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

BURAK KOCAUSTA
1901042605

1. SYSTEM REQUIREMENTS

There are 3 different packages. One of them is for Hash Table implementations. Inside this package, 2 different class implements KWHashMap interface. These 5 methods is required to be implemented.

```java
public interface KWHashMap<K, V> {

    /**
     * Checks if table is empty or not.
     * @return true if table is empty, otherwise false.
     */
    boolean isEmpty ( );

    /**
     * Returns total number of the keys.
     * @return size of non-null elements of the table.
     */
    int size ( );

    /**
     * Puts the given pair to the table, if key exists sets the new value.
     * @param key key
     * @param value value
     * @return if key is new returns null, otherwise returns old value.
     */
    V put ( K key, V value );

    /**
     * Returns the value according to the key.
     * @return value of the given key, if key does not exists, returns null.
     * @param key key to access value.
     */
    V get ( Object key );

    /**
     * Removes the pair according to the given key.
     * @param key key
     * @return the deleted object.
     */
    V remove ( Object key );
}
```

One of these classes is BSTHashTableChain. It uses BinarySearchTree's as buckets, and its threshold is 3.0. It requires Key and Value as generic parameters. Key parameter required to be comparable.

```java
public class BSTHashTableChain<K extends Comparable<K>, V> implements KWHashMap<K, V> {

    /**
     * Hash table
     */
    private BinarySearchTree<Entry<K, V>>[] table;

    /**
     * Total number of keys.
     */
    private int numKeys = 0;

    /**
     * Initial capacity of the array.
     */
    private static final int INITIAL_CAPACITY = 11;

    /**
     * Maximum load factor.
     */
    private static final double LOAD_THRESHOLD = 3.0;
```

It has only no parameter constructor.

```java
KWHashMap<Integer, String> table1 = new BSTHashTableChain<Integer, String>();
```

Other KWHashMap implementation is, HashTableHybrid class. It is a different version of open addressing method which combines double hashing and coalesced hashing. It does not requires comparable key value, but of course it requires key and value classes as generics.

```java
public class HashTableHybrid<K, V> implements KWHashMap<K, V> {

    /**
     * Hash table
     */
    private Entry<K, V>[] table;

    /**
     * Initial capacity of the table.
     */
    private static final int INITIAL_CAPACITY = 11;

    /**
     * Maximum load factor.
     */
    private double LOAD_THRESHOLD = 0.5;

    /**
     * Total number of keys.
     */
    private int numKeys = 0;

    /**
     * Prime num that will be used in hash number calculation.
     */
    private int PRIME_NUM = 7;
```

It has only no parameter constructor.

```
KWHashMap<Integer, String> hybrid1 = new HashTableHybrid<Integer, String>();
```

Other package is for sorting algorithms. They all requires comparable arrays, and, they only have static sort method.

```
MergeSort.sort(arr1);
```
```
QuickSort.sort(arr1);
```
```
NewSort.sort(arr1);
```

Other package is for Binary Search Tree. It implements SearchTree, Iterable interfaces, and extends BinaryTree class. BinaryTree and SearchTree classes are taken from previous homework. This BinarySearchTree class is iterable. It differs from regular BinarySearchTree class with iterable. It requires comparable class as generics.

```
public class BinarySearchTree<E extends Comparable<E>> extends BinaryTree<E> implements SearchTree<E>, Iterable<E>
```

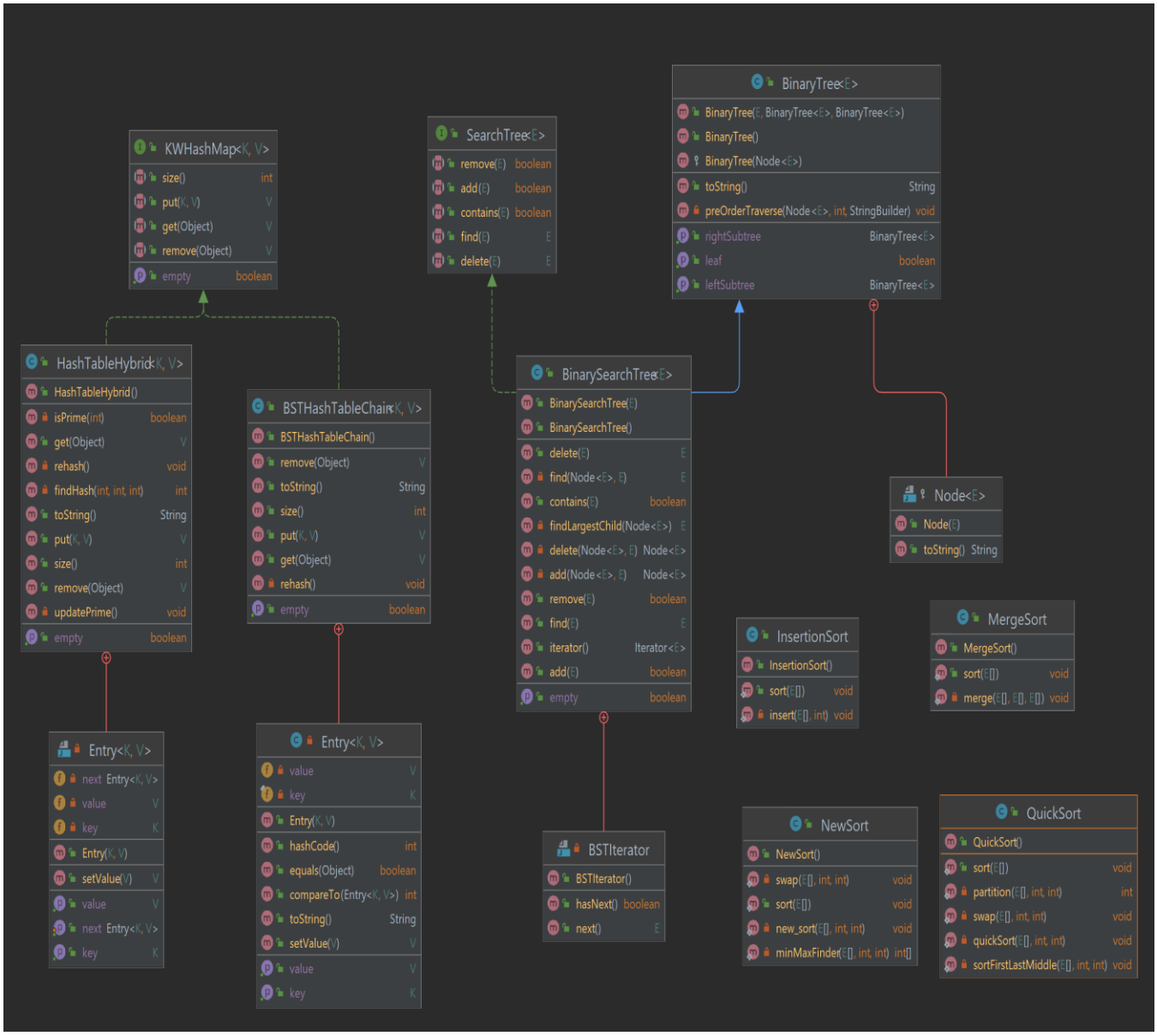
It has no parameter, and one parameter constructor. One parameter constructor requires a comparable object to construct the tree.

```
SearchTree<Integer> bst1 = new BinarySearchTree<Integer>();
SearchTree<String> bst2 = new BinarySearchTree<String>("laughter");
```

All these implementations are inside these packages, therefore this import statement is required.

```
import HashTableGTU.*;
import BinarySearchTreeGTU.*;
import SortGTU.*;
```

2. CLASS DIAGRAM



3. PROBLEM SOLUTION APPROACH

Q1.1

In this part, Binary Search Tree version of chaining technique is wanted instead of linked lists for KWHashMap implementation. First problem of this implementation is regular binary search trees don't have an iterator.

Rehashing and put operations require iterator to make these operations in averagely constant time. Iterator is not necessary for remove operation but necessary for other two. Other difference from linked list implementation of chaining technique is, this class's generic input (at least one I considered key because they must be unique) must be comparable. Because binary search tree constructs the tree according to their comparison result. These binary trees will be used as buckets, so there must be array of binary search trees. Each of these binary trees hold entry in their node. Of course, these entries must be comparable. So, I implemented compareTo method for entries, and comparing key's hashcode value is enough to construct a tree.

For the implementing Iterator of Binary Search Tree, I made a preorder traverse, while doing that preorder traverse, I used ArrayDeque to use it as a stack. I popped the current node, and push all right child, then push the left child. With this traverse, left child come before right child, therefore tree is traversed preorderly. I called this iterator sometimes in enhanced for loop, sometimes using next() methods.

Before making analysis, my expectation is this implementation should be slower than linked list implementation, because linked lists are not have to be ordered, and makes constant time insertion operation(from head and tail). Buckets size thought as constant, but when probe comparison is done for insertion, linked list should be better than binary search tree. As opposed to this, get and remove operation's might faster than linked list because these are O(logn) in their average cases. In general, I still think linked list is better, but if search operation is important, binary search tree might be a good choice.

Q1.2(Problem solution and explanations of Coalesced, Double Hashing Techniques)

In this part, KWHashMap implementation is wanted again, but this time it is a combination of two different technique, which are Coalesced Hashing and Double Hashing. Firstly, I researched about these techniques.

a. Coalesced Hashing

Coalesced Hashing is a technique which is similar to regular open addressing, because it uses the hash table itself. It handles collision with linking them.

While doing it, new item is put on the below of the table. While searching the wanted key, traversing happens through the links if collision is happened.

Advantage of this technique over standard open addressing is, cluster is handled better, because links can be used while traversing. There are more unnecessary probes in regular open addressing. For searching coalesced hashing will work faster. There is no significant advantage or disadvantage of insertion and search. Disadvantage is critical for remove operation, it might quite painful to implement and expensive(time). Because it is hard to track key's starting hash. If search and insert operation is more important than removal, this technique can be preferred rather than regular open addressing.

Advantage of this technique over standard chaining is that it requires less memory space. Searching performance is similar for both. But it is obvious that implementing chaining technique is much simpler than coalesced hashing. Also, deletion in standard chaining technique is better than coalesced hashing.

b. Double Hashing

Double hashing is a technique which is also similar to regular open addressing, because of storing data on the table itself. It handles collision with computing second hash when it happened. While doing it, there is a prime number which is greatest prime before table size. So, when rehashing happens, there must be prime number calculation. Hash calculation happens for each probe to determine the index of given element.

Advantage of this technique over standard open addressing is, cluster is handled better, because hash function can be used while searching. It is useful because there are less probes in double hashing, reason is only the same candidates for that index is visited. In open addressing, there could be probe between keys which don't have same initial hash values. Disadvantage of double hashing is there are more computation in this technique. In each search, insertion, deletion operation hash indexes must be calculated, and to calculate them correct; there must be prime number. This prime number must largest prime number which is less than table size. When rehashing happens prime number must be calculated, therefore rehashing is expensive.

Advantage of this technique over standard chaining is, it requires less memory space, distribution of elements could better. Disadvantages are same with open addressing, which are rehashing(prime), computation requirements.

