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

ÇAĞRI ÇAYCI
1901042629

1. SYSTEM REQUIREMENTS

HashTableChain:

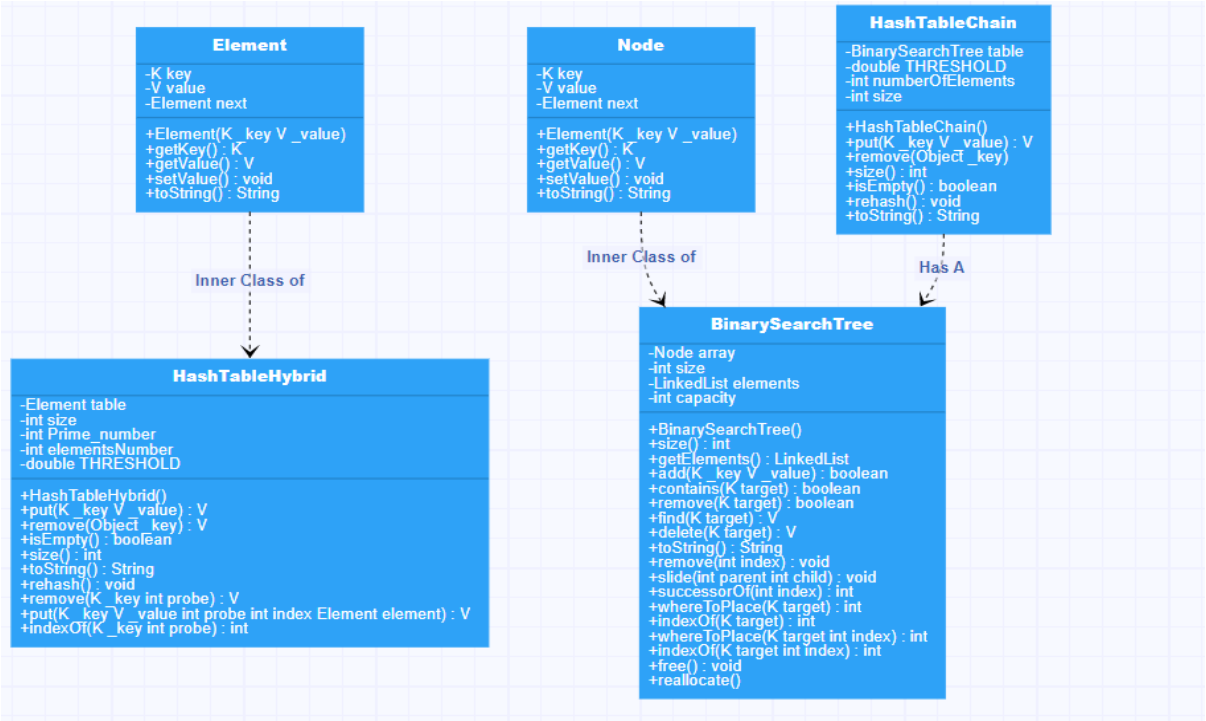
- 4 data fields are needed. One BinarySearchTree array is needed to keep elements in table. Two integers are needed to keep size of the table and keep the number of elements in table, one double number is needed to keep threshold of the table.
- One no parameter constructor is needed to create instances of HashTableChain class.
- A put method is needed to add some elements to HashTableChain.
- A remove method is needed to remove an element from HashTableChain.
- A size method is needed to get size of the table.
- A is empty method is needed to check whether a table is empty or not.
- A rehash method is needed to rehash the table when the division of number of elements by table size is bigger than threshold.
- A toString method is needed to convert class to String.

