GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 5 Report

Emre SEZER
1901042640

SYSTEM REQUIREMENTS:

I created this homework on Windows 10 using terminal (Java Development Kit).My java version is 11.0.8 .You need to compile “driver.java” in order to test my homework.There are java files, a folder named “Javadoc” that includes javadoc files and report file in my homework.

PROBLEM SOLUTION APPROACH:

I wrote a driver class for testing. I also, wrote a menu method for testing part2. With that menu you can test my homework interactively.You can add/remove/get elements and print Table(You can see the Entry’s index, key and value with printTable() method).

PART 1:

I wrote the MyHashMap class that extends HashMap. This class has an inner class named MapIterator as requested at pdf.

While writing MapIterator class:

I used HashMap’s keyset() method to get keys of the HashMap. Then,

I used Set’s toArray() method to store keys at an K array. During iteration integer named index increases and decreases by one. MapIterator’s prev() and next() methods return the key at the array’s index’ th element. I also keep an integer named counter. It is initially 0. I designed next and prev methods circular. Whenever next() reaches end of the array it moves to the beginning of the array and whenever prev reaches beginning of the array it moves to the end of the array.

Inside the next() method counter increases by 1 and inside the prev() method counter decreases by 1. hasNext() method returns false if counter is equal to array’s length or –(1) * array’s length. Else returns true. With that strategy I keep the number of visited elements in the Map.

I wrote 2 constructors for MapIterator. One takes no parameter, other one takes a key as a parameter. Constructor that takes parameters iterates with a for loop until, it finds the key equals to input.When it found the key index will be that key’s index at the array.If input can’t be found insider array index will still be 0.

PART 2:

First, I wrote KWHashMap interface. All 3 classes that I wrote for Part2 implements this interface. I wrote Entry inner class inside the KWHashMap interface.

KWHashTableLinkedList:

This class uses array of LinkedList's as table and implements KWHashMap. I simply followed textbook's instructions and completed this class. I wrote add,remove methods that uses Chaining technique. I also, wrote rehash method that resizes table and puts all of the elements inside it to the new array.I wrote a method named nextPrime (int num).This method returns the next prime number bigger than input. I used nextPrime(2 * table.length) inside the rehash() method.

KWHashTableTree:

This class uses array of TreeSet as table and implements KWHashMap. I simply followed textbook's instructions for Chaining Technique using LinkedList and modified it. I wrote add,remove methods that uses Chaining technique. I also, wrote rehash method that resizes table and puts all of the elements inside it to the new array.I wrote a method named nextPrime (int num).This method returns the next prime number bigger than input.

I used nextPrime(2 * table.length) inside the rehash() method.

KWHashTableCoalesced:

This class uses array of Entrys as table.

Put Method: If first calculated index is equal to null. Adds an Entry with inputs.

If not, checks the element at the index :table[index].getNext(). While during this loop, if finds an Entry with a key that equals to input key. It changes that Entry's Value to input value. If not continues. Whenever table[index].getNext() is null stops the loop and stores that index to an integer. Then, it starts searching an empty position. In order to do that it calculates index by using hashMethod(This method uses Quadratic Probing to calculate next appropriate index).When it finds an empty position, creates a new Entry with inputs and puts it to that position. Also, Entry at the stored index previously 's next will be the new added Entry's index. With that strategy collided elements will be linked.

Remove Method:

If Entry at the first calculated index is null, returns null. In my design whenever remove method is called , removed Entry's position will be filled by removed element's next. Then, recently moved Entry's position will be filled by it's next. This will go on until index reaches the Entry has no next.That Entry will fill the appropriate position and it's previous position will be null.

With that design whenever remove method is called, Entry that key is equal to input key will be removed and Entrys at that chain will be shifted to appropriate positions.(table[index] = table[table[index].next)

Test Cases :

PART 1:

1) Iterating empty HashMap with next methods: Nothing will be printed
2) Iterating empty HashMap with prev methods: Nothing will be printed
3) Iterating not empty HashMap with next methods(Using Default Constructor): HashMap's Keys will be printed starting from beginning of the HashMap.
4) Using constructor with parameter(Entered key is not at the map): HashMap's Keys will be printed starting from beginning of the HashMap.
5) Iterating not empty HashMap with next methods(Using Constructor with parameter): HashMap's Keys will be printed starting from entered key.
6) Iterating not empty HashMap with prev methods(Using Default Constructor): HashMap's Keys will be printed starting from beginning of the HashMap.
7) Iterating not empty HashMap with prev methods(Using Constructor with parameter): HashMap's Keys will be printed starting from entered key.

PART 2:

KWHashTableLinkedList: (Initial Table Length is 7)

1) Puting elements to the table: Successfully adds Entrys to the table.
2) Printing size() method: Number of Keys in the table will be printed.

3) Printing put methods: returns null if there is no Entry with input key, if there is an Entry with input key returns old value.
4) Printing added elements using get() method: Entry's keys and values will be printed
5) Printing not existing element using get() method: get method will return null.
6) Removing existing element from table using remove method: Removed Entry's value will be printed.
7) Removing bot existing element from table using remove method: Returns null.

KWHashTableTree: (Initial Table Length is 7)

1) Puting elements to the table: Successfully adds Entrys to the table.
2) Printing size() method: Number of Keys in the table will be printed.
3) Printing put methods: returns null if there is no Entry with input key, if there is an Entry with input key returns old value.
4) Printing added elements using get() method: Entry's keys and values will be printed
5) Printing not existing element using get() method: get method will return null.
6) Removing existing element from table using remove method: Removed Entry's value will be printed.
7) Removing bot existing element from table using remove method: Returns null.

KWHashTableCoalesced: (Initial Table Length is 10)

1) Puting elements to the table: Successfully adds Entrys to the table.
2) Printing size() method: Number of Keys in the table will be printed.
3) Printing put methods: returns null if there is no Entry with input key, if there is an Entry with input key returns old value.
4) Printing added elements using get() method: Entry's keys and values will be printed
5) Printing not existing element using get() method: get method will return null.
6) Removing existing element from table using remove method: Removed Entry's value will be printed.
7) Removing not existing element from table using remove method: Returns null.

PERFORMANCE TESTS:

It calculates the time taken for each operation by substracting time right before the operation from time right after end of the operation.

For performing performance result comparison. It performs each step (get(), put(), remove()) with Small-Medium-Large sized data 25 times each with random numbers and takes average. With that approach, i will get more realistic results. Values's unit is nano seconds.

Initially all of the HashTable's lengths are same

I filled the next table with average numbers that calculated.

Integers at the HashTables are in range 0- 20000

I used 50 elements as Small sized data

I used 500 elements as Medium sized data

I used 5000 elements as Large sized data

(Unit is nano seconds)

DataStructure/ Data Size	Accessing Existing Element	Accessing Not Existing Element	Puting Element	Removing Element
KWHashTableLinkedList Small Data	1004	340	952	660
KWHashTableLinkedList Medium Data	1112	804	424	880
KWHashTableLinkedList Large Data	992	736	848	952
KWHashTableTree Small Data	476	360	2432	1452
KWHashTableTree Medium Data	720	752	700	1664
KWHashTableTree Large Data	1124	796	1116	1892
KWHashTableCoalesced Small Data	872	632	176	740
KWHashTableCoalesced Medium Data	392	380	232	980
KWHashTableCoalesced Large Data	508	348	308	832

SCREENSHOTS

PART 1:

~/java/cse222/hw5

```
emr3s@DESKTOP-POGAK7B ~/java/cse222/hw5
$ java driver
PART1:
TESTING next and prev methods with an empty HashMap
As expected nothing is printed.Because, There is no elements inside the HashMap

TESTING next method with HashMap that not empty(with default constructor)
KEY: 32
KEY: 1
KEY: 2
KEY: 3
KEY: 99
KEY: 7
KEY: 42
KEY: 11
KEY: 13
As you can see it started printing from begining of the Map

TESTING constructor with parameter: Entered key is not at the map
KEY: 32
KEY: 1
KEY: 2
KEY: 3
KEY: 99
KEY: 7
KEY: 42
KEY: 11
KEY: 13
As you can see it started printing from beginning of the Map

TESTING next method with HashMap that not empty(with constructor that takes parameter)
KEY: 42
KEY: 11
KEY: 13
KEY: 32
KEY: 1
KEY: 2
KEY: 3
KEY: 99
KEY: 7
As you can see it started printing from entered key

TESTING prev method with HashMap that not empty(with default constructor)
KEY: 32
KEY: 13
KEY: 11
KEY: 42
KEY: 7
KEY: 99
KEY: 3
KEY: 2
KEY: 1
As you can see it started printing from begining of the Map

TESTING prev method with HashMap that not empty(with constructor that takes parameter)
KEY: 42
KEY: 7
KEY: 99
KEY: 3
KEY: 2
KEY: 1
KEY: 32
KEY: 13
KEY: 11
As you can see it started printing from entered key

Type anything to continue testing
```

PART 2:

```
KEY: 7 , VALUE: Value:7
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: Value:NEW41

PRINTING TABLE BEFORE REMOVES:

HASHCODE: 0
KEY: 7 VALUE: Value:7
HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 41 VALUE: Value:NEW41
KEY: 13 VALUE: Value:13
KEY: 6 VALUE: Value:6
TESTING size() BEFORE REMOVES:

SIZE: 10


TESTING remove() with existing elements:

REMOVING KEY: 7 , VALUE: Value:7
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 41 , VALUE: Value:NEW41

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 6 VALUE: Value:6
TESTING size() AFTER REMOVES:

SIZE: 7

TESTING get() method After Remove Operations:

KEY: 3 , VALUE: Value:NEW3
KEY: 12 , VALUE: Value:12
KEY: 13 , VALUE: null
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 51 , VALUE: Value:42
KEY: 6 , VALUE: Value:6
KEY: 7 , VALUE: null
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: null

Type anything to continue testing
```

```
Type anything to continue testing

TESTING KWHashTableCoalesced:

TESTING put() method (returns null if there is no elements with entered key, if there is returns old value):

RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: Value:12
TESTING get() with elements added with put() method:

KEY: 3 , VALUE: Value:3
KEY: 12 , VALUE: Value:NEW12
KEY: 13 , VALUE: Value:13
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 52 , VALUE: Value:42
KEY: 4 , VALUE: Value:4

PRINTING TABLE BEFORE REMOVES:

HASHCODE: 3 KEY: 3 VALUE: Value:3
NEXT: 4
HASHCODE: 4 KEY: 23 VALUE: Value:23
NEXT: 8
HASHCODE: 5 KEY: 25 VALUE: Value:25
HASHCODE: 6 KEY: 6 VALUE: Value:NEW6
HASHCODE: 8 KEY: 4 VALUE: Value:4
HASHCODE: 12 KEY: 12 VALUE: Value:NEW12
NEXT: 16
HASHCODE: 13 KEY: 13 VALUE: Value:13
HASHCODE: 16 KEY: 52 VALUE: Value:42
TESTING size() BEFORE REMOVES:

SIZE: 8


TESTING remove() with existing elements:

REMOVING KEY: 12 , VALUE: Value:NEW12
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 52 , VALUE: Value:42

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 3 KEY: 3 VALUE: Value:3
NEXT: 4
HASHCODE: 4 KEY: 23 VALUE: Value:23
NEXT: 8
HASHCODE: 5 KEY: 25 VALUE: Value:25
HASHCODE: 6 KEY: 6 VALUE: Value:NEW6
HASHCODE: 8 KEY: 4 VALUE: Value:4
TESTING size() AFTER REMOVES:

SIZE: 5

TESTING get() method After Remove Operations:

KEY: 3 , VALUE: Value:3
KEY: 12 , VALUE: null
```

```
HASHCODE: 13 KEY: 13 VALUE: Value:13
HASHCODE: 16 KEY: 52 VALUE: Value:42
TESTING size() BEFORE REMOVES:

SIZE: 8


TESTING remove() with existing elements:

REMOVING KEY: 12 , VALUE: Value:NEW12
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 52 , VALUE: Value:42

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 3 KEY: 3 VALUE: Value:3
NEXT: 4
HASHCODE: 4 KEY: 23 VALUE: Value:23
NEXT: 8
HASHCODE: 5 KEY: 25 VALUE: Value:25
HASHCODE: 6 KEY: 6 VALUE: Value:NEW6
HASHCODE: 8 KEY: 4 VALUE: Value:4
TESTING size() AFTER REMOVES:

SIZE: 5

TESTING get() method After Remove Operations:

KEY: 3 , VALUE: Value:3
KEY: 12 , VALUE: null
KEY: 13 , VALUE: null
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 51 , VALUE: null
KEY: 6 , VALUE: Value:NEW6
KEY: 7 , VALUE: null
KEY: 31 , VALUE: null
KEY: 41 , VALUE: null
RETURNS: null
6
7
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null

PRINTING TABLE AFTER REHASH:

HASHCODE: 1 KEY: 101 VALUE: Value:101
HASHCODE: 3 KEY: 3 VALUE: Value:3
NEXT: 4
HASHCODE: 4 KEY: 23 VALUE: Value:23
NEXT: 8
HASHCODE: 5 KEY: 25 VALUE: Value:25
HASHCODE: 6 KEY: 6 VALUE: Value:NEW6
NEXT: 7
HASHCODE: 7 KEY: -34 VALUE: Value:-34
HASHCODE: 8 KEY: 4 VALUE: Value:4
HASHCODE: 9 KEY: 9 VALUE: Value:9
HASHCODE: 13 KEY: 333 VALUE: Value:333
HASHCODE: 18 KEY: 98 VALUE: Value:98
HASHCODE: 19 KEY: -1 VALUE: Value:-1

Type anything to continue testing
```