Problem Solution Approach for Question 2

After this research about coalesced and double hashing, I started to implement HybridHashTable. Firstly, I thought of rehashing operations time complexity. There is a prime number so this number must be the next prime which is greatest before 0.8*(table size). Calculating the prime number is costly, but if load threshold is not small, there will be less rehash operation. Therefore, I decided to set load threshold to 0.7. After making empirical analysis, 0.5 load threshold gave better results, so I changed it to 0.5. While implementing, I saw a problem(?) on the given pdf. For the coalesced hashing "i" value which is probe number, starts from 0, but in the pdf, it starts from 1 according to the given table. I considered the table; therefore, my implementation counts probe from 1. For indicating the next element, I hold an entry reference inside entry class. It refers to the next elements which must be in same index. While traversing, this next field is used as a single linked list. Put and remove operations are made like single linked lists. While removing, I transform the removed object in the array like dummy, but this dummy is not like the regular open addressing implementation's dummy. It could be used again. It is dummy, because while traversing with next fields, we don't know where the entry is in the array, so modifying its key is the only choice. Calculating the index again might be a choice but modifying its key is the optimal solution I found. I assume this is faster than regular open addressing, but rehashing might cause problems for lower load thresholds for greater sizes and space requirements are greater than regular open addressing technique. I don't think this is preferable just because, its remove operation. But it handles collision better than open addressing technique.

Q1.3

Complexity analysis (empirical) are made in Analysis part.

Q2(2.1,2.2,2.3)

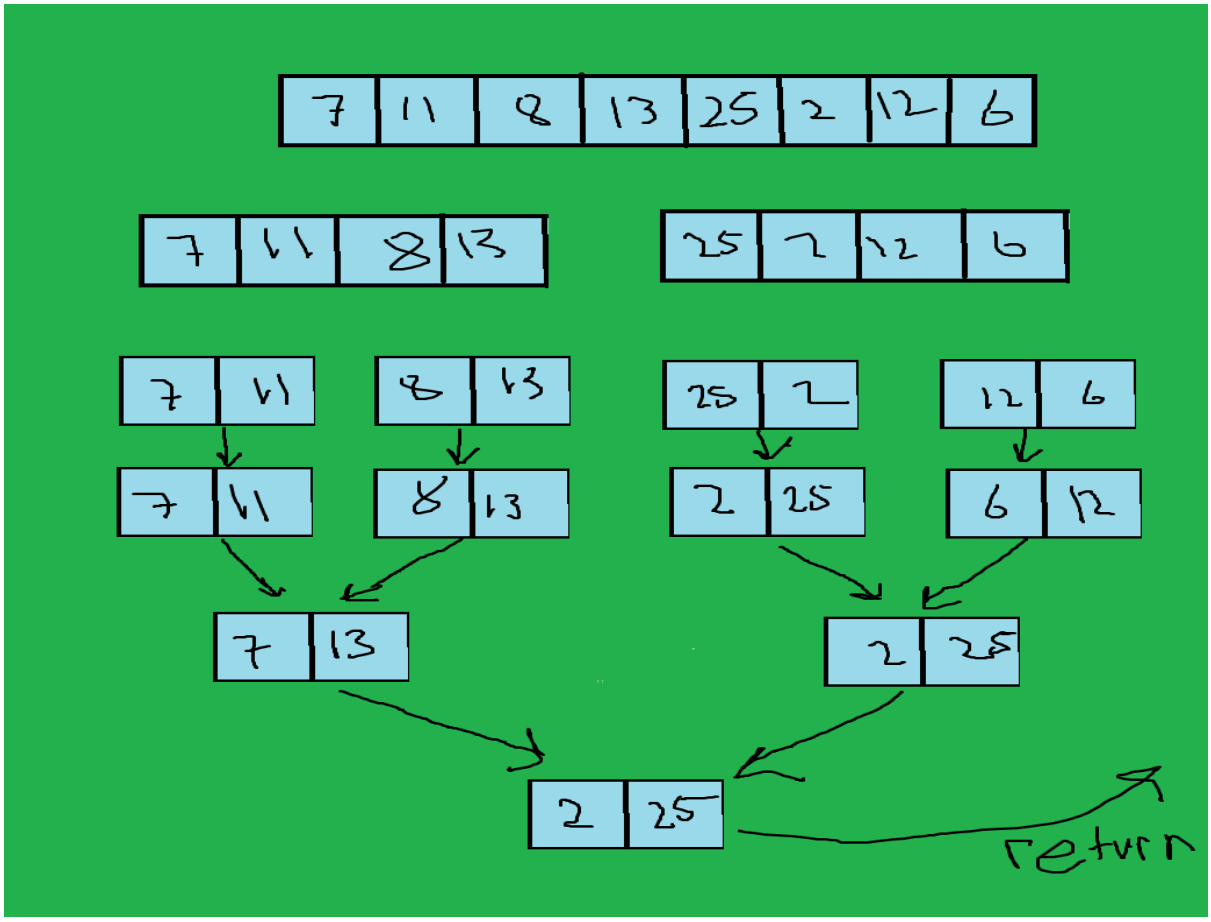
For the merge sort algorithm, sorting performed recursively for both side of that array. Dividing happens till there is one element that is already sorted. After that merging happens. I optimized the merge algorithm given in the lectures with adding insertion sort if number of elements are less than 10. I created a class named MergeSort and called it like MergeSort.sort(arr).

For the quick sort algorithm, sorting performed recursively with partition according to the pivot value. I optimized it a bit with sorting first last and middle element. After this sorting median becomes pivot. It is done to prevent the worst case. I created a class named QuickSort and called it like QuickSort.sort(arr).

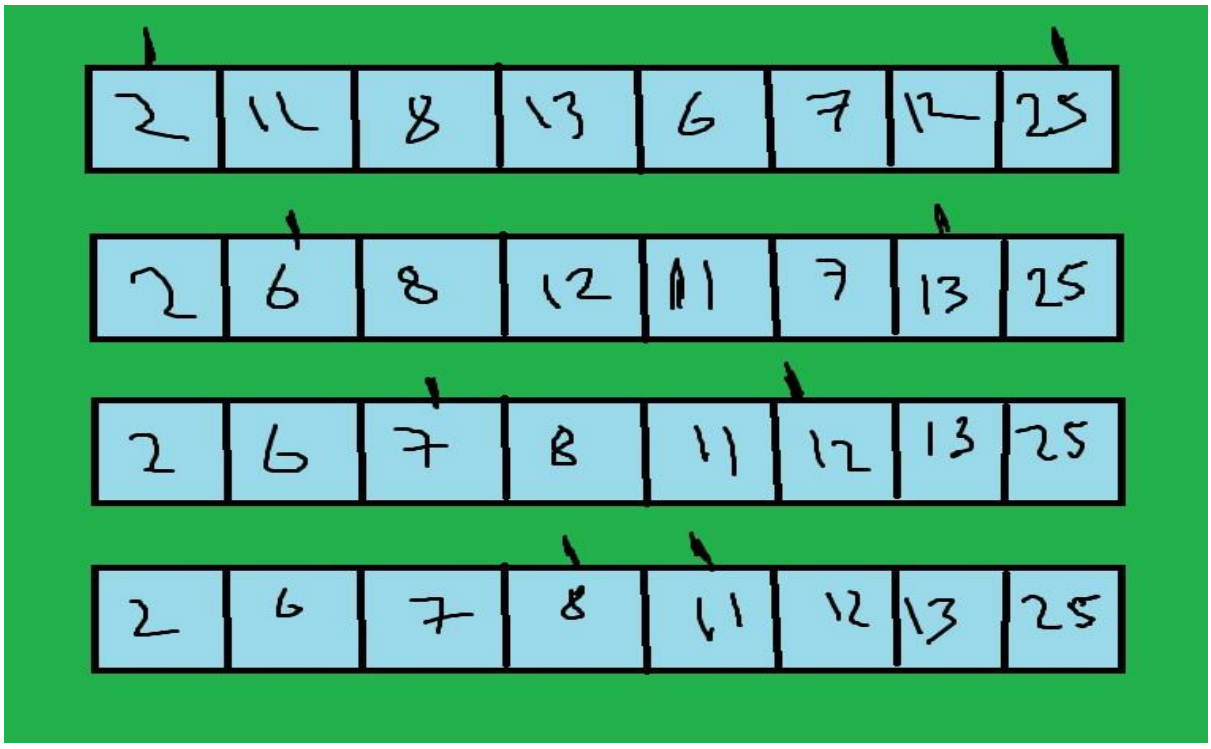
For the new sort algorithm, there is a method which finds minimum and maximum number's indexes recursively, and puts them head and tail, therefore array is sorted from outside the inside like compressing. I encountered a problem in the algorithm when greatest item is in the head. It is swapped before its swap operation comes. Solved it with adding a condition. I made minMaxFinder method which returns integer array. With this return value min and max indexes are accessible.

Complexity analysis (empirical and theoretical) are made in Analysis part.

MinMaxFinder works like this(other page):



Arrays situation after each sort step:



4. Complexity Analysis(Empirical and Theoretical)

Q1(empirical analysis)

```
java Driver
___HashTables Performance Test___

100 tests for 100 1000 10000 random datasets(same for each).
Processing.....................................................................................................
HashTableHybrid (size = 100)
put() = 81395
get() = 37
remove() = 50
size() = 1
isEmpty() = 1

BSTHashTableChain (size = 100)
put() = 1504
get() = 92
remove() = 69
size() = 1
isEmpty() = 1

HashTableHybrid (size = 1000)
put() = 1019
get() = 273
remove() = 278
size() = 0
isEmpty() = 0

BSTHashTableChain (size = 1000)
put() = 332
get() = 80
remove() = 76
size() = 0
isEmpty() = 0

HashTableHybrid (size = 10000)
put() = 10474
get() = 14744
remove() = 13800
size() = 0
isEmpty() = 0

BSTHashTableChain (size = 10000)
put() = 219
get() = 108
remove() = 91
size() = 0
isEmpty() = 0
```

Another tests:

```
java Driver
___HashTables Performance Test___

100 tests for 100 1000 10000 random datasets(same for each).
Processing.....................................................................................................
HashTableHybrid (size = 100)
put() = 194
get() = 34
remove() = 67
size() = 1
isEmpty() = 1

BSTHashTableChain (size = 100)
put() = 1274
get() = 87
remove() = 62
size() = 1
isEmpty() = 1

HashTableHybrid (size = 1000)
put() = 7572
get() = 252
remove() = 263
size() = 0
isEmpty() = 0

BSTHashTableChain (size = 1000)
put() = 311
get() = 80
remove() = 62
size() = 1
isEmpty() = 1

HashTableHybrid (size = 10000)
put() = 9797
get() = 14344
remove() = 13443
size() = 0
isEmpty() = 0

BSTHashTableChain (size = 10000)
put() = 208
get() = 104
remove() = 86
size() = 0
isEmpty() = 0
```

put() in HashTableHybrid for small size can vary too much. 100 test is not enough for its result, but according to my tests generally its above 20000.

```
___HashTables Performance Test___

100 tests for 100 1000 10000 random datasets(same for each).
Processing.....................................................................................................
HashTableHybrid (size = 100)
put() = 20553
get() = 39
remove() = 69
size() = 1
isEmpty() = 1

BSTHashTableChain (size = 100)
put() = 1402
get() = 94
remove() = 66
size() = 1
isEmpty() = 2

HashTableHybrid (size = 1000)
put() = 4952
get() = 263
remove() = 296
size() = 1
isEmpty() = 0

BSTHashTableChain (size = 1000)
put() = 328
get() = 81
remove() = 111
size() = 1
isEmpty() = 1

HashTableHybrid (size = 10000)
put() = 10059
get() = 14654
remove() = 13791
size() = 0
isEmpty() = 0

BSTHashTableChain (size = 10000)
put() = 224
get() = 110
remove() = 93
size() = 0
isEmpty() = 0
```

For the HashTableHybrid:

1- size() and isEmpty() are constant time operations.
2- put() gives better results for greater sizes, but it can be varied because of the number of rehashing. If put is great, that means in that case rehashing happens a lot other option is collision happened a lot. Both possibility causes its increase.
3- get() result's are changes according to number of probes, but generally it is constant.
4- remove() is similar to get, it changes according to number of probes, but generally it is constant.
5- Rehashing() it happens much when the load factor is low, and it is expensive because of the prime number calculations. It directly affects put()'s time.

