CSE 222
HOMEWORK 5 REPORT

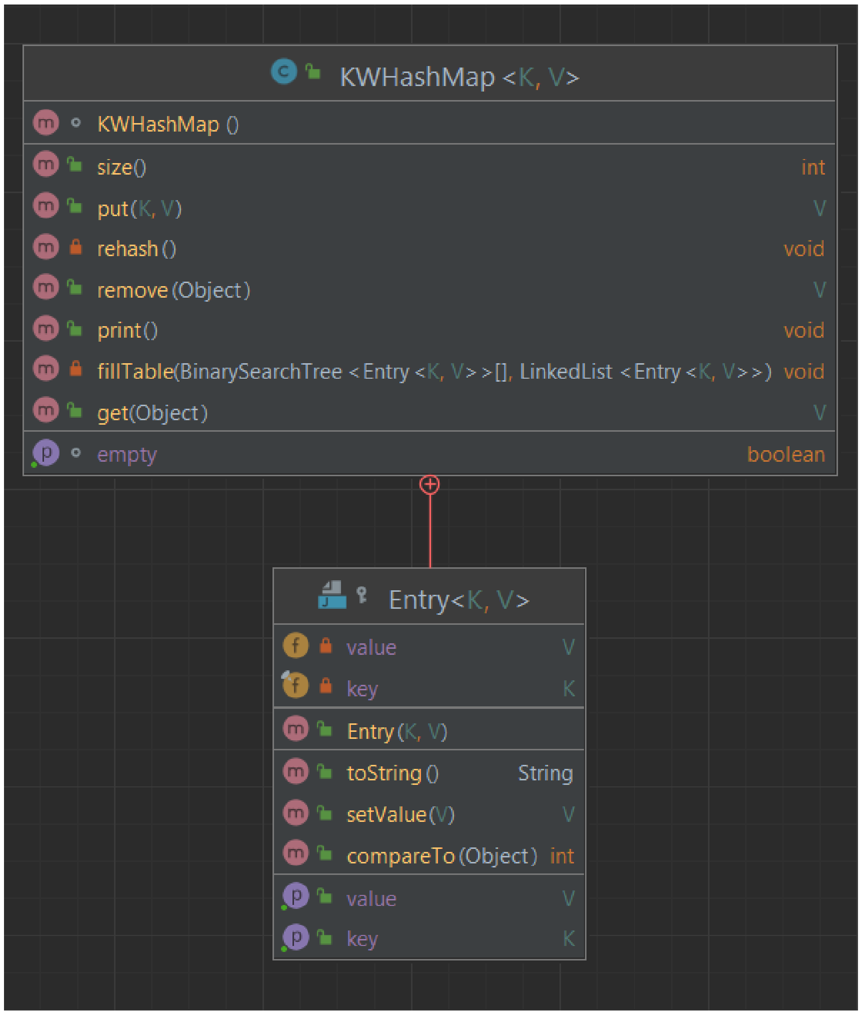
ERSEL CELAL EREN
1901042673

1 - System Requirements

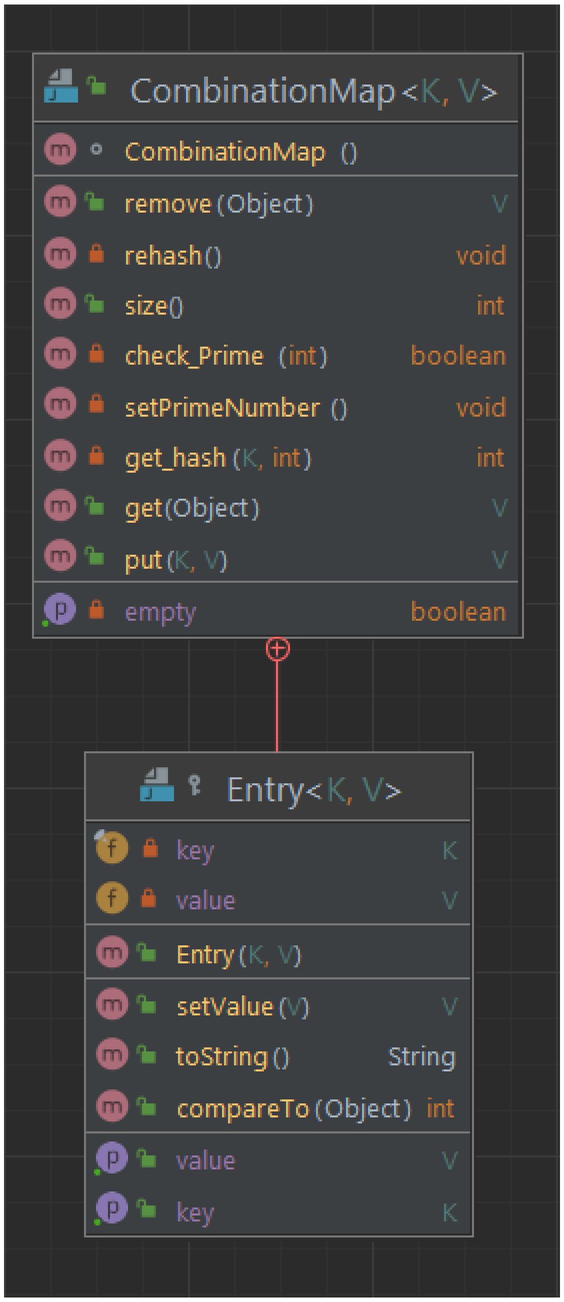
In this task we are asked to implement two hashmap with two different techniques and implement mergesort, quicksort and and another sorting algorithm which is given in PDF. One of the hashmap that we implemented has binary search tree instead of linkedlist in chaining. The other hashmap has two techniques mixed, double hashing and coalesced hashing. This program has one “Driver” program and it will be run by makefile. There is no interaction or menu for user. It has class implementation and test cases which is shown to user.

