# Computer Science III Honors

**Radix Sort Lab/Bonus**

A radix sort is being used to order a list of non-negative integers. It makes as many passes through the list as there are digits in the largest number to be sorted. For example, if the largest integer in the list was 492, then the algorithm would make three passes through the list to sort it.

Assume that the list of numbers to be sorted is in an integer array called nums. In each pass through the list, the radix sort algorithm sorts the numbers based on a different digit, working from the least to the most significant digit. To do this, it uses an intermediate data structure, queues, a List of ten queues. Each number is placed into the queue corresponding to the value of the digit being examined. For example, in the first pass the digit in the ones place is considered, so the number 1345 would be enqueued into index 5 of queues. The number 1260 would be enqueued into index 0 of queues. In each pass, the algorithm moves numbers to be sorted from nums to the List of queues and then back to nums as described below. After the last pass, the integers in nums are in order from smallest to largest.

Radix Sort Algorithm:

In each pass through the list, do the following two steps.

#### Step 1 – Taking each integer from nums in order, insert the integer into the queue corresponding to the value of the digit currently being examined. If the integer being examined does not have a digit at a given place value, 0 is assumed for that place value. For example, 95 has no digit in the hundreds place, so, when examining the hundreds digit, the algorithm would assume the value in the hundreds place is zero and enqueue 95 into index 0 of queues.

#### Step 2 – After all integers have been inserted into the appropriate queues, the array nums is filled from beginning to end by emptying the queues into nums, starting with the integers in index 0 of queues and proceeding sequentially.

For example, assume that nums contains the integers 6380, 95, 1345, 382, 1260, 100, and 2492. The sort will take four passes, because the largest integer in nums has 4 digits. The following diagram shows the sorting process. (For passes 2 and 3, only the nonempty queues are shown in order to save space.)

Pass 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | nums Before Pass | queues After Step 1 | Front |  | Rear |  | nums  After Step 2 |
| [0] | 638**0** | (0) | 638**0** | 126**0** | 10**0** | [0] | 638**0** |
| [1] | 9**5** | (1) |  |  |  | [1] | 126**0** |
| [2] | 134**5** | (2) | 38**2** | 249**2** |  | [2] | 10**0** |
| [3] | 38**2** | (3) |  |  |  | [3] | 38**2** |
| [4] | 126**0** | (4) |  |  |  | [4] | 249**2** |
| [5] | 10**0** | (5) | 9**5** | 134**5** |  | [5] | 9**5** |
| [6] | 249**2** | (6) |  |  |  | [6] | 134**5** |
|  |  | (7) |  |  |  |  |  |
|  |  | (8) |  |  |  |  |  |
|  |  | (9) |  |  |  |  |  |

Pass 2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | nums Before Pass | queues After Step 1 | Front |  | Rear |  | nums  After Step 2 |
| [0] | 63**8**0 | (0) | 1**0**0 |  |  | [0] | 1**00** |
| [1] | 12**6**0 | (4) | 13**4**5 |  |  | [1] | 13**45** |
| [2] | 1**0**0 | (6) | 12**6**0 |  |  | [2] | 12**60** |
| [3] | 3**8**2 | (8) | 63**8**0 | 3**8**2 |  | [3] | 63**80** |
| [4] | 24**9**2 | (9) | 24**9**2 | **9**5 |  | [4] | 3**82** |
| [5] | **9**5 |  |  |  |  | [5] | 24**92** |
| [6] | 13**4**5 |  |  |  |  | [6] | **95** |

Pass 3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | nums Before Pass | queues After Step 1 | Front |  | Rear |  | nums  After Step 2 |
| [0] | **1**00 | (0) | 95 |  |  | [0] | **95** |
| [1] | 1**3**45 | (1) | **1**00 |  |  | [1] | **100** |
| [2] | 1**2**60 | (2) | 1**2**60 |  |  | [2] | 1**260** |
| [3] | 6**3**80 | (3) | 1**3**45 | 6**3**80 | **3**82 | [3] | 1**345** |
| [4] | **3**82 | (4) | 2**4**92 |  |  | [4] | 6**380** |
| [5] | 2**4**92 |  |  |  |  | [5] | **382** |
| [6] | 95 |  |  |  |  | [6] | 2**492** |

Pass 3:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | nums Before Pass | queues After Step 1 | Front |  | Rear |  | nums  After Step 2 |
| [0] | 95 | (0) | 95 | 100 | 382 | [0] | **95** |
| [1] | 100 | (1) | **1**260 | **1**345 |  | [1] | **100** |
| [2] | **1**260 | (2) | **2**492 |  |  | [2] | **382** |
| [3] | **1**345 | (6) | **6**380 |  |  | [3] | **1260** |
| [4] | **6**380 |  |  |  |  | [4] | **1345** |
| [5] | 382 |  |  |  |  | [5] | **2492** |
| [6] | **2**492 |  |  |  |  | [6] | **6380** |

**Assignment**

Implement the following methods in the RadixSort class.

* getMaxDigits – given an array of integers, returns the maximum number of digits of any value in the array. Some possible ways of finding the number of digits: use or convert to string and check the length
* getDigit– given an integer and digit location (*k*), returns the digit at the location *k* in the given integer. The least significant digit is indicated by a value of 0 for *k*. If *k* is greater than nth number of digits in the given integer, the getDigit returns 0. The table below shows several sample calls to getDigit.

|  |  |  |
| --- | --- | --- |
| **number** | **k** | **getDigit(number, k)** |
| 95 | 0 | 5 |
| 95 | 1 | 9 |
| 95 | 2 | 0 |

* itemsToQueues – corresponds to step 1 of each pass of the radix sort algorithm, creating the intermediate array list of ten queues. Each integer in the given array is inserted into the queue corresponding to the value of the digit currently being examined. If an integer does not have a digit at the given place value, 0 is assumed for that place value. The digit being examined is found in the *k*th position of the number. Integers are processed in the order in which they occur in the given array.
* queuesToArray – corresponds to step 2 of each pass of the radix sort algorithm, re-populating the given array from the given array of queues

Bonus:

Write a new class called RadixSortString that sorts an array of Strings using the Radix Sort algorithm. Tester and data file are provided. Hint: Pad the String with trailing spaces so they are all the same length. Trim them after the sort is finished.