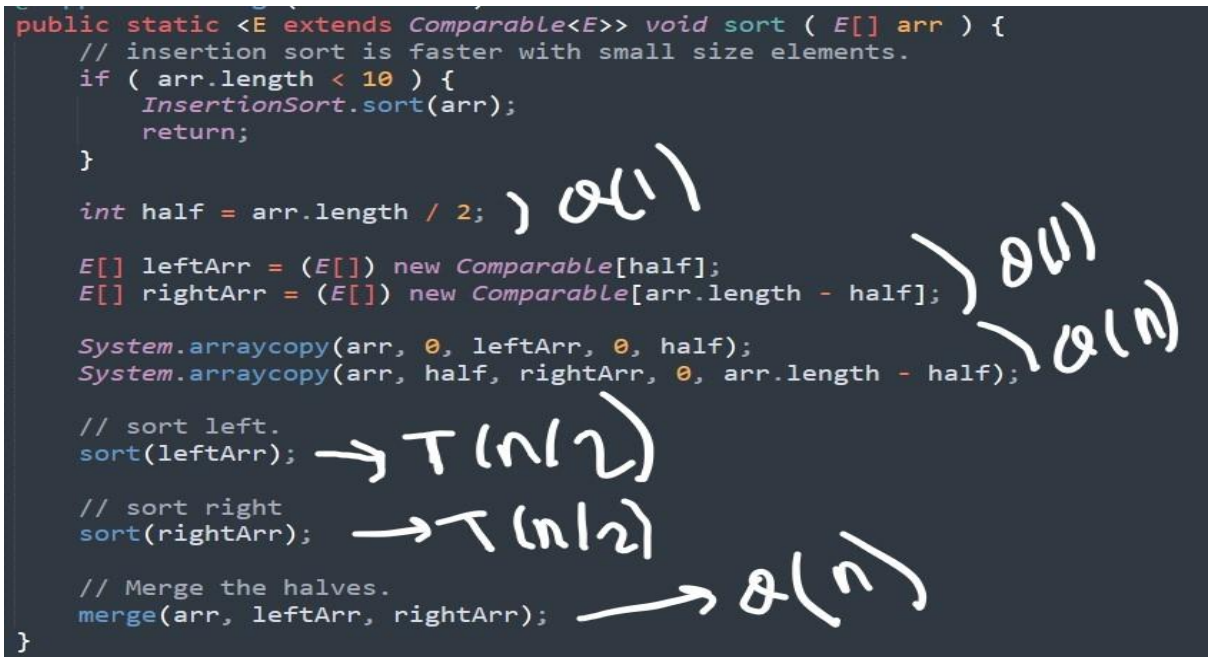
For the BSTHashTableChain:

1- size() and isEmpty() are constant time operations.
2- put() is constant averagely, but it gives better results for medium and large sizes. It could be said that, collision happened less than small size for these two. Collision and number of rehashing could increase its time. Also BinarySearchTree's add operation causes this situation, it has O(logn) time complexity for average cases. For the worst cases, it is quadratic, therefore worst case could happen for greater bucket, and this might increase the time of put method.
3- get() results is constant time for most cases.
4- remove() is very similar to get(), it is performed in constant time mostly. BST's remove operation is averagely O(logn), but it is thought in hash table constant averagely, and empirical results verifies theoretical assumption.
5- rehashing() is amortized constant time, it has not much difference than regular chaining implementation because of the iterator implementation of BinarySearchTree.

Q2

Theoretical Analysis:

MergeSort



Insertion sort is added for smaller size to optimize merge sort. It divides the array two sub arrays in each step, division and merging happens logn times, each of them have n copy. Therefore, complexity is Θ(nlogn).

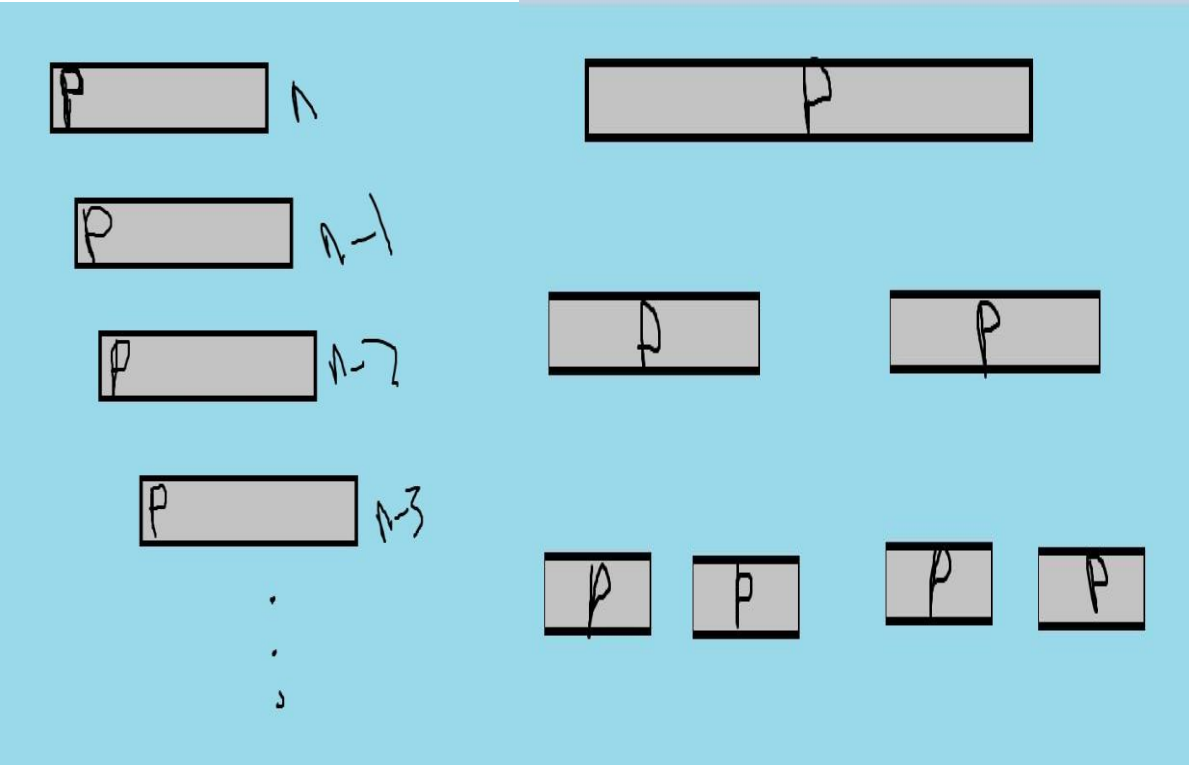
$T(n) = 2T(n/2) + \Theta(n)$ for $n > 1$

$T(n) = \Theta(1)$ for $n <= 1$

After solving this recurrence relation **$T(n) = \Theta(nlogn)$**

QuickSort

Worst case Best case



If partition choses greatest or smallest element as pivot, worst case occurs, and complexity become quadratic.

$T_b(n) = \Theta(n\log n)$, $T_w(n) = \Theta(n^2)$

$T_{av}(n) = O(n\log n)$

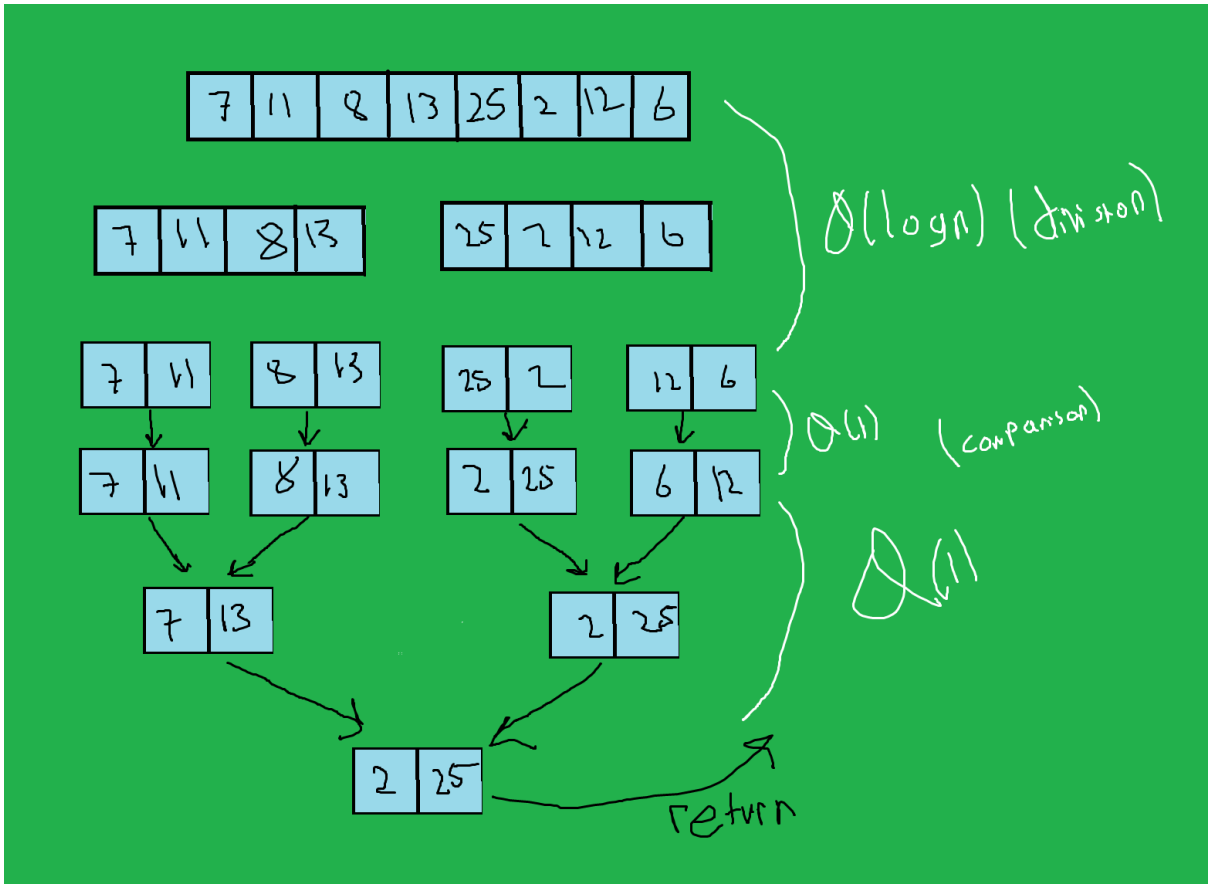
NewSort

Complexity is calculated with visualizing algorithms. Finding min and max takes logn times, but calling it from sort method for different arrays(smaller) increases its complexity. According to my calculations and empirical results it is even greater than quadratic. $\Theta(n^2\log n)$. Calculation and source code are at the other page.

