Seoul National University

Data Structures

Spring 2023, U Kang

Programming Assignment 3: Internal Sorting (Chapter 7)

Due: May 17, 23:59, submit at eTL

**Reminders**

* The points of this homework add up to 100.
* Like all assignments, this has to be done individually.
* Lead T.A.: Ka Hyun Park ( [kahyun.park@snu.ac.kr](mailto:kahyun.park@snu.ac.kr))
* Write a program in Java **(JDK 17)**.
* **Do not use** Java Collection Framework and third-party implementation from the Internet.

# How to submit the programming assignment

## Create a ***JAR*** file including ‘src’ folder that contains your sources files.

### We will run your ***Main*** class in the JAR file to grade your programming assignments. Before submitting the JAR file, please check if your Main class in the JAR file works correctly.

### You **MUST** obey the I/O specification of the programming assignment, and rules for the submission of the programming assignment.

### Before submitting, check if your JAR file runs properly in your terminal with the following commands:

### “java -classpath PA\_##\_(StudentID).jar Main”.

(e.g.java -classpath PA\_03\_2023-12345.jar Main)

## Submit the jar file to the eTL (<http://etl.snu.ac.kr/>) .

* **NOTE:** do not make any changes to the Main.java file as your submitted .jar file will work incorrectly.
* **NOTE:** if you want to check your jar file working, you have to locate the .jar file and the *testcase* folder in the same directory (i.e. make the relative path of the input file to be “testcase/input.txt”)

# How to grade your programming assignment

## We made a grading machine to automatically grade your programming assignment. The machine will run your program and compare the answers and outputs that your program generates for given inputs. If your program cannot generate correct answers for an input file, it will not give you the point corresponding to the input. Our machine will consider the following scenarios:

### (***Accept***) When your program generates exact outputs for an input file, the machine will give you the point of the input.

### (***Wrong* *Answer***) When your program runs normally but generates incorrect outputs for an input file, including typos, the machine will not give you the point of the input.

### (***Run Error***) When your program does not run or is terminated suddenly for some reason, the machine will not give you the point of an input file because it cannot generate any outputs.

### (***Time Limit***) When your program runs over a predefined execution time for an input file, our machine will stop your program, and it will not give you the point of the input. The time limit of the execution is ***5 seconds***.

### 

## We will generate 10 input files and assign 10 points for each input file. For example, if your program gets 9 accepts, and 1 wrong answer by the machine, the total point will be 90 points. Hence, before submitting your programming assignment, please be sure that your program makes correct answers in a reasonable time for any input case.

# Problem

## 3.1 Part 1

How can we make the quicksort algorithm have better performance? One of the possible solutions is a hybrid sorting algorithm which combines the quicksort with an insertion sorting algorithm. The hybrid sorting algorithm begins with the quicksort and switches to the insertion sorting algorithm when the size of the subarray is equal to 32 (The size of the last subarray can be less than 32). This algorithm takes the advantages of both algorithms achieving practical performance on typical data sets.

In part 1 our goal is to implement a program that sorts the given key-value pairs in lexicographic order using the hybrid sorting algorithm. Fill your code in ‘HybridSorter.java’， ‘QuickSorter.java’ and ‘InsertionSorter.java’. Several rules that you **must** follow are as follows:

* Make your program run through the Main class.
* The number of given key-value pair doesn’t exceed 1,000,000.
* All keys are distinct.
* Your program should finish within 5 seconds.
* You should implement all functions listed in Section 4.

The hybrid sorting algorithm and the level to switch the sorting algorithm are important to ensure time complexity in the worst case, and better performance in the best case.

## Part 2

Now you have implemented Hybrid Sorting Algorithm, it is time to apply it to solving a problem. You are given five problems: 1. Find a k-th smallest element from an unsorted array. 2. Find the value of k for the pair that has the given key as its key. 3.Find the first element from two unsorted arrays. 4.Find the median element from two unsorted arrays. 5.Find the last element from two unsorted arrays.

*Problem 1*: implement a **search** method that returns the k-th smallest element from a given array. (element count starts from 1)

*Problem 2*: implement a **searchKey** method that returns the value of k for the pair that has the given key as its key. Here, k represents the k-th element in the sorted array. ( k starts from 1)

*Problem 3*: implement a **first** method that returns the first element from given two unsorted arrays after they are merged and sorted.

*Problem 4*: implement a **median** method that returns the median element from given two unsorted arrays. In statistics, the median is a value that separates the higher half from the lower half of a data sample. E.g.1 given an odd length sequence [1 3 7 8 9], the median is 7. E.g.2 given an even length sequence [2 4 6 8], the median is 5. However, in our implementation, the median element from an even length sequence will be a concatenation of two elements where the *minus sign* (-) is placed between two elements. (Read the interface description for more details)

*Problem 5*: implement a **last** method that returns the last element from given two unsorted arrays.

# Interface of Algorithms

1. HybridSorter

|  |
| --- |
| Function |
| void sort(Pair<K, ?>[] array, int left, int right) |
| Description |
| * Sorts the elements in a given array from left to right in lexicographic order using the hybrid sorting algorithm. * ‘left’ and ‘right’ are inclusive. |

|  |
| --- |
| Function |
| Pair<K,?> search(Pair<K, ?>[] array, int k) |
| Description |
| * Find the pair which is lexicographically the k-th smallest element in a given array. * (k) is a parameter as an integer, where k starts from 1. |

|  |
| --- |
| Function |
| Integer searchKey(Pair<K, ?>[] array, String key) |
| Description |
| * Find the value of k for the pair that has the given key as its key. * k represents the k-th element in the lexicographically sorted array. * (k) is a parameter as an integer, where k starts from 1. |