2 – Class Diagrams

KWHashMap Class with BinarySearchTree

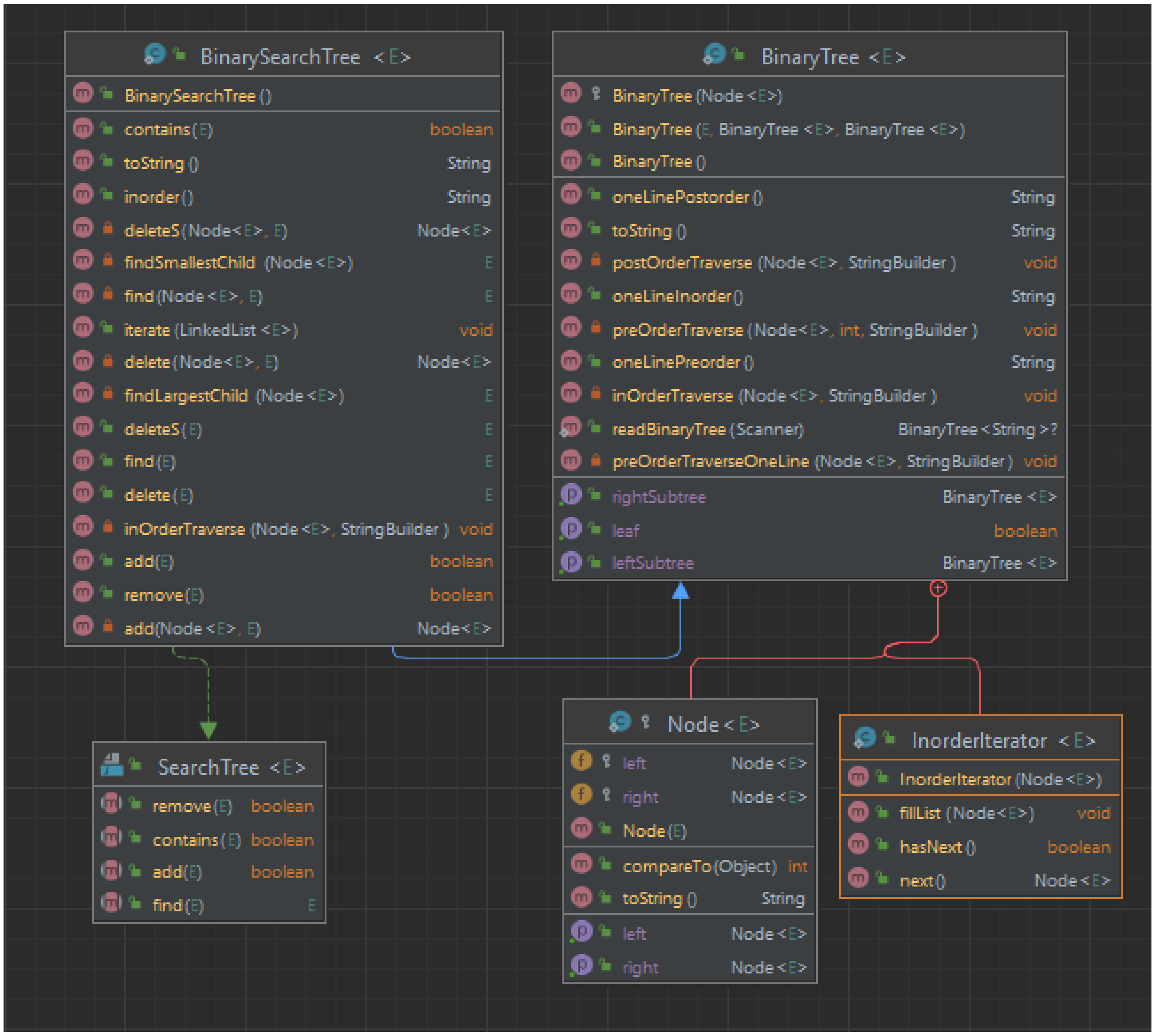


CombinationMap Class with Double Hash and Coalesced

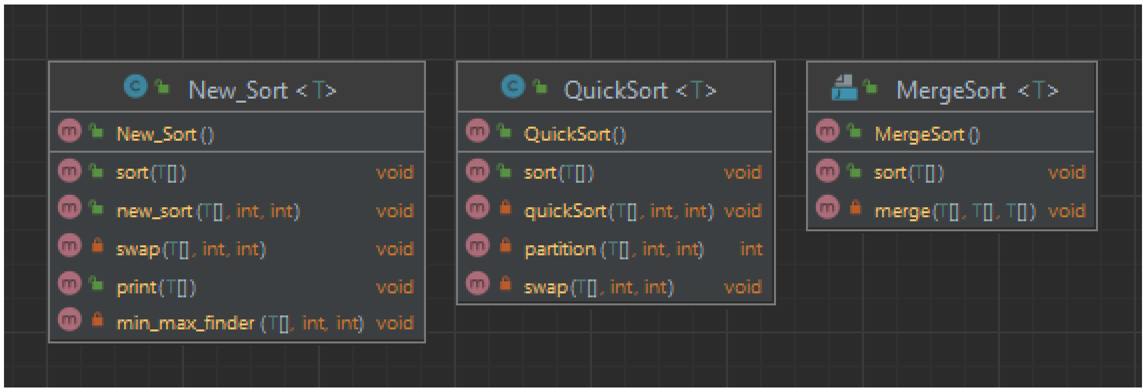


BinarySearchTree Class

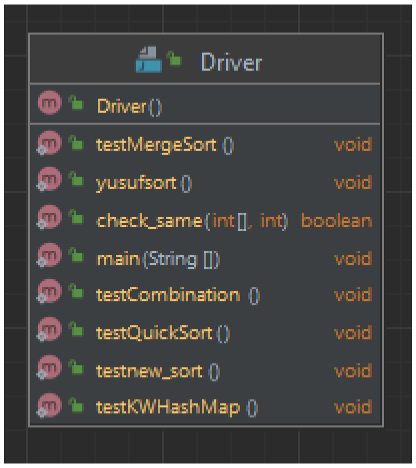
This class implementation is required because KWHashMap class uses BST for chaining. Most of this BinarySearchTree class is taken from our textbook but I implemented Iterator for it.


