Assignment 2

Sorting, Priority Queues, Hashing

Exercise 1

In this exercise you have to provide an implementation based on HashTables of the *Map* Abstract Data Type (synonyms of the *Map* ADT are the *Dictionary* ADT and the *Associative Array* ADT https://en.wikipedia.org/wiki/Associative_array). If you have never heard about *Map* or *Dictionary* ADT before, here it is a short description: A Map contains a collection of (key,value) pairs, such that each possible key appears at most once in the collection. You can think that the usual dictionaries are a map where the Key is a word and the Value is the meaning of the word. Ask me in the next session (or write a post in mymoodle) if you have doubts about this ADT.

You have to implement a HashTable (called MyHashTableImpl.java) that implements the interface MyMap.java and provides operations insert(KeyType key, ValueType value), delete(KeyType key) and contains(KeyType key). It must also provide a method getLengthOfArray() that returns the length of the array that the HashTable is using in each moment (**NOT** the number of elements in the HashTable, just the length of the array). *This operation is not usually part of an ADT, but we need to know the length of the array during the tests to check that your implementation of the HashTable is correct.*

- Operation insert(KeyType key, ValueType value) does the usual insertion. If an element with the same key already exists in the table, this operation replaces its previous value with the value in the parameter (like a changing the definition of a word that already exists).
- Operation delete(KeyType key) does the usual deletion.
- Operation contains(KeyType key) returns the value associated with the key, if it exists in the table. If it does not exist, it returns *null*. For example in a *MyMap<String, String>*; after and insert(“ADT”, “Abstract Data Type”); the operation contains(“ADT”) will return “Abstract Data Type”.

The table must implement the **quadratic probing** and the internal rehash() function should be called when either the *load factor* is higher than a limit passed as an argument in the constructor of MyHashTable or

when an insertion fails (this is, when an insertion does not find an empty cell).

Hint: Your concrete classes for testing *KeyType* should override the hashCode() and equals(Object o) methods. We will test using classes String and Integer as keys, which already override them.

We have provided the Interface, and a set of minimal tests. You can see in the tests how the constructor should be called and insertions in the MyHashTable that stores a dictionary of <String,String>. We have provided the class MyHashTable.java only with the purpose that you can see the type of information that the getLengthOfArray() should return.

You can get full inspiration from the code in the reference book. All the operations must execute in average **O(1)**.

Exercise 2

Implement a Class MyArrayMathImpl.java that implements the interface ArrayMath.java. The explanation of the methods in the interface is the following:

- Method isSameCollection(int[] array1, int[] array2) receives as parameters 2 arrays of integer positive values (array1 and array2) and it returns true if array1 contains the same values as array2. For instance: isSameCollection([10,1,7,10], [1, 10, 7,10]) must return **true**, isSameCollection([10,1,7,9], [1, 10, 7,10]) must return **false**, and isSameCollection([1,7,10], [1, 10, 7, 7]) must return **false**. This operation should execute in average O(N), where N is the number elements in the array (any of them, because if the arrays are of different size you can directly return **false**).

- Method minDifferences(int[] array1, int[] array2) receives as parameters 2 arrays of the same size and
    - Computes the differences between pairs of elements <x1,x2> where x1 belongs to array1 and x2 belongs to array2. The pairs are generated in the following way: 1) the minimum element of array1 is paired with the minimum element of array2; 2) the minimum element of array1 without considering the already chosen in step 1 is paired with the minimum element of array2 without considering

already chosen in step 1; ... N) the maximum element of array 1 is paired with the maximum element in array2.
    - Returns the squared sum of the differences: $\sum(x2-x1)^2$
    - Example: array1=[2,5,3,9] and array2=[15,12,1,3].
        - Pairs are: <2,1>, <3,3>, <5,12>, <9,15> and the method returns: 86, which comes from $(1-2)^2+(3-3)+(12-5)^2+(15-9)^2= 86$
    - This method should execute in worst-case O(N logN)

- A method called getPercentileRange(int[] arr, int lower, int upper) that receives an array and two percentile values (2 numbers between 0 and 100) called lower and upper, with lower ≤ upper. The method returns an array with the values in *arr* that are between percentile “lower” and percentile “upper”. If you have doubts about the difference between the terms ‘percentile’ and ‘percentage’ check https://en.wikipedia.org/wiki/Percentile or ask me.
    - Example: Given the array arr=[20000, 160, -2, 4, 100, 6, 120, 8, 140, 1800].
        - getPercentileRange(arr, 0, 10)= [-2] //the 10th percentile is the value for which 10% of elements are lower than it.
        - getPercentileRange(arr, 10, 20)= [4] //Returns the values between the 10th percentile and the 20th percentile. The 20th percentile is the value for which 20% of the elements in the array are lower than it
        - getPercentileRange(arr, 10, 50)= [4,6,8,100] // It does not need to be ordered, [6, 100, 8, 4] would be also valid, or any permutation of them.
        - getPercentileRange(arr, 60, 70)= [140]
        - getPercentileRange(arr, 0, 100)= [-2,4,6,8,100,120,140,160,1800,20000] // any permutation of the values in *arr* would be valid
    - The constructor of MyMeasure class should not receive any argument. It must be called simply like: "ArrayMath mathsWithArrays = new MyArrayMathImpl();"

    - This method should execute in worst-case O(N logN) to pass the assignment. If you can reach average execution time O(N) you will get maximum grade.
    - About the situation like "*Which elements should we choose if the method receives the 25th and 30th percentile in an array of 5 elements?*" Do not worry for these odd situations. They will not exist because we will test the code with arrays whose size is a multiple of 100.

- *Hint:* you can use the Hash Table implemented in exercise 1.

Important: You must implement the ADTs in this assignment from scratch without reusing data structures from libraries (none among java sets, multisets, hash, provided sorting methods, etc.). This applies to all exercises in this assignment.

Important 2: At least, the minimal tests provided in the .zip must pass without any modification in their code. Therefore, you must be careful with your constructor, package names, implementing the interfaces as they are provided in the .zip, etc. If these tests do not pass, we cannot provide further feedback about other problems. However, these tests are not the comprehensive set of tests for the assignment!! You should test your code yourself.