CSC 2110: Program 3

Radix Sort

Description: In order to be sorted by Radix Sort, objects must have the following static method:

char getRadixChar(T* item, int index); //index is 1-based

This method accepts an object (T is the templated type) and an index. The sort field (a String) is obtained from the object and the character at the requested index is returned. The requested index is 1-based. If the index is out of range, a space (ASCII code 32) is returned. For writing getRadixChar in the CD class, T becomes CD and the method should be static.

Radix Sort: Radix Sort uses characters in the sort key to perform the sort. Radix Sort also utilizes "bins", which you can model with queues. The characters in the sort key can be digits, letters, or special characters. Sort using the convention that all non-digit and non-letter characters (any special characters) come first (place them in bin 0), followed by digits and then letters (convert uppercase to lowercase letters when sorting). Use the ASCII code of the digits and lowercase letters to determine in which bin to place the associated item. You will need to do some arithmetic to convert from ASCII code to queue index.

Using the method declarations below, implement Radix Sort using two algorithms. Both algorithms will require the use of bins, or an array of queues (of size 37 for digits, letters, and special characters). The sort parameter is the array containing the items to be sorted. The parameters num_to_sort and num_chars indicate the number of items to sort (starting with the first element in the sort array) and the number of characters to sort to.

- Algorithm 1: sort in ascending order using recursion (first character processed first)
- Algorithm 2: sort in descending order using a reverse loop (last character processed first)

```
   private:
      static void binSort(QueueLinked<T>* bin, int curr_char, int
num_chars, char (*getRadixChar) (T* item, int index));
      static void radixSortAsc(T** sort, int num_to_sort, int
num_chars, char (*getRadixChar) (T* item, int index));  //recursion,
uses binSort
      static void radixSortDesc(T** sort, int num_to_sort, int
num_chars, char (*getRadixChar) (T* item, int index)); //reverse loop

   public:
      static T** radixSort(T** sort, int num_to_sort, int num_chars,
bool asc, char (*getRadixChar) (T* item, int index));
```

Function Pointers: Note the use of function pointers to maximize the generality of the sorting algorithms as done in class for other sorting algorithms.

Templates: Your Radix Sort methods should be fully templated as done for the other sorting algorithms.

Radix Sort Driver: Write a driver (RadixSortDriver) to fully test both of your Radix Sort implementations. Make sure that duplicates are sorted in FIFO order.

Documentation: Fully document RadixSort:

- Preconditions and Postconditions
- Class level comments

Starting Files: Files I provide as a starting point can be found in Program 3 Files in the Common Files section on Piazza