~/java/cse222/hw5

```
Type anything to continue testing

PART2:

TESTING KWHashTableLinkedList:

TESTING put() method (returns null if there is no elements with entered key, if there is returns old value):

RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: Value:41
RETURNS: Value:3

TESTING get() with elements added with put() method:

KEY: 3 , VALUE: Value:NEW3
KEY: 12 , VALUE: Value:12
KEY: 13 , VALUE: Value:13
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 51 , VALUE: Value:42
KEY: 6 , VALUE: Value:6
KEY: 7 , VALUE: Value:7
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: Value:NEW41

PRINTING TABLE BEFORE REMOVES:

HASHCODE: 0
KEY: 7 VALUE: Value:7
HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 41 VALUE: Value:NEW41
KEY: 6 VALUE: Value:6
KEY: 13 VALUE: Value:13
TESTING size() BEFORE REMOVES:

SIZE: 10


TESTING remove() with existing elements:

REMOVING KEY: 7 , VALUE: Value:7
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 41 , VALUE: Value:NEW41

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 2
```

```
KEY: 7 , VALUE: Value:7
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: Value:NEW41

PRINTING TABLE BEFORE REMOVES:

HASHCODE: 0
KEY: 7 VALUE: Value:7
HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 41 VALUE: Value:NEW41
KEY: 6 VALUE: Value:6
KEY: 13 VALUE: Value:13
TESTING size() BEFORE REMOVES:

SIZE: 10


TESTING remove() with existing elements:

REMOVING KEY: 7 , VALUE: Value:7
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 41 , VALUE: Value:NEW41

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 6 VALUE: Value:6
TESTING size() AFTER REMOVES:

SIZE: 7

TESTING get() method After Remove Operations:

KEY: 3 , VALUE: Value:NEW3
KEY: 12 , VALUE: Value:12
KEY: 13 , VALUE: null
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 51 , VALUE: Value:42
KEY: 6 , VALUE: Value:6
KEY: 7 , VALUE: null
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: null

Type anything to continue testing
```

~/java/cse222/hw5

```
TESTING KWHashTableTree:

TESTING put() method (returns null if there is no elements with entered key, if there is returns old value):

RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: Value:41
RETURNS: Value:3

TESTING get() with elements added with put() method:

KEY: 3 , VALUE: Value:NEW3
KEY: 12 , VALUE: Value:12
KEY: 13 , VALUE: Value:13
KEY: 25 , VALUE: Value:25
KEY: 23 , VALUE: Value:23
KEY: 51 , VALUE: Value:42
KEY: 6 , VALUE: Value:6
KEY: 7 , VALUE: Value:7
KEY: 31 , VALUE: Value:31
KEY: 41 , VALUE: Value:NEW41

PRINTING TABLE BEFORE REMOVES:

HASHCODE: 0
KEY: 7 VALUE: Value:7
HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
KEY: 31 VALUE: Value:31
KEY: 3 VALUE: Value:NEW3
HASHCODE: 4
KEY: 25 VALUE: Value:25
HASHCODE: 5
KEY: 12 VALUE: Value:12
HASHCODE: 6
KEY: 41 VALUE: Value:NEW41
KEY: 13 VALUE: Value:13
KEY: 6 VALUE: Value:6
TESTING size() BEFORE REMOVES:

SIZE: 10


TESTING remove() with existing elements:

REMOVING KEY: 7 , VALUE: Value:7
REMOVING KEY: 13 , VALUE: Value:13
REMOVING KEY: 41 , VALUE: Value:NEW41

TESTING remove() with non-existing elements:

REMOVING KEY: 202 , VALUE: null
REMOVING KEY: -10 , VALUE: null
REMOVING KEY: 99 , VALUE: null

PRINTING TABLE AFTER REMOVES:

HASHCODE: 2
KEY: 51 VALUE: Value:42
KEY: 23 VALUE: Value:23
HASHCODE: 3
```