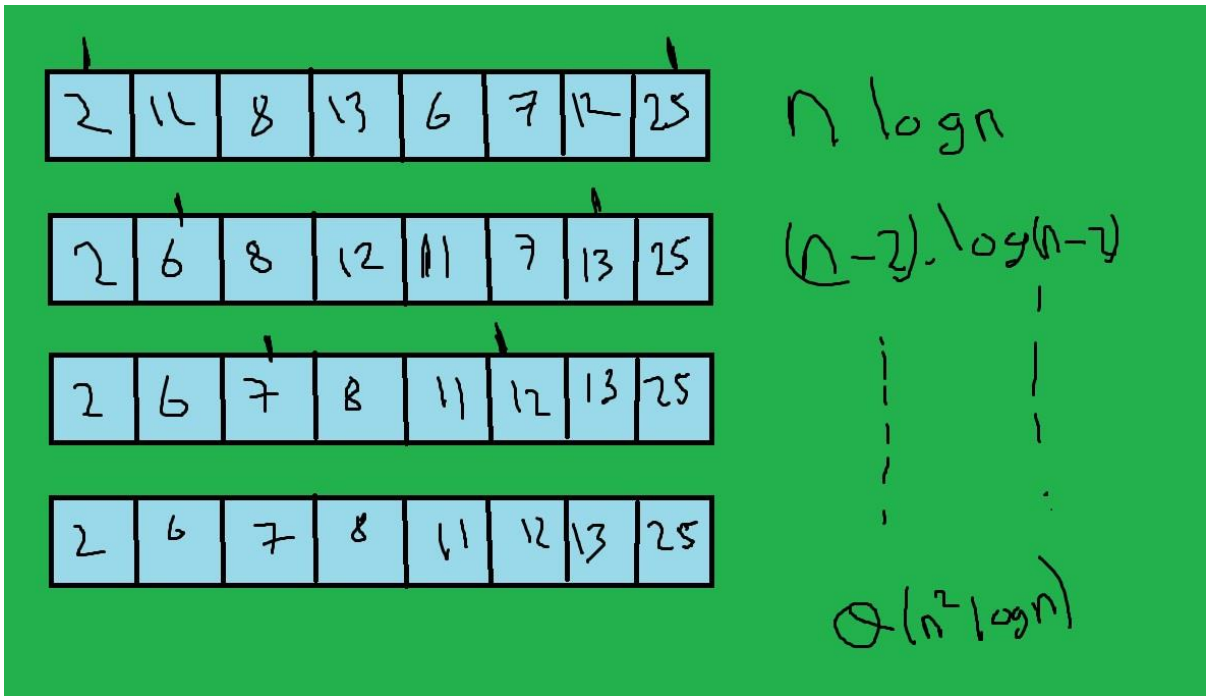
```java
private static <E extends Comparable<E>> void new_sort ( E[] arr, int head, int tail ) {
    if ( head > tail )
        return;

    else {
        int[] minMax = new int[2];

        minMax = minMaxFinder(arr, head, tail);
        swap(arr, head, minMax[0]);
        if ( minMax[1] == head ) {
            // assert : maximum is at minimum's previous place.
            swap(arr, tail, minMax[0] );
        }
        else
            swap(arr, tail, minMax[1]);

        // Invariant : arr[0, ..., head] and arr[tail, .., arr.length] are sorted.
        new_sort(arr, head + 1, tail - 1);
    }
}
```

```java
private static <E extends Comparable<E>> int[] minMaxFinder ( E[] arr, int head, int tail ) {
    if ( tail - head <= 1 ) {
        int[] minMax = new int[2];

        if ( arr[tail].compareTo(arr[head]) > 0 ) {
            minMax[0] = head;
            minMax[1] = tail;
            return minMax;
        }

        // assert : arr[head] >= arr[tail]
        minMax[1] = head;
        minMax[0] = tail;
        return minMax;
    }

    int[] minMax1 = new int[2];
    int[] minMax2 = new int[2];
    int medium = (head + tail) / 2;

    // go left and right
    minMax1 = minMaxFinder(arr, head, medium);
    minMax2 = minMaxFinder(arr, medium + 1, tail);

    // modify minMax1 only
    if ( arr[minMax1[0]].compareTo(arr[minMax2[0]]) > 0 )
        minMax1[0] = minMax2[0];

    if ( arr[minMax2[1]].compareTo(arr[minMax1[1]]) > 0 )
        minMax1[1] = minMax2[1];

    return minMax1;
}
```

Finding minimum and maximum value's indexes of an array.



Finding minimum and maximum is Θ(logn).

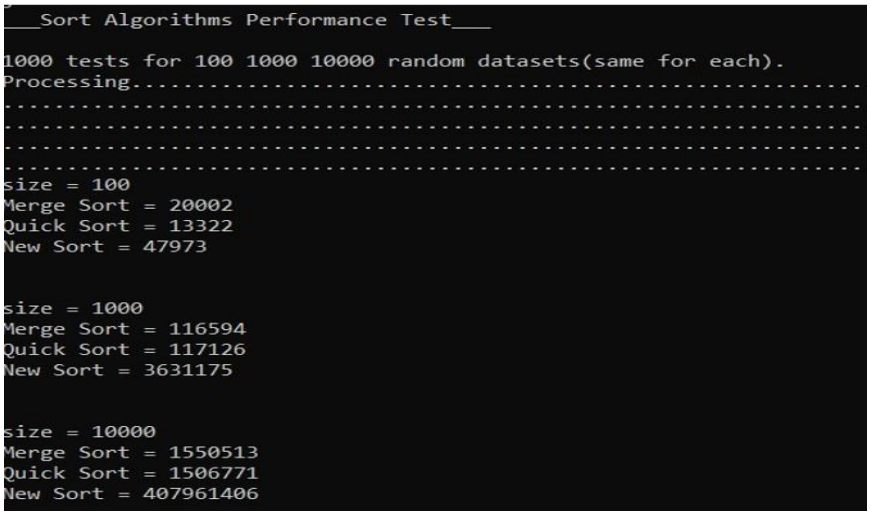


Sort algorithm is $\Theta(n^2 \log n)$.

So, time complexity of newSort is **$\Theta(n^2 \log n)$.**

Merge Sort	**$T(n) = \Theta(n\log n)$**
Quick Sort	**$T_b(n) = \Theta(n\log n)$** **$T_w(n) = \Theta(n^2)$** **$T_{av}(n) = O(n\log n)$**
New Sort	**$T(n) = \Theta(n^2\log n)$**

Empirical Analysis:



It can be seen that merge and quick sort is quite close. Their increasing is logarithmic, and theoretical expectations are satisfied. New sort is quite slow according to merge and quick sort. Increasing of new sort is greater than quadratic increasing. In theoretical analysis it is determined that, time complexity is $\Theta(n^2\log n)$. Empirical analysis does not confirm this %100 percent, but it can be seen that it is greater than quadratic. It could be more accurate to test this with another quadratic sort to see it clearly. Another result is quick sort gives better result for smaller size. For larger sizes merge and quick sort is quite close. Quick sort's worst case is quadratic, but we

cannot see any quadratic increasing in this test. So average case is confirmed with 1000 datasets.

5. TEST CASES

Testing BSTHashTableChain

```
public static void main ( String args[] ) {

    testBSTHashTableChain();
    testHashTableHybrid();
    testMergeSort();
    testQuickSort();
    testNewSort();
    //  testBinarySearchTree();
    //  performHashTables(100);
    //  performSortAlgorithms(1000);
}
```

Test cases are this, 3 of them is commented out perform methods are for performance test, they don't run when the program runs. If you want to check performance test, comment these out. But test might take long time. BinarySearchTree is also tested and its results, cases are same with previous homework. Therefore, I don't add its cases and results to this homework, but it can be tried just commenting out testBinarySearchTree method.

```
public static void testBSTHashTableChain ( ) {
    System.out.println("________Testing BSTHashTableChain________\n\n");

    System.out.println("Creating an empty BSTHashTableChain\n");
    KWHashMap<Integer, String> table1 = new BSTHashTableChain<Integer, String>();
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("table1.isEmpty() = " + table1.isEmpty() + "\n");

    System.out.println("put (6, burak)");
    table1.put(6, "burak");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (126, james)");
    table1.put(126, "james");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (11, elif)");
    table1.put(11, "elif");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (16, claire)");
    table1.put(16, "claire");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (421, josh)");
    table1.put(421, "josh");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (6, john) (trying to modify key 6)");
    table1.put(6, "john");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

    System.out.println("put (421, cassie) (trying to modify key 421)");
    table1.put(421, "cassie");
    System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");
```

```
System.out.println("after removing key 16");
table1.remove(16);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 11");
table1.remove(11);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 421");
table1.remove(421);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 19(for testing error)");
table1.remove(19);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 126");
table1.remove(126);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 6");
table1.remove(6);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");

System.out.println("after removing key 15(for testing error)");
table1.remove(15);
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");
System.out.println("table1.isEmpty() = " + table1.isEmpty() + "\n");
```

```
System.out.println("put each element again and add aditional elements.");
table1.put(82, "lain");
table1.put(6, "burak");
table1.put(126, "james");
table1.put(119, "john");
table1.put(11, "elif");
table1.put(16, "claire");
table1.put(421, "josh");
table1.put(47, "osman");
table1.put(731, "mike");
table1.put(522, "ryan");
table1.put(413, "michelle");
table1.put(234, "ayse");
table1.put(632, "ahmet");
table1.put(13, "donald");
table1.put(539, "will");
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");
```

```
table1.put(512, "maisie");
table1.put(411, "hulya");
table1.put(231, "chris");
table1.put(852, "jada");
table1.put(609, "emin");
table1.put(382, "esra");
table1.put(786, "kerem");
table1.put(742, "tony");
table1.put(191, "shiv");
table1.put(5, "suzan");
table1.put(31, "owen");
table1.put(75, "kim");
table1.put(832, "ayca");
table1.put(741, "bora");
table1.put(235, "bob");
table1.put(919, "caine");
table1.put(213, "sevgi");
table1.put(561, "mert");
table1.put(901, "roy");
table1.put(1, "asli");
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");
System.out.println("Rehashing happened, and size of the table is increased.\n");

System.out.println("table1.get(75) = " + table1.get(75));
System.out.println("table1.get(5) = " + table1.get(5));
System.out.println("table1.get(16) = " + table1.get(16));
System.out.println("table1.get(2)(does not exist) = " + table1.get(2));
System.out.println("table1.get(3)(does not exist) = " + table1.get(3));
System.out.println("table1.get(11) = " + table1.get(11) + "\n");

System.out.println("put (539, jack) (trying to modify key 539)");
table1.put(539, "jack");
System.out.println(table1 + "--------------------------- size = " + table1.size() + "\n");
```