BinarySearchTree <E>
BinarySearchTree ()
contains (E) boolean
toString () String
inorder() String
deleteS (Node<E>, E) Node<E>
findSmallestChild (Node<E>) E
find (Node<E>, E) E
iterate (LinkedList <E>) void
delete (Node<E>, E) Node<E>
findLargestChild (Node<E>) E
deleteS (E) E
find (E) E
delete (E) E
inOrderTraverse (Node<E>, StringBuilder) void
add(E) boolean
remove(E) boolean
add(Node<E>, E) Node<E>
BinaryTree <E>
BinaryTree (Node<E>)
BinaryTree (E, BinaryTree <E>, BinaryTree <E>)
BinaryTree ()
oneLinePostorder () String
toString () String
postOrderTraverse (Node<E>, StringBuilder) void
oneLineInorder() String
preOrderTraverse (Node<E>, int, StringBuilder) void
oneLinePreorder () String
inOrderTraverse (Node<E>, StringBuilder) void
readBinaryTree (Scanner) BinaryTree<String>?
preOrderTraverseOneLine (Node<E>, StringBuilder) void
rightSubtree BinaryTree <E>
leaf boolean
leftSubtree BinaryTree <E>
SearchTree <E>
remove(E) boolean
contains(E) boolean
add(E) boolean
find(E) E
Node <E>
left Node<E>
right Node<E>
Node(E)
compareTo(Object) int
toString () String
left Node<E>
right Node<E>
InorderIterator <E>
InorderIterator (Node<E>)
fillList (Node<E>) void
hasNext () boolean
next() Node<E>

Sorting classes


New_Sort < T>
New_Sort ()
sort(T[]) void
new_sort (T[], int, int) void
swap (T[], int, int) void
print (T[]) void
min_max_finder (T[], int, int) void
QuickSort < T>
QuickSort()
sort(T[]) void
quickSort(T[], int, int) void
partition (T[], int, int) int
swap (T[], int, int) void
MergeSort < T>
MergeSort ()
sort(T[]) void
merge(T[], T[], T[]) void

Driver Class



3 – Problem Solution Approach

Q1.1) In this question, I have used BinarySearchTree array to store my datas. To use that, I have implemented BST class separately. All keys and values that belong to our data is kept in Entry<K,V> instances. In default version, linkedlist is used instead of BST. If "size" exceeds limit that I decided by load_threshold, table will be rehashed, and capacity will be duplicated.

Q1.2) For this class, I used Entry<K,V> array to keep my datas. At each insertion, after double hashing which is given in pdf as formula, I adjusted negative hash values and search for proper index according to coalesced hashing technique. In coalesced hashing every index has pointer to another index which has same hash value. Integer “used” field keeps number of elements in table. So if “used” exceeds limit that I decided by load_threshold, table will be rehashed and capacity will be duplicated.

Q1.3) In this question, I implemented MergeSort, QuickSort and new_sort classes. MergeSort and QuickSort is taken from our textbook. Pseudo code of new_sort algorithm is given in PDF but without true condition expressions and assignments, that pseudo code gives incorrect results. Because of that, I had to modify it. Problem in that code was that after min_max_finder method if head index is equal to MAX index, according to code, first head will be swapped by MIN index. After that, MAX which is equal to previous version of head will be swapped but swapped value will no longer be maximum element because we swapped head and MIN before.

After all implementations, with 100 randomly generated data sets for with size of 100, 1000, 10000, inserted elements into maps and all sorting algorithms are tested in same way.

4 – TEST CASES

Q1.1) KWHashMap

Items are added. I put 2 entries with same key, first (7, ersel-seven), then (7,ersel-seven2). But only first one is inserted. Same keys are not allowed.

```
KWmap.put(7, "ersel-seven");
KWmap.put(7, "ersel-seven2");
KWmap.put(23, "celal-twenty-three");
KWmap.put(9, "celal-nine");
KWmap.put(1, "eren-one");
KWmap.put(3, "ersel-tree");
KWmap.put(4, "ersel-four");
KWmap.put(12, "celal-twelve");
KWmap.put(16, "eren-sixteen");
KWmap.put(52, "ersel-fifty-two");
KWmap.put(15, "eren-fifteen");
```

Then trying to reach existing and non-existing values by testing get method. Value with 7 is inserted but 999 is not.

```
KWmap.get(7)
KWmap.get(999)
```

Removing existing and non-existing values by testing remove method. Value with 23 is inserted but 999 is not.

```
KWmap.remove(23)
KWmap.remove(999)
```

