Concurrency Python

arrayGenerator.py

# Generate and return a list of random integers

Replace return [random.randint(0, 10000) for \_ in range(array\_size)] with

return random.sample(range(10001), array\_size)

# A list containing array\_size elements is generated, and the size of every element is between 0 and 10000 both inclusive.

This is because the values generated by the random.sample function will be unique, and the code will be shorter and easier to understand. If one were to use the random.randint function to generate the values and make sure that they are unique it would take a lot of iterations and it would take longer for the code to execute if the array contains a lot of elements.

bubbleSortFunction.py

To improve the code a Boolean variable can be added to check if there was a swap in the pass. If there was no swap the program exits as it would already be sorted. We can also keep track of the last index which was swapped and put in the correct position so that we avoid any extra comparisons and do all the necessary arrangements for the for loop. This would look like this:

def bubble\_sort(input\_array):

if not isinstance(input\_array, list):

raise ValueError("Input must be a list")

array = input\_array.copy()

for i in range(len(array)):

swapped = False

lastindexswapped = n-1

for j in range(0, lastindexswapped):

if array[j] > array[j + 1]:

array[j], array[j + 1] = array[j + 1], array[j]

swapped = True

lastindexswapped = j

if not swapped:

return array

return array

Concurrency Java

ArrayGenerator.java

When creating the ArrayList the size of the ArrayList can be initialized as since it is a dynamic data structure if its size is specified before hand this reduces the amount of times it has to resize the array. This can be done by changing the line: List<Integer> elements = new ArrayList<>(); to List<Integer> elements = new ArrayList<>(size);

BubbleSortFunction.java

Just like the bubble sort function using python a Boolean variable to check if there was a swap can be added so that if the list is sorted the program exists immediately. In the inner for loop where variable j is used as a counter for the range for which it needs to loop can be changed to array.size()-i since in this way the loop won’t need to iterate over the elements which have already been sorted. All of this can be implemented by changing the code to look like this:

boolean swapped;0

// Boolean variable is declared

for (int i = 0; i < sortedList.size(); i++) {

swapped = false;

// Boolean variable is initialized to be false so that it is then checked if there was another swap

for (int j = 0; j < sortedList.size() - i; j++) {

// range is changed to sortedList.size() – i

if (sortedList.get(j) > sortedList.get(j + 1)) {

int temp = sortedList.get(j);

sortedList.set(j, sortedList.get(j + 1));

sortedList.set(j + 1, temp);

swapped = true;

// if there was a swap Boolean variable is initialized to true

}

}

if (!swapped) {

// if Boolean variable is false the program will break out of the loop

break;

}

}

MergedArray.java

To add all the remaining elements after one of the arrays is empty as all elements have been merged the program is looping and always adding every element one by one. Rather than looping, the addAll function can be used which makes the code shorter and uses the same concept. 8 lines of code become only 2 lines of code. This can be implemented in the following way:

This is how it currently is:

while (currentPos1 < size1) {

mergedArray.add(array1.get(currentPos1));

currentPos1++;

}

while (currentPos2 < size2) {

mergedArray.add(array2.get(currentPos2));

currentPos2++;

}

This is how it can be implemented:

mergedArray.addAll(array1.subList(currentPos1, size1));

// This basically adds all elements of array 1 which fall in the index range of currentPos1 and size1.

mergedArray.addAll(array2.subList(currentPos2, size2));

// This basically adds all elements of array 2 which fall in the index range of currentPos2 and size2.

Just like before the size of the list of the mergedArray can also be specified beforehand. So the line: List<Integer> mergedArray = new ArrayList<>(); now becomes List<Integer> mergedArray = new ArrayList<>(size1 + size2);

Main.java

When the user is prompted to input the size of the array it should be checked if the size inputted by the user is valid. To do this to the readSizeFromConsole() method a do while loop needs to be implemented. This is how it can be implemented:

private static int readSizeFromConsole() {

Scanner scanner = new Scanner(System.in);

int num;

do {

System.out.print("Enter a positive integer for the array size: ");

while (!scanner.hasNextInt()) {

System.out.print("Invalid input. Please enter a positive integer: ");

scanner.next(); // consume the invalid input

}

num = scanner.nextInt();

if (num <= 0) {

System.out.println("Please enter a positive integer for the array size.");

}

} while (num <= 0);

scanner.close();

return num;

}