Testing HashTableHybrid

```java
public static void testHashTableHybrid ( ) {
    System.out.println("________Testing HashTableHybrid________\n\n");

    System.out.println("Creating an empty HashTableHybrid\n");
    KWHashMap<Integer, String> hybrid1 = new HashTableHybrid<Integer, String>();
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("hybrid1.isEmpty() = " + hybrid1.isEmpty() + "\n");

    System.out.println("put (6, burak)");
    hybrid1.put(6, "burak");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("put (126, james)");
    hybrid1.put(126, "james");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("put (11, elif)");
    hybrid1.put(11, "elif");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("put (742, tony)");
    hybrid1.put(742, "tony");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("after removing key 126");
    hybrid1.remove(126);
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("put (203, claire)");
    hybrid1.put(203, "claire");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

    System.out.println("put (126, james)");
    hybrid1.put(126, "james");
    System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 742");
hybrid1.remove(742);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (421, josh)");
hybrid1.put(421, "josh");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (75, kim)");
hybrid1.put(75, "kim");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (235, bob)");
hybrid1.put(235, "bob");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 235");
hybrid1.remove(235);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (852, jada)");
hybrid1.put(852, "jada");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (5, susan)");
hybrid1.put(5, "susan");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 852");
hybrid1.remove(852);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (852, jada)");
hybrid1.put(852, "jada");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");
```

```
System.out.println("put (539, will)");
hybrid1.put(539, "will");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (235, bob)");
hybrid1.put(235, "bob");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (6, john) (trying to modify key 6)");
hybrid1.put(6, "john");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("put (421, cassie) (trying to modify key 421)");
hybrid1.put(421, "cassie");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 6");
hybrid1.remove(6);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 11");
hybrid1.remove(11);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 421");
hybrid1.remove(421);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 19(for testing error)");
hybrid1.remove(19);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("hybrid1.isEmpty() = " + hybrid1.isEmpty() + "\n");
 System.out.println("put additional elements.");
 hybrid1.put(82, "lain");
 hybrid1.put(6, "burak");
 hybrid1.put(126, "james");
 hybrid1.put(119, "john");
 hybrid1.put(11, "elif");
 hybrid1.put(16, "claire");
 hybrid1.put(421, "josh");
 hybrid1.put(47, "osman");
 hybrid1.put(731, "mike");
 hybrid1.put(522, "ryan");
 hybrid1.put(413, "michelle");
 hybrid1.put(234, "ayse");
 hybrid1.put(632, "ahmet");
 hybrid1.put(13, "donald");

 System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

 System.out.println("Put more elements\n");

 hybrid1.put(512, "maisie");
 hybrid1.put(411, "hulya");
 hybrid1.put(231, "chris");
 hybrid1.put(609, "emin");
 hybrid1.put(382, "esra");
 hybrid1.put(786, "kerem");
 hybrid1.put(191, "shiv");
 hybrid1.put(31, "owen");
 hybrid1.put(832, "ayca");
 hybrid1.put(741, "bora");
 hybrid1.put(919, "caine");
 hybrid1.put(213, "sevgi");
 hybrid1.put(561, "mert");
 hybrid1.put(901, "roy");
 hybrid1.put(1, "asli");
 System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");
```

```
System.out.println("Rehashing happened, and size of the table is increased.\n");

System.out.println("hybrid1.get(75) = " + hybrid1.get(75));
System.out.println("hybrid1.get(5) = " + hybrid1.get(5));
System.out.println("hybrid1.get(16) = " + hybrid1.get(16));
System.out.println("hybrid1.get(2)(does not exist) = " + hybrid1.get(2));
System.out.println("hybrid1.get(3)(does not exist) = " + hybrid1.get(3));
System.out.println("hybrid1.get(11) = " + hybrid1.get(11) + "\n");

System.out.println("put (539, jack) (trying to modify key 539)");
hybrid1.put(539, "jack");
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("after removing key 13");
hybrid1.remove(13);
System.out.println(hybrid1 + "--------------------------- size = " + hybrid1.size() + "\n");

System.out.println("hybrid1.get(512) = " + hybrid1.get(512));
System.out.println("hybrid1.get(539) = " + hybrid1.get(539));
System.out.println("hybrid1.get(213) = " + hybrid1.get(213));
System.out.println("hybrid1.get(13)(does not exist) = " + hybrid1.get(13));
System.out.println("hybrid1.get(901) = " + hybrid1.get(901) + "\n");
```

Testing MergeSort

```
public static void testMergeSort ( ) {

    System.out.println("\n___Testing MergeSort___");

    System.out.println("\nTest for 50 sized array.(randomly generated)\n");
    Integer[] arr1 = new Integer[50];
    Random rand = new Random();
    for ( int i = 0; i < 50; ++i ) {
        arr1[i] = rand.nextInt(100) + 1;
    }

    System.out.println("Before Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");

    MergeSort.sort(arr1);
    System.out.println("After Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");

    System.out.println("\nTest for 25 sized array.(randomly generated)\n");
    Integer[] arr2 = new Integer[25];
    for ( int i = 0; i < 25; ++i ) {
        arr2[i] = rand.nextInt(100) + 1;
    }
```

```
MergeSort.sort(arr2);
System.out.println("After Sorting");
for ( int i = 0; i < arr2.length; ++i ) {
    System.out.print(arr2[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for 5 sized array.(randomly generated)\n");
Integer[] arr3 = new Integer[5];
for ( int i = 0; i < 5; ++i ) {
    arr3[i] = rand.nextInt(100) + 1;
}

System.out.println("Before Sorting");
for ( int i = 0; i < arr3.length; ++i ) {
    System.out.print(arr3[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr3);
System.out.println("After Sorting");
for ( int i = 0; i < arr3.length; ++i ) {
    System.out.print(arr3[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for 100 sized array.(randomly generated)\n");
Integer[] arr4 = new Integer[100];
for ( int i = 0; i < 100; ++i ) {
    arr4[i] = rand.nextInt(100) + 1;
}
System.out.println("Before Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr4);
System.out.println("After Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");
System.out.println("\nTest for 1 sized array.\n");
Integer[] arr5 = new Integer[1];
arr5[0] = 3;

System.out.println("Before Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr5);

System.out.println("After Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");
```

```
System.out.println("\nTest for sorted array.\n");
Integer[] arr6 = new Integer[13];
for ( int i = 0; i < 13; ++i )
    arr6[i] = i + 1;

System.out.println("Before Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr6);

System.out.println("After Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for inversely sorted array.\n");
Integer[] arr7 = new Integer[10];
for ( int i = 0; i < arr7.length; ++i )
    arr7[i] = arr7.length - i;

System.out.println("Before Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr7);

System.out.println("After Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");
```

```
System.out.println("\nTest for an array which has same elements.");
Integer[] arr8 = new Integer[10];
for ( int i = 0; i < 10; i++ )
    arr8[i] = 5;

System.out.println("Before Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");

MergeSort.sort(arr8);

System.out.println("After Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");
```

Testing QuickSort

```
public static void testQuickSort ( ) {
    System.out.println("\n___Testing QuickSort___");

    System.out.println("\nTest for 50 sized array.(randomly generated)\n");
    Integer[] arr1 = new Integer[50];
    Random rand = new Random();
    for ( int i = 0; i < 50; ++i ) {
        arr1[i] = rand.nextInt(100) + 1;
    }


    System.out.println("Before Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");

    QuickSort.sort(arr1);
    System.out.println("After Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");
```

```
System.out.println("\nTest for 25 sized arrays(randomly generated).\n");
Integer[] arr2 = new Integer[25];
for ( int i = 0; i < 25; ++i ) {
    arr2[i] = rand.nextInt(100) + 1;
}

System.out.println("Before Sorting");
for ( int i = 0; i < arr2.length; ++i ) {
    System.out.print(arr2[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr2);
System.out.println("After Sorting");
for ( int i = 0; i < arr2.length; ++i ) {
    System.out.print(arr2[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for 5 sized array.(randomly generated)\n");
Integer[] arr3 = new Integer[5];
for ( int i = 0; i < 5; ++i ) {
    arr3[i] = rand.nextInt(100) + 1;
}

System.out.println("Before Sorting");
for ( int i = 0; i < arr3.length; ++i ) {
    System.out.print(arr3[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr3);
System.out.println("After Sorting");
for ( int i = 0; i < arr3.length; ++i ) {
    System.out.print(arr3[i] + " ");
}
System.out.print("\n");
```

```
System.out.println("\nTest for 100 sized array.(randomly generated)\n");
Integer[] arr4 = new Integer[100];
for ( int i = 0; i < 100; ++i ) {
    arr4[i] = rand.nextInt(100) + 1;
}

System.out.println("Before Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr4);
System.out.println("After Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for 1 sized array.\n");
Integer[] arr5 = new Integer[1];
arr5[0] = 3;

System.out.println("Before Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr5);

System.out.println("After Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for sorted array.\n");
Integer[] arr6 = new Integer[13];
for ( int i = 0; i < 13; ++i )
    arr6[i] = i + 1;

System.out.println("Before Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr6);

System.out.println("After Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for inversely sorted array.\n");
Integer[] arr7 = new Integer[10];
for ( int i = 0; i < arr7.length; ++i )
    arr7[i] = arr7.length - i;

System.out.println("Before Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr7);

System.out.println("After Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");
```

```
System.out.println("\nTest for an array which has same elements.");
Integer[] arr8 = new Integer[10];
for ( int i = 0; i < 10; i++ )
    arr8[i] = 5;

System.out.println("Before Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");

QuickSort.sort(arr8);

System.out.println("After Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");
```

Testing NewSort

```
public static void testNewSort ( ) {

    System.out.println("\n___Testing NewSort___");

    System.out.println("\nTest for 50 sized array.(randomly generated)\n");
    Integer[] arr1 = new Integer[50];
    Random rand = new Random();
    for ( int i = 0; i < 50; ++i ) {
        arr1[i] = rand.nextInt(100) + 1;
    }


    System.out.println("Before Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");

    NewSort.sort(arr1);
    System.out.println("After Sorting");
    for ( int i = 0; i < arr1.length; ++i ) {
        System.out.print(arr1[i] + " ");
    }
    System.out.print("\n");

    System.out.println("\nTest for 25 sized array.(randomly generated)\n");
    Integer[] arr2 = new Integer[25];
    for ( int i = 0; i < 25; ++i ) {
        arr2[i] = rand.nextInt(100) + 1;
    }
```

```
  System.out.println("Before Sorting");
  for ( int i = 0; i < arr2.length; ++i ) {
      System.out.print(arr2[i] + " ");
  }
  System.out.print("\n");

  NewSort.sort(arr2);
  System.out.println("After Sorting");
  for ( int i = 0; i < arr2.length; ++i ) {
      System.out.print(arr2[i] + " ");
  }
  System.out.print("\n");

  System.out.println("\nTest for 5 sized array.(randomly generated)\n");
  Integer[] arr3 = new Integer[5];
  for ( int i = 0; i < 5; ++i ) {
      arr3[i] = rand.nextInt(100) + 1;
  }

  System.out.println("Before Sorting");
  for ( int i = 0; i < arr3.length; ++i ) {
      System.out.print(arr3[i] + " ");
  }
  System.out.print("\n");

  NewSort.sort(arr3);
  System.out.println("After Sorting");
  for ( int i = 0; i < arr3.length; ++i ) {
      System.out.print(arr3[i] + " ");
  }
  System.out.print("\n");
```