Q1.2) CombinationMap

Items are added. I put 2 entries with same key, first (8, ersel-eight), then (8,ersel-eight2). But only first one is inserted. Same keys are not allowed.

```
        Combmap.put(8, "ersel-eight");
        Combmap.put(8, "ersel-eight2");
        Combmap.put(23, "celal-twenty-three");
        Combmap.put(9, "celal-nine");
        Combmap.put(1, "eren-one");
        Combmap.put(3, "ersel-tree");
        Combmap.put(4, "ersel-four");
        Combmap.put(12, "celal-twelve");
        Combmap.put(16, "eren-sixteen");
        Combmap.put(52, "ersel-fifty-two");
        Combmap.put(15, "eren-fifteen");
```

Then trying to reach existing and non-existing values by testing get method. Value with 8 is inserted but 987 is not.

```
Combmap.get(8)
Combmap.get(987)
```

Removing existing and non-existing values by testing remove method. Value with 15 is inserted but 987 is not.

```
KWmap.remove(15)
KWmap.remove(987)
```

Printing final version of table

```
Combmap.print();
```

Q1.3) MergeSort Test Case

```
Integer[] array = {120,1,-1,33,99,100,65};
```

Q1.3) QuickSort Test Case

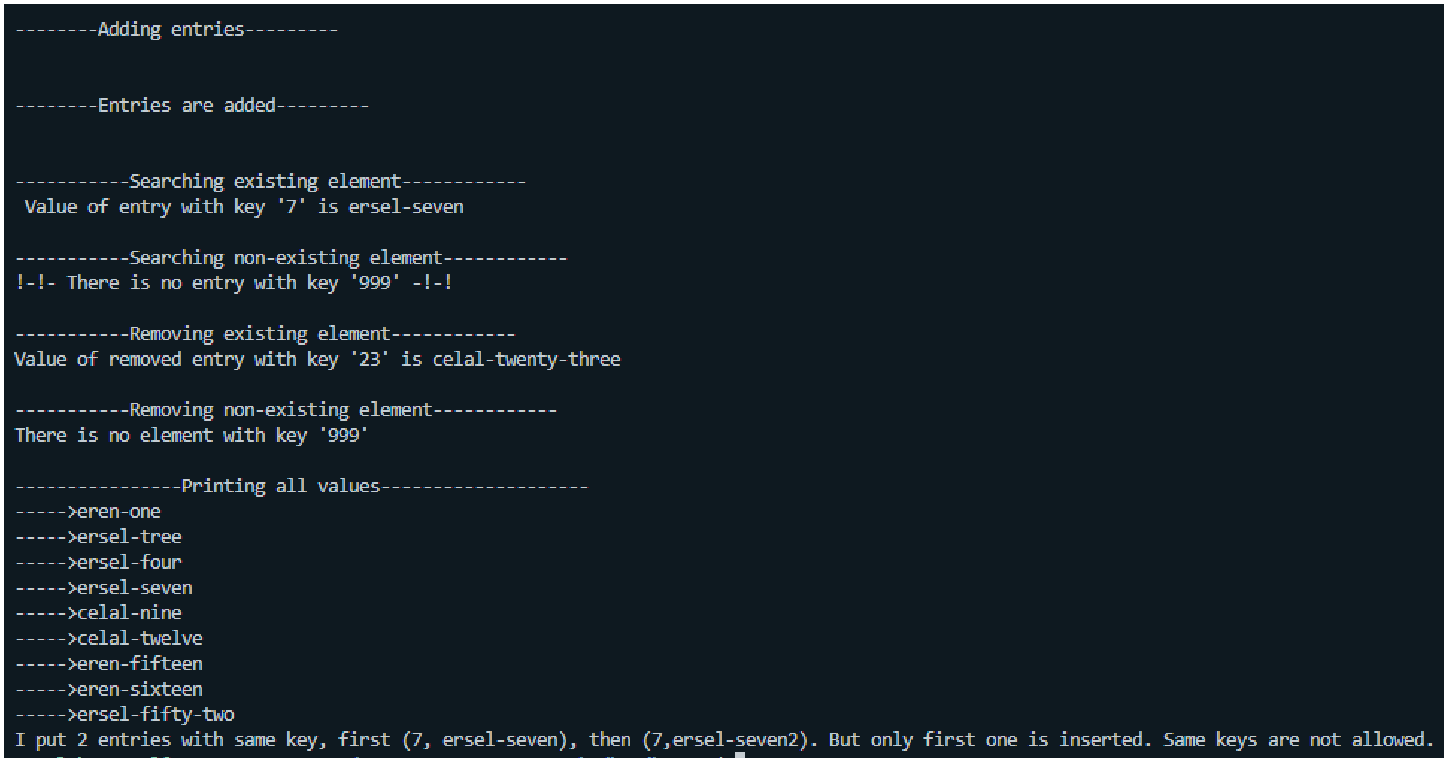
```
Integer[] array = {120,1,-1,33,99,100,65};
```