PART 2 PERFORMANCE RESULTS:

```
MEDIUM-SIZE:
Average: 752.0

LARGE-SIZE:
Average: 796.0

ADDING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 2432.0

MEDIUM-SIZE:
Average: 700.0

LARGE-SIZE:
Average: 1116.0

REMOVING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 1452.0

MEDIUM-SIZE:
Average: 1664.0

LARGE-SIZE:
Average: 1892.0


Type anything to continue testing

ACCESSING EXISTING ELEMENTS WITH KWHashTableCoalesced:
SMALL-SIZE:
Average: 872.0

MEDIUM-SIZE:
Average: 392.0

LARGE-SIZE:
Average: 508.0

ACCESSING NOT EXISTING ELEMENTS WITH KWHashTableCoalesced:
SMALL-SIZE:
Average: 632.0

MEDIUM-SIZE:
Average: 380.0

LARGE-SIZE:
Average: 348.0

ADDING ELEMENTS WITH KWHashTableCoalesced:
SMALL-SIZE:
Average: 176.0

MEDIUM-SIZE:
Average: 232.0

LARGE-SIZE:
Average: 308.0

REMOVING ELEMENTS WITH KWHashTableCoalesced:
SMALL-SIZE:
Average: 740.0

MEDIUM-SIZE:
Average: 980.0

LARGE-SIZE:
Average: 832.0


emr3s@DESKTOP-POGAK7B ~/java/cse222/şaka
$
```

~/java/cse222/şaka

```
6
7
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null
RETURNS: null

PRINTING TABLE AFTER REHASH:

HASHCODE: 1 KEY: 101 VALUE: Value:101
HASHCODE: 3 KEY: 3 VALUE: Value:3
NEXT: 4
HASHCODE: 4 KEY: 23 VALUE: Value:23
NEXT: 8
HASHCODE: 5 KEY: 25 VALUE: Value:25
HASHCODE: 6 KEY: 6 VALUE: Value:NEW6
NEXT: 7
HASHCODE: 7 KEY: -34 VALUE: Value:-34
HASHCODE: 8 KEY: 4 VALUE: Value:4
HASHCODE: 9 KEY: 9 VALUE: Value:9
HASHCODE: 13 KEY: 333 VALUE: Value:333
HASHCODE: 18 KEY: 98 VALUE: Value:98
HASHCODE: 19 KEY: -1 VALUE: Value:-1

Type anything to continue testing

TESTING PERFORMANCES:

ACCESSING EXISTING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 1004.0

MEDIUM-SIZE:
Average: 1112.0

LARGE-SIZE:
Average: 992.0

ACCESSING NOT EXISTING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 340.0

MEDIUM-SIZE:
Average: 804.0

LARGE-SIZE:
Average: 736.0

ADDING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 952.0

MEDIUM-SIZE:
Average: 424.0

LARGE-SIZE:
Average: 848.0

REMOVING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 660.0

MEDIUM-SIZE:
Average: 880.0

LARGE-SIZE:
Average: 952.0


Type anything to continue testing
```

```
MEDIUM-SIZE:
Average: 804.0

LARGE-SIZE:
Average: 736.0

ADDING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 952.0

MEDIUM-SIZE:
Average: 424.0

LARGE-SIZE:
Average: 848.0

REMOVING ELEMENTS WITH KWHashTableLinkedList:
SMALL-SIZE:
Average: 660.0

MEDIUM-SIZE:
Average: 880.0

LARGE-SIZE:
Average: 952.0


Type anything to continue testing

ACCESSING EXISTING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 476.0

MEDIUM-SIZE:
Average: 720.0

LARGE-SIZE:
Average: 1124.0

ACCESSING NOT EXISTING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 360.0

MEDIUM-SIZE:
Average: 752.0

LARGE-SIZE:
Average: 796.0

ADDING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 2432.0

MEDIUM-SIZE:
Average: 700.0

LARGE-SIZE:
Average: 1116.0

REMOVING ELEMENTS WITH KWHashTableTree:
SMALL-SIZE:
Average: 1452.0

MEDIUM-SIZE:
Average: 1664.0

LARGE-SIZE:
Average: 1892.0


Type anything to continue testing
```