**Question-2:**

**Github:** <https://github.com/Kishan-Kumar-Zalavadia/ACA_Homework-1>

**a. Describe an algorithm in pseudocode, with comments on how to find the median.**

**Algorithm: Find Median**

**Pseudocode:**

Function mergeNSortedArrayLists(arrays):

result = create an empty list

indices = create an array of integers of size N, initially filled with zeros

allListsEmpty = false

while not allListsEmpty:

allListsEmpty = true

minValue = Integer.MAX\_VALUE

minIndex = -1

for i = 0 to N - 1:

if indices[i] < size of arrays[i]:

allListsEmpty = false

currentValue = arrays[i][indices[i]]

if currentValue < minValue:

minValue = currentValue

minIndex = i

if minIndex != -1:

result.append(minValue)

indices[minIndex]++

return result

# The main function reads input from a file, merges N sorted arrays, and finds the median.

Function main():

inputFilePath = "path to the input file"

fileScanner = create Scanner object for inputFilePath

inputSortedList = create an empty list of lists

# Read input from the file and create a list of sorted arrays

while fileScanner has next line:

line = read the next line from fileScanner

if line is empty:

continue # Skip empty lines

values = split line by space

innerList = create an empty list

# Convert the values to integers and add them to innerList

for value in values:

append Integer.parseInt(value) to innerList

append innerList to inputSortedList

print "Input:", inputSortedList

# Merge the sorted arrays into one sorted list

mergedList = mergeNSortedArrayLists(inputSortedList)

print "Sorted Array:", mergedList

finder = create a MedianFinder object

median = finder.findMedian(mergedList)

print "Median:", median

**b.** **Given there are N schools with x1,x2,x3,..., xn students respectively, what is the**

**the complexity of your algorithm**

The time complexity of the code for merging arrays of sizes x1, x2, x3, ..., xn is O(N \* (x1 + x2 + x3 + ... + xn)), where N is the number of arrays.

In other words, The time complexity of the code is O(M + N), where M is the sum of the students of all the schools, and N is the number of schools.

**c. Provide documentation on how your algorithm works.**

The algorithm implemented is a variation of the Merge sort (or Merge-Merge) Algorithm. The merge algorithm is commonly used to combine two or more sorted sequences into a single sorted sequence. In this specific code, it's extended to merge multiple (more than two) sorted lists.

The main idea of the merge algorithm is to select the smallest available element from all the input lists in each step, ensuring that the merged list remains sorted without the need to sort.

**Code Explanation:**

The code is designed to merge N-sorted arrays (represented as lists) into a single sorted list and then find the median of that merged list. Here's a step-by-step explanation:

Merging Sorted Lists:

The first part of the code focuses on merging N-sorted arrays efficiently.

Function: `mergeNSortedArrayLists`

- **Input**: `List<List<Integer>> arrays` - A list of N sorted lists, where each inner list represents a sorted array of integers.

- **Output**: `List<Integer>` - A single sorted list containing all elements from the input lists.

**Merging Algorithm:**

1. Initialize an empty result list `result` that will store the merged elements.

2. Create an array of `indices` to keep track of the current position (index) in each input list. Initialize all indices to 0.

3. Create a boolean variable `allListsEmpty` and set it to `false`. This variable will help determine when all input lists are exhausted.

4. Start a timer to measure the runtime.

5. Enter a loop that continues until `allListsEmpty` is `true`, meaning that all input lists have been fully processed.

6. Inside the loop, initialize `minValue` to `Integer.MAX\_VALUE` and `minIndex` to -1. These variables will keep track of the smallest value among the heads of the input lists and the index of the list from which the smallest value was taken.

7. Iterate over each input list using a for loop.

8. For each list, check if the corresponding index (`indices[i]`) is within the bounds of the list. If it is, this list is not empty.

9. If the list is not empty, retrieve the value at the current index in that list and compare it to `minValue`. If the current value is smaller, update `minValue` and `minIndex`.

10. After processing all lists, you will have found the smallest value among the heads of the input lists.

11. If `minIndex` is not -1 (meaning a valid minimum value was found), add `minValue` to the `result` list and increment the index for that list (`indices[minIndex]`) to move to the next value in that list.

12. Check if all input lists are empty by inspecting the `allListsEmpty` flag. If any list is not empty, set `allListsEmpty` to `false`.

13. The loop repeats until `allListsEmpty` becomes `true`, at which point all input lists have been fully processed.

14. Stop the timer to measure the runtime.

15. Return the `result` list, which contains all elements from the input lists in sorted order.

**Finding the Median:**

After merging the input lists, the code proceeds to find the median of the merged list.

**Function**: `findMedian`

- **Input**: `List<Integer> sortedList` - A sorted list of integers.

- **Output**: `double` - The median value of the list.

**Median Calculation:**

1. Calculate the total size of the merged list using `sortedList.size()`.

2. Determine the middle index by dividing the total size by 2.

3. If the total size is even (i.e., `totalSize % 2 == 0`), return the average of the two middle elements. This is done by retrieving the elements at indices `middle - 1` and `middle`, summing them, and dividing by 2.0 to ensure a floating-point result.

4. If the total size is odd, simply return the middle element, which can be obtained using `sortedList.get(middle)`.

**Main Function:**

The `main` function coordinates the execution of the code.

1. It reads the input data from a file and stores it in a list of lists (`inputSortedList`).

2. The code then merges the input lists using the `mergeNSortedArrayLists` function and measures the runtime for this operation.

3. After merging, it calculates the median using the `findMedian` function and displays both the merged list and the calculated median.

In summary, this code efficiently merges N sorted lists without re-sorting the entire dataset and then calculates the median from the merged list. The result is runtime-efficient, making it suitable for processing large datasets of sorted lists.

**Reference:**

- https://www.codingninjas.com/studio/online-compiler/online-cpp-compiler

- ChatGPT -> For Quick Snippets