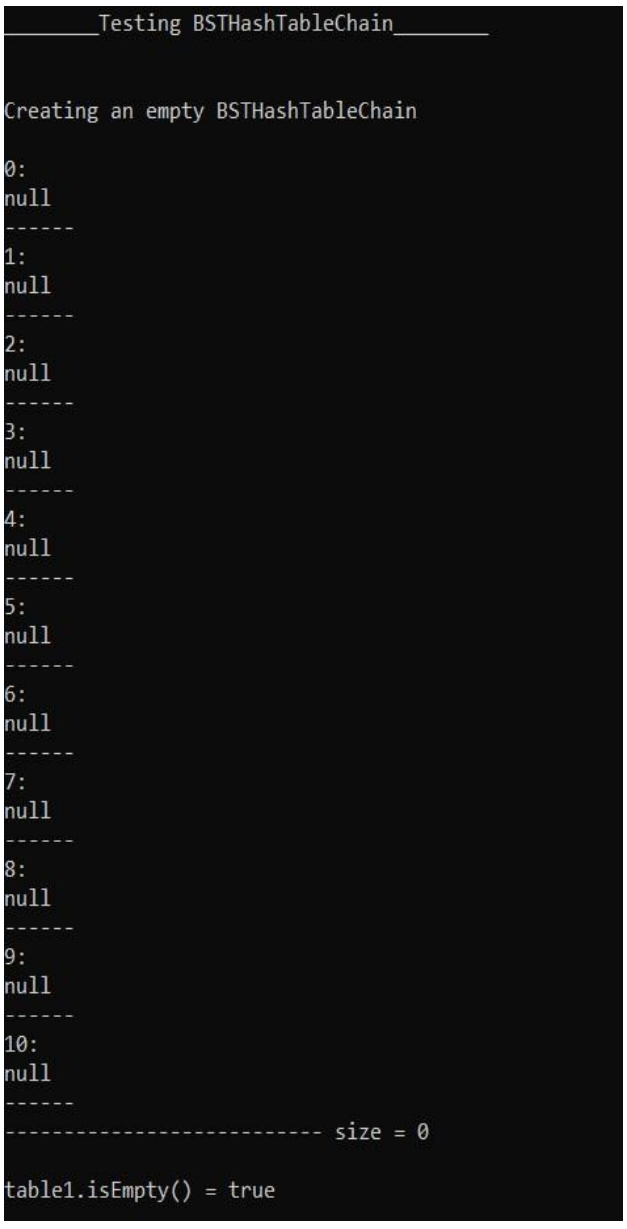
```
System.out.println("\nTest for 100 sized array.(randomly generated)\n");
Integer[] arr4 = new Integer[100];
for ( int i = 0; i < 100; ++i ) {
    arr4[i] = rand.nextInt(100) + 1;
}

System.out.println("Before Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");

NewSort.sort(arr4);
System.out.println("After Sorting");
for ( int i = 0; i < arr4.length; ++i ) {
    System.out.print(arr4[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for 1 sized array.\n");
Integer[] arr5 = new Integer[1];
arr5[0] = 3;

System.out.println("Before Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");

NewSort.sort(arr5);

System.out.println("After Sorting");
for ( int i = 0; i < arr5.length; ++i ) {
    System.out.print(arr5[i] + " ");
}
System.out.print("\n");
```

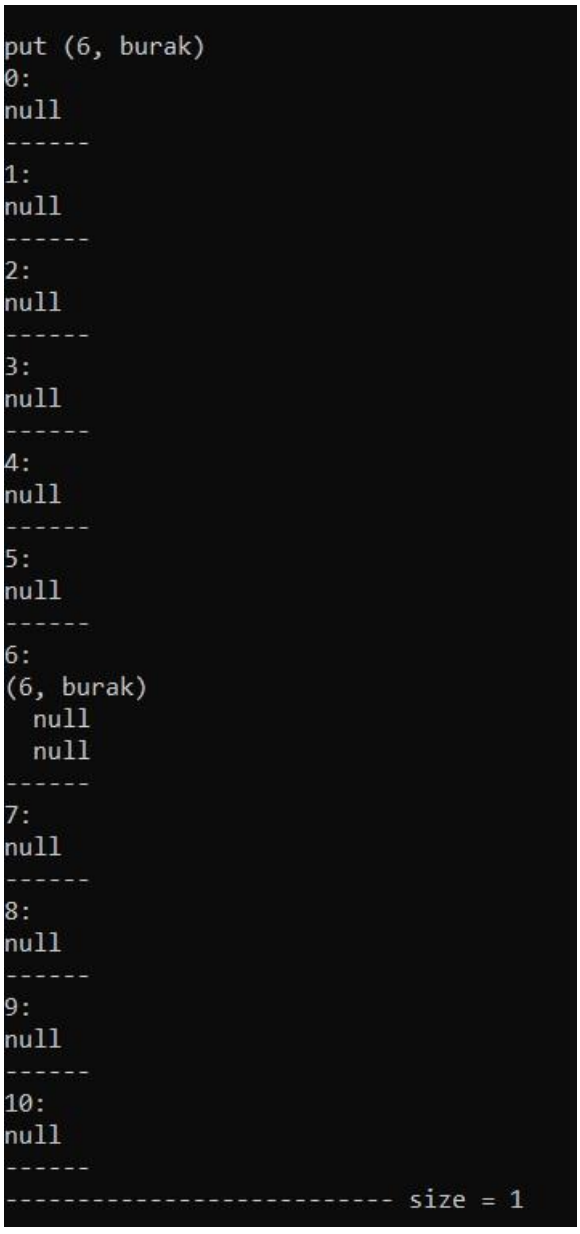
```
System.out.println("\nTest for sorted array.\n");
Integer[] arr6 = new Integer[13];
for ( int i = 0; i < 13; ++i )
    arr6[i] = i + 1;

System.out.println("Before Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

NewSort.sort(arr6);

System.out.println("After Sorting");
for ( int i = 0; i < arr6.length; ++i ) {
    System.out.print(arr6[i] + " ");
}
System.out.print("\n");

System.out.println("\nTest for inversely sorted array.\n");
Integer[] arr7 = new Integer[10];
for ( int i = 0; i < arr7.length; ++i )
    arr7[i] = arr7.length - i;

System.out.println("Before Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");

NewSort.sort(arr7);

System.out.println("After Sorting");
for ( int i = 0; i < arr7.length; ++i ) {
    System.out.print(arr7[i] + " ");
}
System.out.print("\n");
```

```
System.out.println("\nTest for an array which has same elements.");
Integer[] arr8 = new Integer[10];
for ( int i = 0; i < 10; i++ )
    arr8[i] = 5;

System.out.println("Before Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");

NewSort.sort(arr8);

System.out.println("After Sorting");
for ( int i = 0; i < arr8.length; ++i ) {
    System.out.print(arr8[i] + " ");
}
System.out.print("\n");
```

6. RUNNING AND RESULTS

BSTHashTableChain Results





```
put (126, james)
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, burak)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 2
```

```
put (11, elif)
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, burak)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 3
```

```
put (16, claire)
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
(126, james)
  (16, claire)
    null
    null
  null
------
6:
(6, burak)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 4
```

```
put (421, josh)
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
(421, josh)
  null
  null
------
4:
null
------
5:
(126, james)
  (16, claire)
    null
    null
  null
------
6:
(6, burak)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 5
```

```
put (6, john) (trying to modify key 6)
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
(421, josh)
  null
  null
------
4:
null
------
5:
(126, james)
  (16, claire)
    null
    null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 5
```

```
put (421, cassie) (trying to modify key 421)
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
(421, cassie)
  null
  null
------
4:
null
------
5:
(126, james)
  (16, claire)
    null
    null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 5
```

```
after removing key 16
0:
(11, elif)
  null
  null
------
1:
null
------
2:
null
------
3:
(421, cassie)
  null
  null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 4
```

```
after removing key 11
0:
null
------
1:
null
------
2:
null
------
3:
(421, cassie)
  null
  null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 3
```

```
after removing key 421
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 2
```

```
after removing key 19(for testing error)
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
(126, james)
  null
  null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 2
```

```
after removing key 126
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
null
------
6:
(6, john)
  null
  null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
-------------------------- size = 1
```

```
after removing key 6
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
null
------
6:
null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 0
```

```
after removing key 15(for testing error)
0:
null
------
1:
null
------
2:
null
------
3:
null
------
4:
null
------
5:
null
------
6:
null
------
7:
null
------
8:
null
------
9:
null
------
10:
null
------
--------------------------- size = 0

table1.isEmpty() = true
```

```
put each element again and add aditional
0:
(11, elif)
  null
  (539, will)
    null
    null
------
1:
null
------
2:
(13, donald)
  null
  null
------
3:
(421, josh)
  (47, osman)
    null
    (234, ayse)
      null
      null
  null
------
4:
null
------
5:
(82, lain)
  (16, claire)
    null
    null
  (126, james)
    null
    (731, mike)
      (522, ryan)
        null
        (632, ahmet)
          null
          null
      null
------
6:
(6, burak)
```

```
0:
null
------
1:
(231, chris)
  (47, osman)
    (1, asli)
      null
      null
    null
  (852, jada)
    null
    null
------
2:
null
------
3:
null
------
4:
(234, ayse)
  (119, john)
    null
    null
  (786, kerem)
    null
    (832, ayca)
      null
      (901, roy)
        null
        null
------
5:
(235, bob)
  (5, suzan)
    null
    null
  (741, bora)
    null
    null
------
6:
(213, sevgi)
  (6, burak)
  null
  (413, michelle)
    null
    null
------
7:
null
------
8:
null
------
9:
(119, john)
  null
  null
------
10:
null
------
--------------------------- size = 15
```

```
  (6, burak)
    null
    (75, kim)
      null
      null
  (742, tony)
    (512, maisie)
      null
      null
    null
------
7:
(421, josh)
  (191, shiv)
    null
    null
  null
------
8:
(31, owen)
  null
  null
------
9:
(561, mert)
  null
  null
------
10:
(539, will)
  null
  null
------
11:
(11, elif)
  null
  (609, emin)
    (126, james)
      null
      null
    (632, ahmet)
      null
      null
------
12:
null
------
```

```
------
13:
(13, donald)
  null
  (82, lain)
    null
    null
------
14:
(382, esra)
  null
  null
------
15:
null
------
16:
(16, claire)
  null
  (522, ryan)
    null
    null
------
17:
null
------
18:
(731, mike)
  null
  null
------
19:
null
------
20:
(411, hulya)
  null
  null
------
21:
null
------
22:
(413, michelle)
  null
  (919, caine)
    null
    null
------
```

```
------
---------------------------- size = 35

Rehashing happened, and size of the table is increased.

table1.get(75) = kim
table1.get(5) = suzan
table1.get(16) = claire
table1.get(2)(does not exist) = null
table1.get(3)(does not exist) = null
table1.get(11) = elif
```