|  |
| --- |
| Function |
| Pair<K,?> first(Pair<K, ?>[] array1, Pair<K, ?>[] array2) |
| Description |
| * Find the pair which is lexicographically the first element of a given two arrays. * You might need to implement a helper method that will take two sorted arrays and return one merged array. |

|  |
| --- |
| Function |
| Pair<K,?> median(Pair<K, ?>[] array1, Pair<K, ?>[] array2) |
| Description |
| * Find the pair which is lexicographically the median element of a given two arrays. * Median of merged even length array is concatenated middle two elements, where a *minus sign* (-) placed between. * You might need to implement a helper method that will take two sorted arrays and return one merged array. |

|  |
| --- |
| Function |
| Pair<K,?> last(Pair<K, ?>[] array1, Pair<K, ?>[] array2) |
| Description |
| * Find the pair which is lexicographically the last element of a given two arrays. * You might need to implement a helper method that will take two sorted arrays and return one merged array. |

1. QuickSorter

|  |
| --- |
| Function |
| void sort(Pair<K, ?>[] array, int left, int right) |
| Description |
| * Sorts the elements in a given array from left to right in lexicographic order using the hybrid sorting algorithm. * ‘left’ and ‘right’ are inclusive. |

1. InsertionSorter

|  |
| --- |
| Function |
| void sort(Pair<K, ? >[] array, int left, int right) |
| Description |
| * Sorts the elements in a given array from left to right in lexicographic order using the insertion sort algorithm. * ‘left’ and ‘right’ are inclusive. |

# Specification of I/O

The program should accept only the inputs listed below and print the listed outputs.

1. n

|  |  |
| --- | --- |
| Input form | Output form |
| n (#elements1) [(#elements2)] |  |
| Description | |
| * Creates the first array of size (#elements1). * If the optional parameter (#element2) is given, creates a second array of size (#element2). * ‘n’ command can appear multiple times. | |
| * Example Input | * **Example Output** |
| * n 3 * n 3 3 | * n1: 3 n2: 3 |

1. append

|  |  |
| --- | --- |
| Input form | Output form |
| append (key) (value) |  |
| Description | |
| * Appends a new pair of which key and values are (key), and (value), respectively. * (key) is a string that doesn’t contain any whitespace. * (value) is a string. * If n command was passed with 2 parameters, then after the first array is filled, start to fill the second array. | |
| * Example Input | * **Example Output** |
| * append data 2 |  |

1. sort

|  |  |
| --- | --- |
| Input form | Output form |
| sort | Sorted |
| Description | |
| * Sorts the pairs in the array by key in lexicographic order using the hybrid sorting algorithm. | |
| * Example Input | * **Example Output** |
| * sort | * Sorted |

1. print

|  |  |
| --- | --- |
| Input form | Output form |
| print (k) | Print: (k) (key) (value) |
| Description | |
| * Prints the (k)-th key-value pair to the console. * (k) is between 0 and length of the array – 1. | |
| * Example Input | * **Example Output** |
| * print 0 | * Print: 0 data 2 |

1. search

|  |  |
| --- | --- |
| Input form | Output form |
| search (k) | Search: (k) (key) (value) |
| Description | |
| * Prints the (k)-th smallest key-value pair to the console. * (k) is between 1 and length of the array inclusive. | |
| * Example Input | * **Example Output** |
| * search 1 | * Search: 1 apple 2 |

1. searchKey

|  |  |
| --- | --- |
| Input form | Output form |
| searchKey (key) | searchKey: (k) (key) |
| Description | |
| * Prints the value of (k) for the pair that has the given key as its key value * (k) represents the (k)-th element in the sorted array. * (k) is between 1 and length of the array inclusive. | |
| * Example Input | * **Example Output** |
| * searchKey apple | * searchKey: 1 apple |

1. first

|  |  |
| --- | --- |
| Input form | Output form |
| first | First: (key) (value) |
| Description | |
| * Prints the first key-value pair to the console. | |
| * Example Input | * **Example Output** |
| * first | * First: apple 1 |

1. median

|  |  |
| --- | --- |
| Input form | Output form |
| median | Median: (key) (value) |
| Description | |
| * Prints the median key-value pair to the console. * When the merged array is even key and value are concatenated and *minus sign* placed between. | |
| * Example Input | * **Example Output** |
| * median * median | * Median: coconut 2 * Median: coconut-durian 2-4 |

1. last

|  |  |
| --- | --- |
| Input form | Output form |
| last | Last: (key) (value) |
| Description | |
| * Prints the last key-value pair to the console. | |
| * Example Input | * **Example Output** |
| * last | * Last: strawberry 3 |

# Sample Input

n 6

append apple 3

append coconut 5

append strawberry 1

append durian 2

append banana 4

append watermelon 6

print 0

print 1

print 2

search 1

searchKey apple

search 2

searchKey durian

sort

print 0

print 1

print 2

searchKey coconut

n 3 3

append apple 1

append coconut 2

append strawberry 3

append durian 4

append banana 5

append watermelon 6

median

first

n 2 3

append apple 1

append coconut 2

append strawberry 3

append durian 4

append banana 5

first

median

last

# Sample Output

apple

coconut

strawberry

durian

banana

watermelon

Print: 0 apple 3

Print: 1 coconut 5

Print: 2 strawberry 1

Search: 1 apple 3

searchKey: 1 apple

Search: 2 banana 4

searchKey: 4 durian

Sorted

Print: 0 apple 3

Print: 1 banana 4

Print: 2 coconut 5

searchKey: 3 coconut

n1: 3 n2: 3

apple

coconut

strawberry

durian

banana

watermelon

Median: coconut-durian 2-4

First: apple 1

n1: 2 n2: 3

apple

coconut

strawberry

durian

banana

First: apple 1

Median: coconut 2

Last: strawberry 3