Q1.3) New_Sort Test Case

```
Integer[] array = {120,1,-1,33,99,100,65};
```

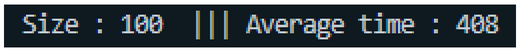
5 – RESULTS

Q1.1) KWHashMap



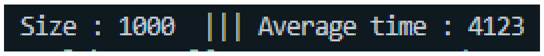
```
--------Adding entries---------


--------Entries are added---------


-----------Searching existing element------------
 Value of entry with key '7' is ersel-seven

-----------Searching non-existing element------------
!-!- There is no entry with key '999' -!-!

-----------Removing existing element------------
Value of removed entry with key '23' is celal-twenty-three

-----------Removing non-existing element------------
There is no element with key '999'

----------------Printing all values--------------------
----->eren-one
----->ersel-tree
----->ersel-four
----->ersel-seven
----->celal-nine
----->celal-twelve
----->eren-fifteen
----->eren-sixteen
----->ersel-fifty-two
I put 2 entries with same key, first (7, ersel-seven), then (7,ersel-seven2). But only first one is inserted. Same keys are not allowed.
```

100 array with 100 size, put() method



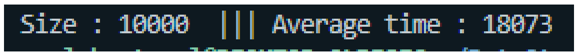
```
Size : 100  ||| Average time : 408
```

100 array with 1000 size, put() method



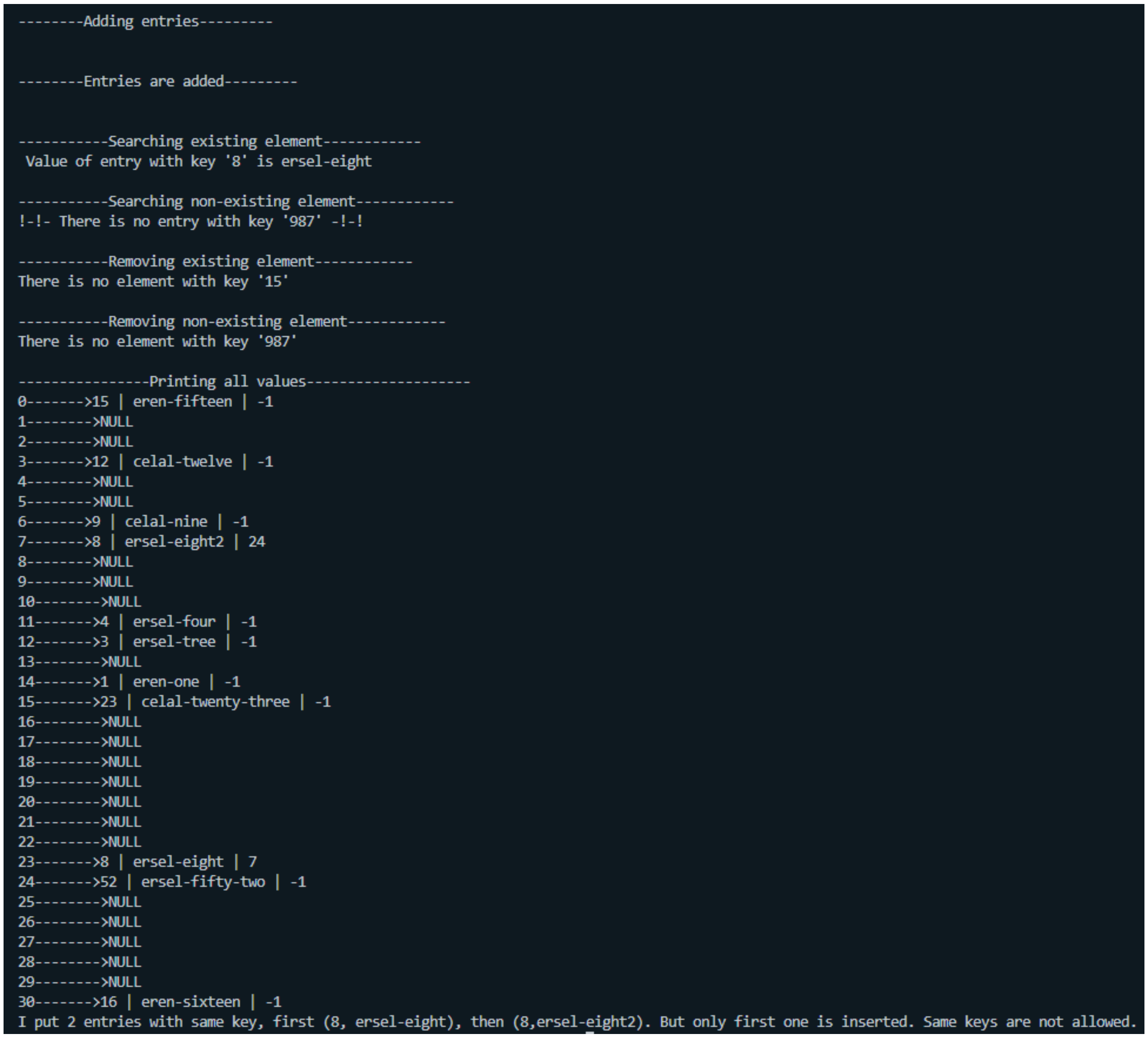
```
Size : 1000  ||| Average time : 4123
```