Results Of HashTableHybrid

```
_______Testing HashTableHybrid_______


Creating an empty HashTableHybrid

0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: null
8: null
9: null
10: null
--------------------------- size = 0

hybrid1.isEmpty() = true

put (6, burak)
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: null
9: null
10: null
--------------------------- size = 1

put (126, james)
0: null
1: (key = 126, value = james, next key = null)
2: null
3: null
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: null
9: null
10: null
--------------------------- size = 2
```

```
put (11, elif)
0: null
1: (key = 126, value = james, next key = null)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: null
9: null
10: null
--------------------------- size = 3

put (742, tony)
0: null
1: (key = 126, value = james, next key = 742)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: (key = 742, value = tony, next key = null)
9: null
10: null
--------------------------- size = 4

after removing key 126
0: null
1: (key = 742, value = tony, next key = null)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: null
9: null
10: null
--------------------------- size = 3
```

```
put (203, claire)
0: null
1: (key = 742, value = tony, next key = 203)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: (key = 203, value = claire, next key = null)
9: null
10: null
-------------------------- size = 4

put (126, james)
0: null
1: (key = 742, value = tony, next key = 203)
2: null
3: (key = 11, value = elif, next key = null)
4: (key = 126, value = james, next key = null)
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: (key = 203, value = claire, next key = 126)
9: null
10: null
-------------------------- size = 5

after removing key 742
0: null
1: (key = 126, value = james, next key = 203)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: (key = 203, value = claire, next key = null)
9: null
10: null
-------------------------- size = 4
```

```
put (421, josh)
0: null
1: (key = 126, value = james, next key = 203)
2: null
3: (key = 11, value = elif, next key = null)
4: null
5: null
6: null
7: (key = 6, value = burak, next key = null)
8: (key = 203, value = claire, next key = null)
9: (key = 421, value = josh, next key = null)
10: null
-------------------------- size = 5

put (75, kim)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = null)
6: null
7: null
8: null
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = null)
17: (key = 11, value = elif, next key = 6)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 6
```

```
put (235, bob)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = null)
6: null
7: null
8: (key = 235, value = bob, next key = null)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = null)
17: (key = 11, value = elif, next key = 6)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
------------------------- size = 7

after removing key 235
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = null)
6: null
7: null
8: null
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = null)
17: (key = 11, value = elif, next key = 6)
18: null
19: null
```

```
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
------------------------- size = 6

put (852, jada)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = null)
6: null
7: null
8: (key = 852, value = jada, next key = null)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
------------------------- size = 7
```

```
put (5, susan)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = null)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 8

after removing key 852
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = 5)
6: null
7: null
8: null
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = null)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
```

```
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 7

put (852, jada)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = null)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: null
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 8
```

```
put (539, will)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = null)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: null
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 9

put (235, bob)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = burak, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
```

```
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 10

put (6, john) (trying to modify key 6)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = john, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: (key = 421, value = josh, next key = null)
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
```

```
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 10

put (421, cassie) (trying to modify key 421)
0: null
1: null
2: null
3: null
4: null
5: (key = 6, value = john, next key = 5)
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: (key = 421, value = cassie, next key = null)
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 6)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 10
```

```
after removing key 6
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: (key = 421, value = cassie, next key = null)
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 11, value = elif, next key = 5)
18: (key = 5, value = susan, next key = null)
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
-------------------------- size = 9

after removing key 11
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: (key = 421, value = cassie, next key = null)
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 5, value = susan, next key = null)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
```

```
21: (key = 126, value = james, next key = null)
22: null
--------------------------- size = 8

after removing key 421
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: null
12: null
13: null
14: (key = 235, value = bob, next key = null)
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 5, value = susan, next key = null)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
--------------------------- size = 7

after removing key 19(for testing error)
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: null
8: (key = 852, value = jada, next key = 235)
9: null
10: null
11: null
12: null
13: null
14: (key = 235, value = bob, next key = null)
```

```
15: (key = 539, value = will, next key = null)
16: (key = 75, value = kim, next key = 852)
17: (key = 5, value = susan, next key = null)
18: null
19: null
20: (key = 203, value = claire, next key = null)
21: (key = 126, value = james, next key = null)
22: null
--------------------------- size = 7

hybrid1.isEmpty() = false

put additional elements.
0: null
1: null
2: (key = 234, value = ayse, next key = null)
3: null
4: null
5: (key = 413, value = michelle, next key = null)
6: (key = 75, value = kim, next key = null)
7: (key = 119, value = john, next key = 126)
8: (key = 632, value = ahmet, next key = null)
9: null
10: (key = 522, value = ryan, next key = null)
11: null
12: null
13: null
14: (key = 47, value = osman, next key = 13)
15: null
16: (key = 11, value = elif, next key = 5)
17: (key = 82, value = lain, next key = 75)
18: null
19: null
20: null
21: (key = 421, value = josh, next key = 413)
22: null
23: null
24: (key = 235, value = bob, next key = 234)
25: null
26: null
27: (key = 16, value = claire, next key = 47)
28: null
29: (key = 126, value = james, next key = null)
30: null
31: null
32: null
33: null
34: (key = 203, value = claire, next key = null)
```

```
34: (key = 203, value = claire, next key = null)
35: (key = 731, value = mike, next key = null)
36: null
37: (key = 6, value = burak, next key = 11)
38: (key = 539, value = will, next key = 522)
39: (key = 5, value = susan, next key = 16)
40: (key = 13, value = donald, next key = null)
41: null
42: (key = 852, value = jada, next key = null)
43: null
44: null
45: null
46: null
-------------------------- size = 20

Put more elements

0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: (key = 234, value = ayse, next key = 235)
8: null
9: (key = 539, value = will, next key = 561)
10: null
11: null
12: null
13: null
14: (key = 512, value = maisie, next key = 522)
15: null
16: (key = 126, value = james, next key = null)
17: (key = 411, value = hulya, next key = 382)
18: null
19: null
20: null
21: (key = 852, value = jada, next key = 832)
22: null
23: null
24: null
25: null
26: null
27: null
28: null
29: (key = 203, value = claire, next key = 191)
30: null
31: null
```

```
31: null
32: null
33: null
34: (key = 231, value = chris, next key = null)
35: null
36: (key = 382, value = esra, next key = null)
37: null
38: null
39: null
40: (key = 11, value = elif, next key = 16)
41: (key = 213, value = sevgi, next key = null)
42: null
43: (key = 731, value = mike, next key = 786)
44: null
45: null
46: null
47: (key = 901, value = roy, next key = null)
48: null
49: null
50: null
51: (key = 75, value = kim, next key = 119)
52: null
53: (key = 13, value = donald, next key = 31)
54: null
55: null
56: (key = 6, value = burak, next key = 5)
57: (key = 191, value = shiv, next key = 213)
58: (key = 413, value = michelle, next key = 421)
59: null
60: (key = 5, value = susan, next key = 13)
61: null
62: null
63: null
64: (key = 235, value = bob, next key = 231)
65: (key = 832, value = ayca, next key = null)
66: null
67: null
68: null
69: null
70: null
71: null
72: (key = 741, value = bora, next key = null)
73: (key = 47, value = osman, next key = 11)
74: null
75: (key = 421, value = josh, next key = 411)
76: (key = 522, value = ryan, next key = 539)
77: (key = 786, value = kerem, next key = 741)
78: (key = 119, value = john, next key = 82)
79: (key = 1, value = asli, next key = null)
```

```
79: (key = 1, value = asli, next key = null)
80: null
81: null
82: (key = 31, value = owen, next key = 1)
83: (key = 561, value = mert, next key = null)
84: (key = 82, value = lain, next key = 126)
85: null
86: null
87: (key = 632, value = ahmet, next key = 609)
88: (key = 609, value = emin, next key = null)
89: null
90: null
91: null
92: (key = 16, value = claire, next key = 6)
93: null
94: (key = 919, value = caine, next key = 901)
--------------------------- size = 35

Rehashing happened, and size of the table is increased.

hybrid1.get(75) = kim
hybrid1.get(5) = susan
hybrid1.get(16) = claire
hybrid1.get(2)(does not exist) = null
hybrid1.get(3)(does not exist) = null
hybrid1.get(11) = elif
```

```
put (539, jack) (trying to modify key 539)
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: (key = 234, value = ayse, next key = 235)
8: null
9: (key = 539, value = jack, next key = 561)
10: null
11: null
12: null
13: null
14: (key = 512, value = maisie, next key = 522)
15: null
16: (key = 126, value = james, next key = null)
17: (key = 411, value = hulya, next key = 382)
18: null
19: null
20: null
21: (key = 852, value = jada, next key = 832)
22: null
23: null
24: null
25: null
26: null
27: null
28: null
29: (key = 203, value = claire, next key = 191)
30: null
31: null
32: null
33: null
34: (key = 231, value = chris, next key = null)
35: null
36: (key = 382, value = esra, next key = null)
37: null
```

```
after removing key 13
0: null
1: null
2: null
3: null
4: null
5: null
6: null
7: (key = 234, value = ayse, next key = 235)
8: null
9: (key = 539, value = jack, next key = 561)
10: null
11: null
12: null
13: null
14: (key = 512, value = maisie, next key = 522)
15: null
16: (key = 126, value = james, next key = null)
17: (key = 411, value = hulya, next key = 382)
18: null
19: null
20: null
21: (key = 852, value = jada, next key = 832)
22: null
23: null
24: null
25: null
26: null
27: null
28: null
29: (key = 203, value = claire, next key = 191)
30: null
31: null
32: null
33: null
34: (key = 231, value = chris, next key = null)
35: null
36: (key = 382, value = esra, next key = null)
37: null
38: null
39: null
40: (key = 11, value = elif, next key = 16)
41: (key = 213, value = sevgi, next key = null)
42: null
43: (key = 731, value = mike, next key = 786)
44: null
45: null
46: null
47: (key = 901, value = roy, next key = null)
```

```
48: null
49: null
50: null
51: (key = 75, value = kim, next key = 119)
52: null
53: null
54: null
55: null
56: (key = 6, value = burak, next key = 5)
57: (key = 191, value = shiv, next key = 213)
58: (key = 413, value = michelle, next key = 421)
59: null
60: (key = 5, value = susan, next key = 31)
61: null
62: null
63: null
64: (key = 235, value = bob, next key = 231)
65: (key = 832, value = ayca, next key = null)
66: null
67: null
68: null
69: null
70: null
71: null
72: (key = 741, value = bora, next key = null)
73: (key = 47, value = osman, next key = 11)
74: null
75: (key = 421, value = josh, next key = 411)
76: (key = 522, value = ryan, next key = 539)
77: (key = 786, value = kerem, next key = 741)
78: (key = 119, value = john, next key = 82)
79: (key = 1, value = asli, next key = null)
80: null
81: null
82: (key = 31, value = owen, next key = 1)
83: (key = 561, value = mert, next key = null)
84: (key = 82, value = lain, next key = 126)
85: null
86: null
87: (key = 632, value = ahmet, next key = 609)
88: (key = 609, value = emin, next key = null)
89: null
90: null
91: null
92: (key = 16, value = claire, next key = 6)
93: null
94: (key = 919, value = caine, next key = 901)
-------------------------- size = 34
```

```
hybrid1.get(512) = maisie
hybrid1.get(539) = jack
hybrid1.get(213) = sevgi
hybrid1.get(13)(does not exist) = null
hybrid1.get(901) = roy
```

