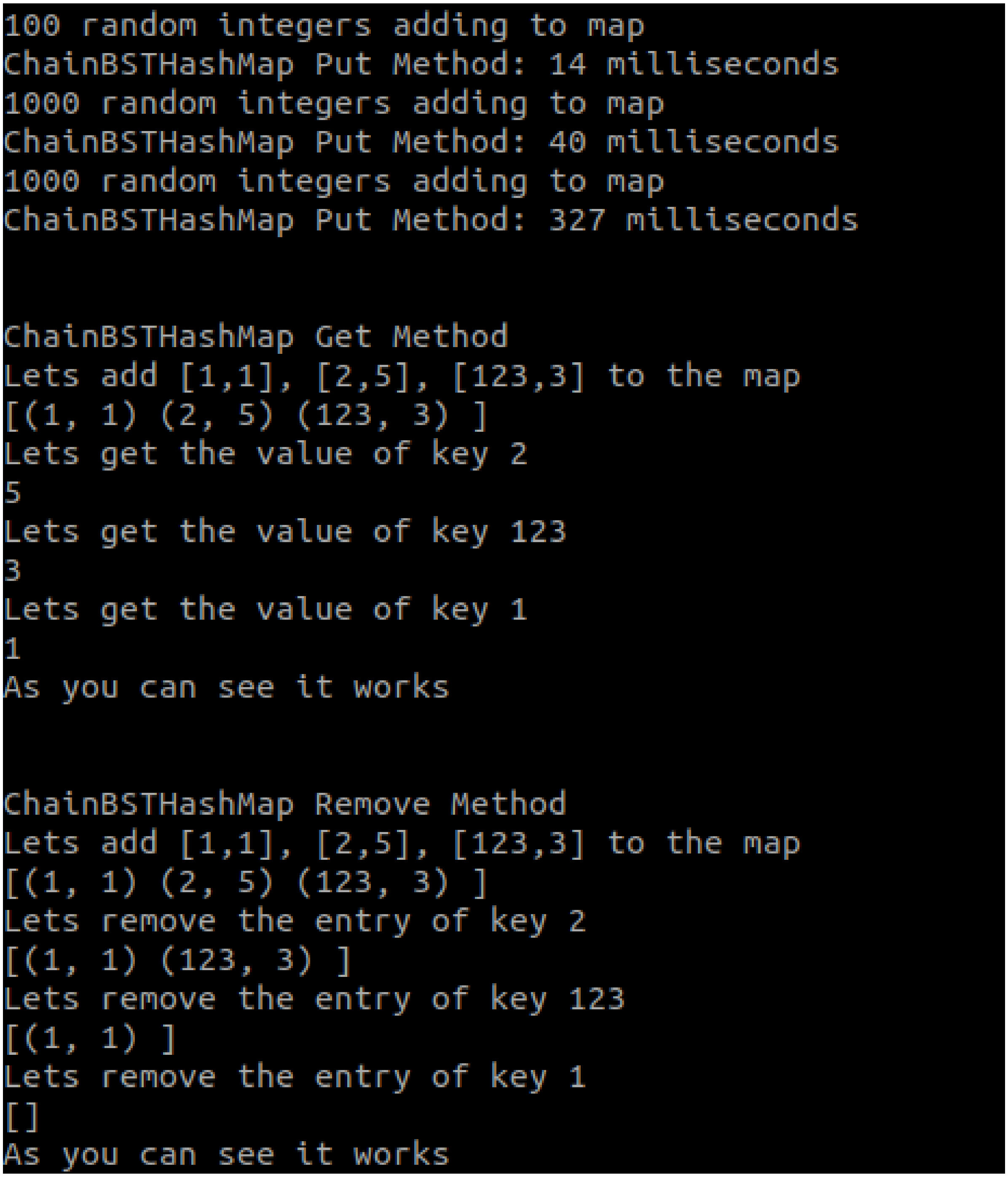


```java
System.out.println("\n1000 random integers");

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[1000];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
  }
  Integer[] table1 = table;
  Integer[] table2 = table;

  startTime = System.currentTimeMillis();
  QuickSort.sort(table);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  quickSortTime += seconds;

  startTime = System.currentTimeMillis();
  MergeSort.sort(table1);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  mergeSortTime += seconds;

  startTime = System.currentTimeMillis();
  NewSort.sort(table2);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  newSortTime += seconds;
}

System.out.println("QuickSort: " + quickSortTime / 1000);
System.out.println("MergeSort: " + mergeSortTime / 1000);
System.out.println("NewSort: " + newSortTime / 1000);
```

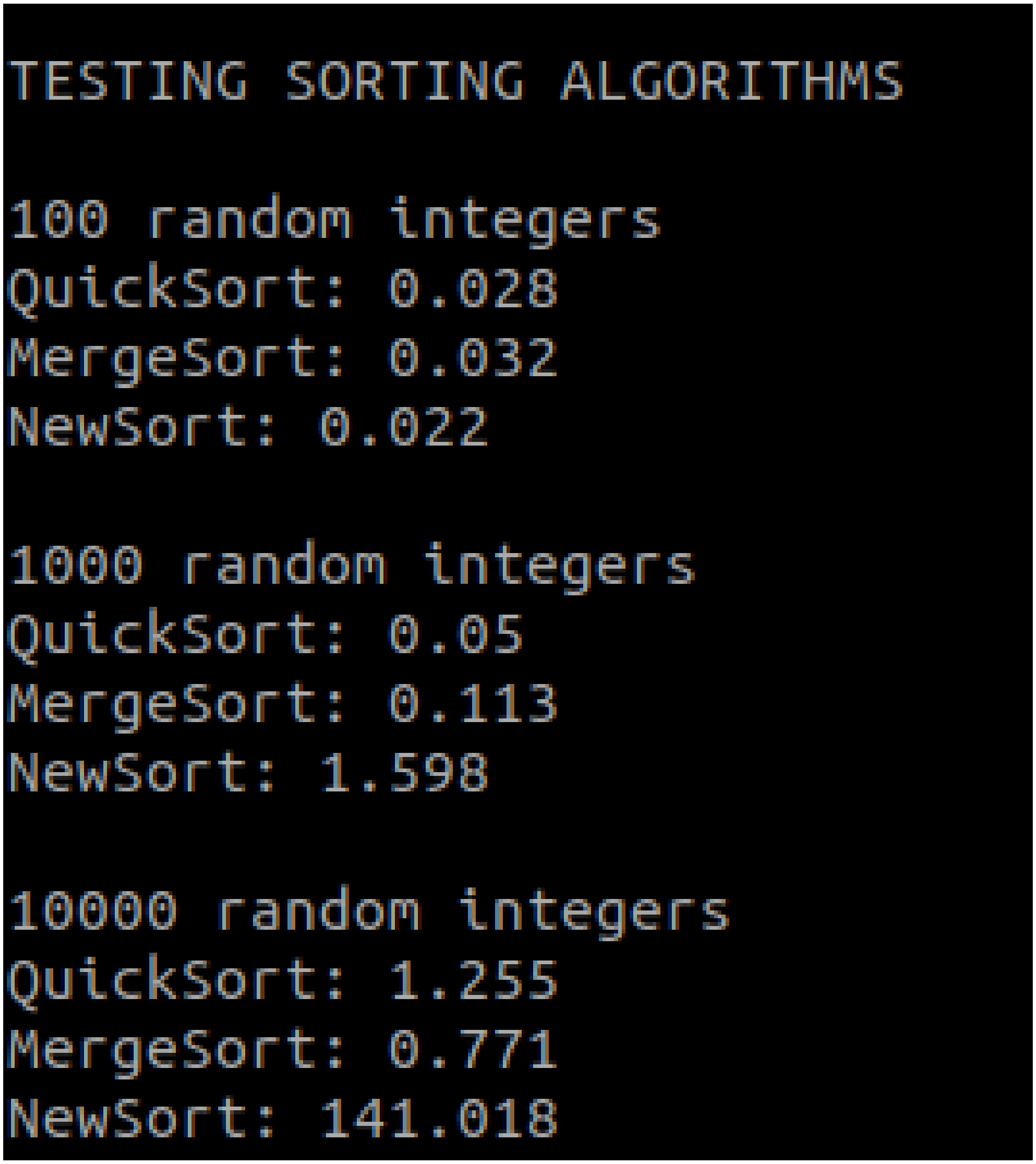


```java
System.out.println("\n10000 random integers");

for (int i = 0; i < 1000; i++) {
  Integer[] table = new Integer[10000];

  for (int j = 0; j < table.length; j++) {
    table[j] = (int) (Math.random() * 100);
  }
  Integer[] table1 = table;
  Integer[] table2 = table;

  startTime = System.currentTimeMillis();
  QuickSort.sort(table);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  quickSortTime += seconds;

  startTime = System.currentTimeMillis();
  MergeSort.sort(table1);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  mergeSortTime += seconds;

  startTime = System.currentTimeMillis();
  NewSort.sort(table2);
  endTime = System.currentTimeMillis();
  seconds = (endTime - startTime);
  newSortTime += seconds;
}

System.out.println("QuickSort: " + quickSortTime / 1000);
System.out.println("MergeSort: " + mergeSortTime / 1000);
System.out.println("NewSort: " + newSortTime / 1000);
```

RUNNING COMMANDS AND RESULTS



```
100 random integers adding to map
ChainBSTHashMap Put Method: 14 milliseconds
1000 random integers adding to map
ChainBSTHashMap Put Method: 40 milliseconds
1000 random integers adding to map
ChainBSTHashMap Put Method: 327 milliseconds


ChainBSTHashMap Get Method
Lets add [1,1], [2,5], [123,3] to the map
[(1, 1) (2, 5) (123, 3) ]
Lets get the value of key 2
5
Lets get the value of key 123
3
Lets get the value of key 1
1
As you can see it works


ChainBSTHashMap Remove Method
Lets add [1,1], [2,5], [123,3] to the map
[(1, 1) (2, 5) (123, 3) ]
Lets remove the entry of key 2
[(1, 1) (123, 3) ]
Lets remove the entry of key 123
[(1, 1) ]
Lets remove the entry of key 1
[]
As you can see it works
```



```
TESTING SORTING ALGORITHMS

100 random integers
QuickSort: 0.028
MergeSort: 0.032
NewSort: 0.022

1000 random integers
QuickSort: 0.05
MergeSort: 0.113
NewSort: 1.598

10000 random integers
QuickSort: 1.255
MergeSort: 0.771
NewSort: 141.018
```