HashTableHybrid:

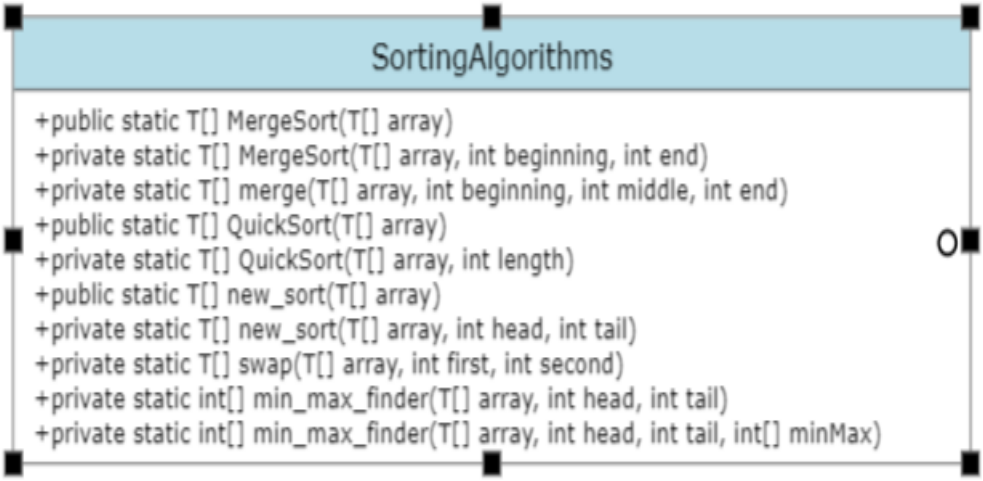
- An Inner class named Element is needed to keep key and value of an element.
- 5 data fields are needed. One Element array is needed to keep elements in table. One double number is needed to keep threshold. Three integers are needed to keep size, prime number, and the number of elements in table.
- One no parameter constructor is needed to create instances of HashTableHybrid class.
- A put method is needed to add some elements to HashTableHybrid.
- A remove method is needed to remove an element from HashTableHybrid.
- A size method is needed to get size of the table.
- A is empty method is needed to check whether a table is empty or not.
- A rehash method is needed to rehash the table when the division of number of elements by table size is bigger than threshold.
- An index of method is needed to find index of a key.
- A toString method is needed to convert class to String.

2. USE CASE AND CLASS DIAGRAMS

Question 1:



Question 2:



3. PROBLEM SOLUTION APPROACH

Coalesced Hashing: To provide coalesced hashing technique, an element keeps following key, value and reference of the next element. When adding an element to table, if there is already an element which has the same hash code with new element, the table is searched from bottom to the top, if an empty place is found, the new element is placed there, and reference of the new element adds the link of the element which has the same hash code with new element.

Double Hashing: To provide double hashing technique, when adding an element, if there is already an element which has the same hash code with new element, the index is recalculated. This process continues until an empty place is found.

Quick Sort Algorithm: Quick sort algorithm gets an array, selects first element of the array as pivot. Divides the array by two. One of the array contains elements which are less than pivot, the other one contains elements which are bigger than pivot. Apply this algorithm to new two array too. This process continues until length of the array is smaller than 2.

Merge Sort Algorithm: Merge sort algorithm gets an array and divides the array to subarrays which contains at most two elements. After this splitting, elements is ordered in the subarrays. Then, all subarrays are merged by considering their order.

New Sort Algorithm: New sort algorithm gets an array and each recursive call, it founds max element and min element, then max element and min element are placed, and their position is protected. After that, the same process is applied to array by not considering previous elements.