100 array with 10000 size, put() method



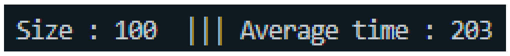
```
Size : 10000  ||| Average time : 18073
```

Q1.2) CombinationMap



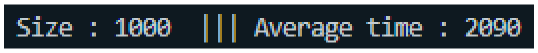
```
--------Adding entries---------


--------Entries are added---------


-----------Searching existing element------------
 Value of entry with key '8' is ersel-eight

-----------Searching non-existing element------------
!-!- There is no entry with key '987' -!-!

-----------Removing existing element------------
There is no element with key '15'

-----------Removing non-existing element------------
There is no element with key '987'

----------------Printing all values--------------------
0------->15 | eren-fifteen | -1
1-------->NULL
2-------->NULL
3------->12 | celal-twelve | -1
4-------->NULL
5-------->NULL
6------->9 | celal-nine | -1
7------->8 | ersel-eight2 | 24
8-------->NULL
9-------->NULL
10-------->NULL
11------->4 | ersel-four | -1
12------->3 | ersel-tree | -1
13-------->NULL
14------->1 | eren-one | -1
15------->23 | celal-twenty-three | -1
16-------->NULL
17-------->NULL
18-------->NULL
19-------->NULL
20-------->NULL
21-------->NULL
22-------->NULL
23------->8 | ersel-eight | 7
24------->52 | ersel-fifty-two | -1
25-------->NULL
26-------->NULL
27-------->NULL
28-------->NULL
29-------->NULL
30------->16 | eren-sixteen | -1
I put 2 entries with same key, first (8, ersel-eight), then (8,ersel-eight2). But only first one is inserted. Same keys are not allowed.
```

100 array with 10000 size, put() method



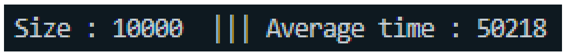
```
Size : 100  ||| Average time : 203
```

100 array with 10000 size, put() method



```
Size : 1000  ||| Average time : 2090
```

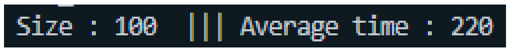
100 array with 10000 size, put() method



```
Size : 10000  ||| Average time : 50218
```

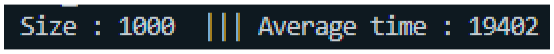
Q1.3) new_sort

1000 array with 100 size, put() method



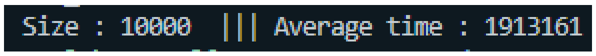
```
Size : 100   ||| Average time : 220
```