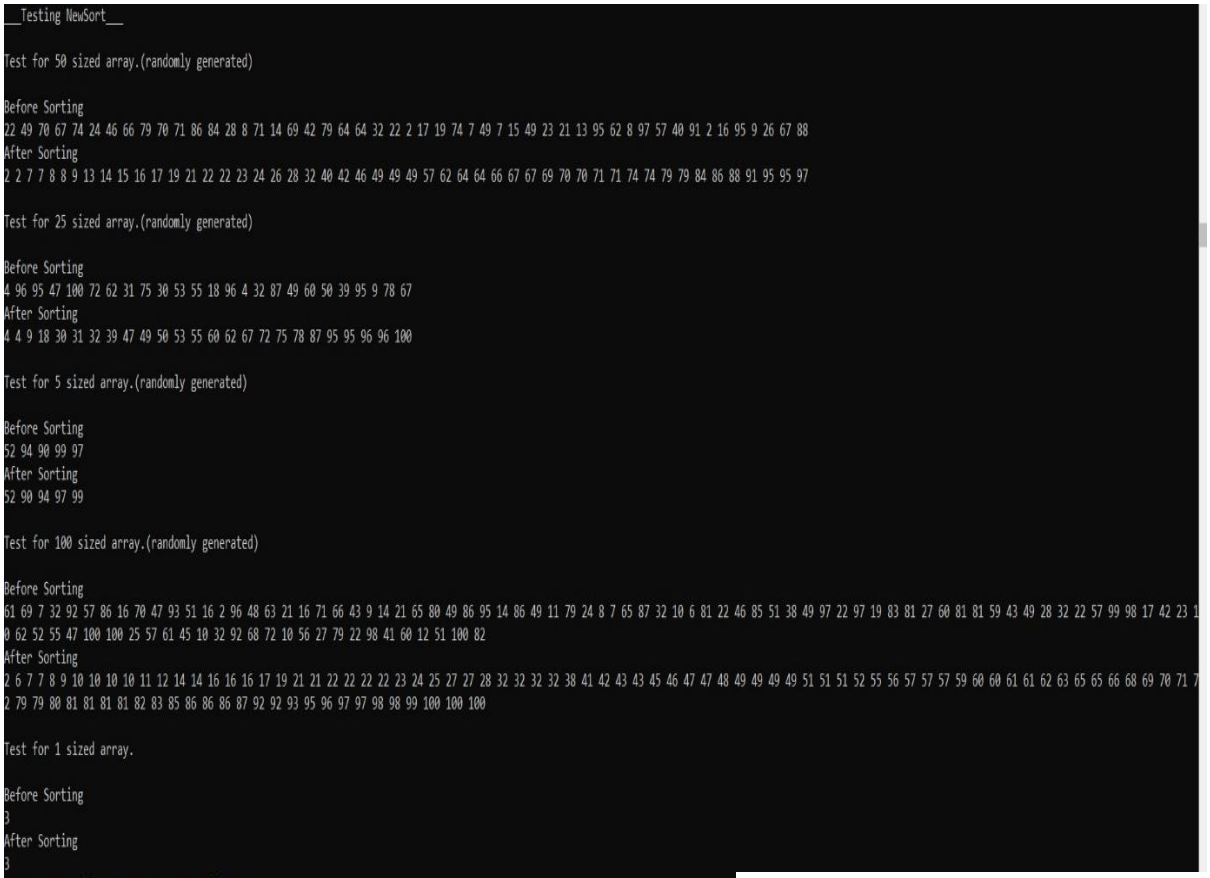
Testing Sort Algorithms

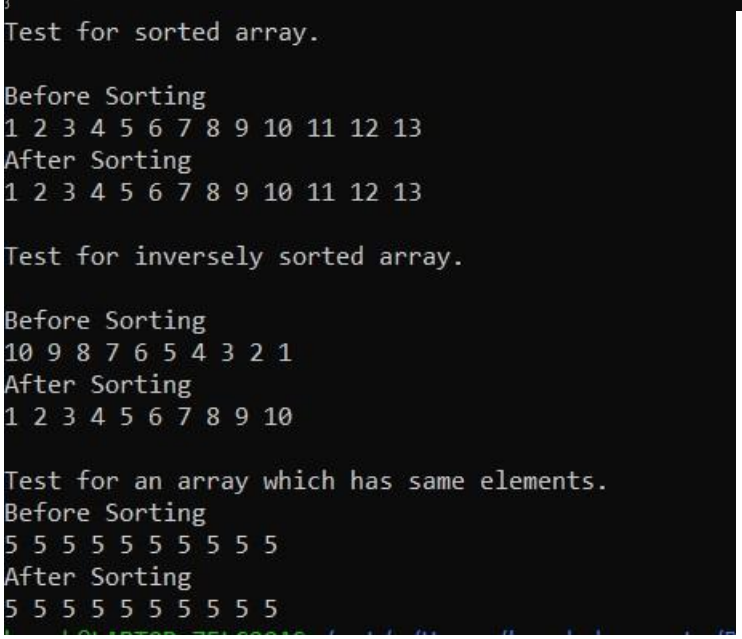
```
__Testing MergeSort__

Test for 50 sized array.(randomly generated)

Before Sorting
16 99 58 90 40 82 90 32 43 40 50 17 51 4 2 13 44 28 95 17 56 18 33 71 93 85 88 89 79 57 10 82 73 85 2 1 55 90 98 20 7 53 51 44 72 88 31 52 85 45
After Sorting
1 2 2 4 7 10 13 16 17 17 18 20 28 31 32 33 40 40 43 44 44 45 50 51 51 52 53 55 56 57 58 71 72 73 79 82 82 85 85 85 88 88 89 90 90 90 93 95 98 99

Test for 25 sized array.(randomly generated)

Before Sorting
80 40 79 92 40 85 13 28 78 46 1 86 43 29 15 19 66 20 83 92 81 64 24 51 32
After Sorting
1 13 15 19 20 24 28 29 32 40 40 43 46 51 64 66 78 79 80 81 83 85 86 92 92

Test for 5 sized array.(randomly generated)

Before Sorting
41 56 31 25 84
After Sorting
25 31 41 56 84

Test for 100 sized array.(randomly generated)

Before Sorting
24 88 99 14 36 58 53 52 84 70 15 56 18 73 30 12 89 24 17 6 67 47 36 32 18 41 10 57 69 81 96 10 73 97 11 92 36 44 19 98 55 18 14 87 95 92 97 13 18 84 87 65 20 20 91 9 98 23 67 8 71 86 51 33 64 50 57 77 78 50 28 9
0 4 8 23 75 60 4 34 50 52 48 96 22 45 76 90 10 82 50 65 69 11 30 3 52 82 9 31 22
After Sorting
3 4 4 6 8 8 9 9 10 10 10 11 11 12 13 14 14 15 17 18 18 18 18 19 20 20 22 22 23 23 24 24 28 30 30 31 32 33 34 36 36 36 41 44 45 47 48 50 50 50 50 51 52 52 52 53 55 56 57 57 58 60 64 65 65 67 67 69 69 70 71 73 73
75 76 77 78 81 82 82 84 84 86 87 87 88 89 90 90 91 92 92 95 96 96 97 97 98 98 99

Test for 1 sized array.

Before Sorting
3
After Sorting
3
```

```
Test for sorted array.

Before Sorting
1 2 3 4 5 6 7 8 9 10 11 12 13
After Sorting
1 2 3 4 5 6 7 8 9 10 11 12 13

Test for inversely sorted array.

Before Sorting
10 9 8 7 6 5 4 3 2 1
After Sorting
1 2 3 4 5 6 7 8 9 10

Test for an array which has same elements.
Before Sorting
5 5 5 5 5 5 5 5 5 5
After Sorting
5 5 5 5 5 5 5 5 5 5
```

```
__Testing QuickSort__

Test for 50 sized array.(randomly generated)

Before Sorting
21 38 18 75 92 45 77 97 52 11 70 35 29 10 29 58 60 22 32 79 56 78 15 83 46 13 38 30 55 99 47 13 92 94 18 43 61 9 97 84 43 40 25 62 65 42 28 40 6 24
After Sorting
6 9 10 11 13 13 15 18 18 21 22 24 25 28 29 29 30 32 35 38 38 40 40 42 43 43 45 46 47 52 55 56 58 60 61 62 65 70 75 77 78 79 83 84 92 92 94 97 97 99

Test for 25 sized arrays(randomly generated).

Before Sorting
3 20 16 51 76 56 100 42 68 52 26 42 62 24 6 75 12 70 73 96 84 16 74 22 88
After Sorting
3 6 12 16 16 20 22 24 26 42 42 51 52 56 62 68 70 73 74 75 76 84 88 96 100

Test for 5 sized array.(randomly generated)

Before Sorting
8 36 37 34 52
After Sorting
8 34 36 37 52

Test for 100 sized array.(randomly generated)

Before Sorting
5 37 90 13 25 17 25 99 24 10 48 72 19 70 43 98 6 20 32 39 40 17 49 80 32 57 85 70 96 20 3 25 92 19 8 2 47 50 67 17 22 53 88 28 38 44 18 12 29 24 22 18 14 16 76 97 62 28 9 6 71 78 52 12 23 79 50 88 86 64 97 42 1
34 40 2 7 64 53 89 82 57 74 7 83 22 62 46 8 49 3 10 15 87 79 49 63 29 48 46
After Sorting
1 2 2 3 3 5 6 6 7 7 8 8 9 10 10 12 12 13 14 15 16 17 17 17 18 18 19 19 20 20 22 22 23 24 24 25 25 25 28 28 29 29 32 32 34 37 38 39 40 40 42 43 44 46 46 47 48 48 49 49 49 50 50 52 53 53 57 57 62 62 63 64 64 67
 70 70 71 72 74 76 78 79 79 80 82 83 85 86 87 88 88 89 90 92 96 97 97 98 99

Test for 1 sized array.

Before Sorting
3
After Sorting
3

Test for sorted array.

Before Sorting
1 2 3 4 5 6 7 8 9 10 11 12 13
After Sorting
1 2 3 4 5 6 7 8 9 10 11 12 13
```

```
Test for inversely sorted array.

Before Sorting
10 9 8 7 6 5 4 3 2 1
After Sorting
1 2 3 4 5 6 7 8 9 10

Test for an array which has same elements.
Before Sorting
5 5 5 5 5 5 5 5 5 5
After Sorting
5 5 5 5 5 5 5 5 5 5
```





“make” command compiles, and “make run” command runs the program.