4. TEST CASES

Question 1:

```java
public static void testHashTableHybridTime(int size){
    Integer[] array = new Integer[size];
    for(int i = 0; i < size; i++){
        array[i] = (int) (Math.random() * 100);
    }

    HashTableHybrid<Integer, Integer> htc = null;
    long start = System.nanoTime();
    for(int i = 0; i < 100; i++){
        htc = new HashTableHybrid<Integer, Integer>();
        for(int j = 0; j < size; j++){
            htc.put(array[j], j);
        }
    }
    long end = System.nanoTime();
    System.out.println("Adding an element to HashTableHybrid takes " + ((end - start) / (100 * size)) + ". Size is: " + size);
    start = System.nanoTime();
    for(int i = 0; i < 100; i++){
        HashTableHybrid<Integer, Integer> htcNew = htc;
        for(int j = 0; j < size; j++){
            htcNew.remove(array[j]);
        }
    }
    end = System.nanoTime();

    System.out.println("Removing an element to HashTAbleHybrid takes " + ((end - start) / (100 * size)) + ". Size is: " + size);
}
```

```java
public static void testHashTableChainTime(int size){
    Double[] array = new Double[size];
    for(int i = 0; i < size; i++){
        array[i] = Math.random();
    }

    HashTableChain<Double, Integer> htc = null;
    long start = System.nanoTime();
    for(int i = 0; i < 100; i++){
        htc = new HashTableChain<Double, Integer>();
        for(int j = 0; j < size; j++){
            htc.put(array[j], j);
        }
    }
    long end = System.nanoTime();
    System.out.println("Adding an element to HashTableChain takes " + ((end - start) / (100 * size)) + ". Size is: " + size);

    start = System.nanoTime();

    for(int i = 0; i < 100; i++){
        HashTableChain<Double, Integer> htcNew = htc;
        for(int j = 0; j < size; j++){
            htcNew.remove(array[j]);
        }
    }
    end = System.nanoTime();

    System.out.println("Removing an element to HashTableChain takes " + ((end - start) / (100 * size)) + ". Size is: " + size);
}
```

Question 2:

```
public static void testNew(int size){
    Integer[][] array = createArray(size);
    long start = System.nanoTime();
    for(int i = 0; i < 1000; i++){
        SortingAlgorithms.new_sort(array[i]);
    }
    long end = System.nanoTime();
    System.out.println("Sorting " + array.length + " size array takes " + ((end - start)/1000) + " time with new sort.");
}
public static void testQuick(int size){
    Integer[][] array = createArray(size);

    long start = System.nanoTime();
    for(int i = 0; i < 1000; i++){
        SortingAlgorithms.QuickSort(array[i]);
    }
    long end = System.nanoTime();
    System.out.println("Sorting " + array.length + " size array takes " + ((end - start)/1000) + " time with quick sort.");
}

public static void testmerge(int size){
    Integer[][] array = createArray(size);
    long start = System.nanoTime();
    for(int i = 0; i < 1000; i++){
        SortingAlgorithms.MergeSort(array[i]);
    }
    long end = System.nanoTime();
    System.out.println("Sorting " + array[0].length + " size array takes " + ((end - start)/1000) + " with merge sort.");
}
```

5. RUNNING AND RESULTS

Question 1:

```
Adding an element to HashTableChain takes 690. Size is: 10000
Removing an element to HashTableChain takes 34. Size is: 10000
Adding an element to HashTableChain takes 969. Size is: 1000
Removing an element to HashTableChain takes 77. Size is: 1000
Adding an element to HashTableChain takes 4570. Size is: 100
Removing an element to HashTableChain takes 178. Size is: 100
Adding an element to HashTableHybrid takes 488. Size is: 10000
Removing an element to HashTAbleHybrid takes 16. Size is: 10000
Adding an element to HashTableHybrid takes 668. Size is: 1000
Removing an element to HashTAbleHybrid takes 24. Size is: 1000
Adding an element to HashTableHybrid takes 1061. Size is: 100
Removing an element to HashTAbleHybrid takes 82. Size is: 100
```

Question 2:

```
Sorting 10000 size array takes 4407858 time with quick sort.
Sorting 1000 size array takes 655896 time with quick sort.
Sorting 100 size array takes 22119 time with quick sort.
Sorting 10000 size array takes 3264503 with merge sort.
Sorting 1000 size array takes 427813 with merge sort.
Sorting 100 size array takes 102950 with merge sort.
Sorting 1000 size array takes 973567 time with new sort.
Sorting 100 size array takes 21819 time with new sort.
```