1000 array with 1000 size, put() method



```
Size : 1000   ||| Average time : 19402
```

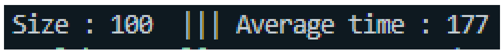
1000 array with 10000 size, put() method



```
Size : 10000   ||| Average time : 1913161
```

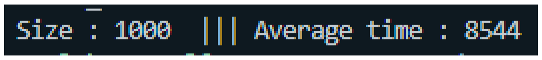
Q1.3) merge_sort

1000 array with 100 size, put() method



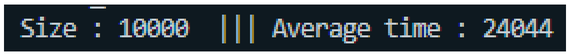
```
Size : 100   ||| Average time : 177
```

1000 array with 1000 size, put() method



```
Size : 1000   ||| Average time : 8544
```

1000 array with 10000 size, put() method



```
Size : 10000   ||| Average time : 24044
```

Time Complexity

Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

T(n) = 2T(n/2) + O(n)

The solution of the above recurrence is **O(nLogn)**. The list of size N is divided into a max of Logn parts, and the merging of all sublists into a single list takes O(N) time, the worst-case run time of this algorithm is O(nLogn)

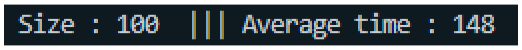
Best Case Time Complexity: **O(n*log n)**

Worst Case Time Complexity: **O(n*log n)**

Average Time Complexity: **O(n*log n)**

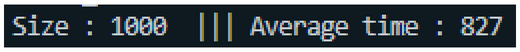
Q1.3) quick_sort

1000 array with 100 size, put() method



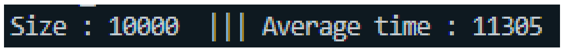
```
Size : 100  ||| Average time : 148
```

1000 array with 1000 size, put() method



```
Size : 1000  ||| Average time : 827
```

1000 array with 10000 size, put() method



```
Size : 10000  ||| Average time : 11305
```

Time complexity

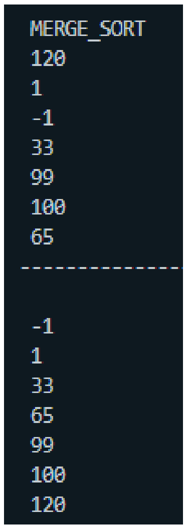
The time complexity of Quicksort algorithm is given by,

O(n log(n)) for best case,

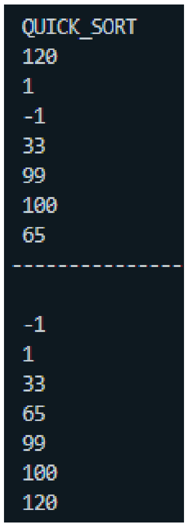
O(n log(n)) for the average case,

And **O(n^2)** for the worst-case scenario

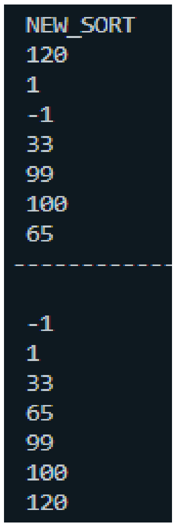
Here, before and after sorting by mergesort, quicksort and newSort with same data set



```
MERGE_SORT
120
1
-1
33
99
100
65
--------------

-1
1
33
65
99
100
120
```



```
QUICK_SORT
120
1
-1
33
99
100
65
---------------

-1
1
33
65
99
100
120
```



```
NEW_SORT
120
1
-1
33
99
100
65
------------

-1
1
33
65
99
100
120
```

Q1.2.a) Coalesced Hashing

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

Q1.2.b) Double hashing

Double hashing is also a collision resolution technique when two different values to be searched for produce the same hash key.

It uses one hash value generated by the hash function as the starting point and then increments the position by an interval which is decided using a second, independent hash function. Thus here there are 2 different hash functions.

The advantage of double hashing is as follows –

- ***Double hashing finally overcomes the problems of the clustering issue.***

The disadvantages of double hashing are as follows:

- ***Double hashing is more difficult to implement than any other.***
- Double hashing can cause thrashing.