Paul Franklin

DBA Critical Thinking 5

1. For the arrayPrint method, time complexity is O(n), or linear. The digitSort method is also O(n), as is arrayMax. radixSorter is more complex given that the length of the array and the number of digits in the array values are both important factors. After doing some research, I found that the proper expression of the time complexity of radixSorter is O(d⋅n), where 𝑑 is the number of digits in the largest number in the array and 𝑛 is the number of elements. Big Oh notation for the entire Radix Sort algorithm implemented in RadixSortManager is O(d⋅n)

The RadixSortManager class uses Radix Sort to sort arrays of Integer values efficiently. The sorting time mainly depends on the number of digits in the largest number. For arrays with numbers having relatively few digits, this method provides fast sorting.

1. My Radix Sort algorithm sorts the following Integer array: 783 99 472 182 264 543 356 295 692 491 94. Firstly, radixSorter method calls the arrayMax method to obtain the maximum Integer object in the array, which is 783. Next, radixSorter calls the digit sort method as many times as there are digits in the mainArray’s maximum Integer object. The digitSorter first sorts the values according to their least significant digit, which is the rightmost digit, aka unit place, and with each subsequent call to digitSort, the next most significant digit place is sorted, until the most significant digit place is sorted.

For each call, digitSort begins by filling in the digitCount array so that the values at each index of digitCount represent the number of digits in the mainArray at a particular positional location (units case, tens place, hundreds place, etc.) in the Integer elements that are equivalent to the value of the index. For example, if, for the units case, digitCount[2] = 5, that means 5 of the values in mainArray have the digit 2 in the units case.

With the digitCount array established, this array must be converted so that we can ultimately use it to change mainArray[] so that its values are sorted by the positional value (units, tens, etc.) that the current iteration of digitSort is engaging. the array {2, 3, 0, 1} is converted so that each value represents the first occurrence of the digit at the current index value. So, 2 means that the first occurrence of the value 1 occurs at index 2.

Now we take the converted digitCount, which we call “convertedCount”, and we iterate through mainArray in reverse order and along the way we add the original mainArray values but in a sorted order—sorted by the positional value aka. place (tens, hundreds, etc.) that is engaged by the current iteration of digitSort. When the multiple values have the same digit in the current positional value of whatever significance, these Integer values are then sorted according to their relative positions in the original, unaltered mainArray.

digitSort is repeated as many times as there are digits in mainArray’s maximum value, and once the sorting iterations are completed, mainArray’s Integer values, 783 99 472 182 264 543 356 295 692 491 and 94, will now be properly sorted from smallest to largest: 4 99 182 264 295 356 472 491 543 692 783, and the arrayPrint method will iteratively print these ordered values to the console. The main method of my other class, RadixSortTester, is where RadixSortManager will be instantiated, the initial version of mainArray will be initialized, and the methods in RadixSortManager will be called by the new instance to conduct Radix Sort.

1. The RadixSortManager class uses Radix Sort to efficiently sort arrays of Integer values. The sorting speed primarily depends on how many digits the largest number in the array has. If the numbers in the array have relatively few digits, the sorting process is faster because Radix Sort operates efficiently on numbers with a limited number of digits.

In essence, RadixSortManager implements Radix Sort, which sorts numbers by their individual digits, making it faster when numbers are not excessively large or have a lot of digits. This methodical approach ensures efficient sorting for various arrays of Integer values.

The majority of the methods in class RadixSortManager have linear time complexity, O(n), which means that the algorithm can handle a large quantity of inputs without its performance being hampered.

**Source Code:**

import java.util.\*;  
  
public class RadixSortManager {  
  
 */\*\*  
 \* Prints array of Integer values.  
 \*/* static void arrayPrint(Integer mainArray[], int length) {  
 for (int j = 0; j < length; j++) {  
 System.*out*.print(mainArray[j] + " ");  
 }  
 }  
  
 */\*\*  
 \* Sorts an array of Integer values by sorting digits, starting with the least  
 \* significant digits and ending with the most significant digits  
 \*/* static void digitSort(Integer mainArray[], int length, int exp) {  
 Integer[] convertedCount = new Integer[length]; // output array  
 Integer[] digitCount = new Integer[10];  
 Arrays.*fill*(digitCount, 0);  
  
 for (int j = 0; j < length; j++) {  
 digitCount[(mainArray[j] / exp) % 10]++;  
 }  
  
 for (int j = 1; j < 10; j++) {  
 digitCount[j] += digitCount[j - 1];  
 }  
  
 for (int j = length - 1; j >= 0; j--) {  
 convertedCount[digitCount[(mainArray[j] / exp) % 10] - 1] = mainArray[j];  
 digitCount[(mainArray[j] / exp) % 10]--;  
 }  
  
 System.*arraycopy*(convertedCount, 0, mainArray, 0, length);  
 }  
  
 */\*\*  
 \* Central radix sort method that uses int value returned by arrayMax method  
 \* and Integer array returned by digitSort method  
 \*/* static void radixSorter(Integer[] integerArray, int length) {  
 int m = *arrayMax*(integerArray, length);  
 for (int exp = 1; m / exp > 0; exp \*= 10) {  
 *digitSort*(integerArray, length, exp);  
 }  
 }  
  
 */\*\*  
 \* Finds the maximum value in an array of Integer values.  
 \*/* static int arrayMax(Integer mainArray[], int length) {  
 int maxValue = mainArray[0];  
 for (int j = 1; j < length; j++) {  
 if (mainArray[j] > maxValue) {  
 maxValue = mainArray[j];}  
 }  
 return maxValue;  
 }  
}

*/\*\*  
 \* This class tests the functionality of Radix Sort implementation in RadixSortManager.  
 \* It creates an instance of RadixSortManager, initializes an array of integers,  
 \* sorts the array using radix sort, and prints the sorted array.  
 \*/*public class RadixSortTester {  
 */\*\*  
 \* main method and test entry point  
 \* @param args  
 \*/* public static void main(String[] args) {  
 RadixSortManager testMyRadix = new RadixSortManager();  
 Integer mainArray[] = { 783, 99, 472, 182, 264, 543, 356, 295, 692, 491, 94 };  
 int length = mainArray.length;  
 testMyRadix.*radixSorter*(mainArray, length);  
 testMyRadix.*arrayPrint*(mainArray, length);  
 }  
}

**Screenshots of execution:**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated**