CS3610 Project 4

Dr. Jundong Liu

In this project, you will write a program to simulate an external sorting algorithm. External sorting is used in scenarios in which the size of the dataset to be sorted exceeds the capacity of the main memory. In the first stage of external sorting, the dataset is divided into a set of smaller, unordered sublists. Each sublist is then transferred from disk to memory and sorted individually using an efficient internal sorting algorithm, such as quicksort. Finally, the sorted sublists are efficiently merged together on disk into a single sorted dataset.

To begin simulating external sort, you will first read an unsorted list of integers stored in file into an $n \times m$ matrix, where n represents the number of unordered sublists, and m is the number of integers comprising each sublist. Next, you will sort each of the n sublists individually using quicksort. To increase the chance that ensure that your quicksort would run in $O(m\log m)$ time in the worst case, you must implement a partition function with the pivot element set as the **median of the first, middle, and last elements** of the sublist given as input. For example, if the first, middle and last elements are 5, 9, and 6, the median of them would be 6. In the final stage, you will merge together the n sorted sublists into a single output list using a multiway merge that runs in $O(nm\log n)$ time. To achieve an $O(nm\log n)$ runtime, first create a min-heap from the first elements of each sorted sublist. (Consequently, you will have a min-heap of size n.) Next, extract the min element from the min-heap and place it into a 1D, sorted output list. Return back to the min-heap and replace the previously extracted min element with its successor from their originating, sorted sublist. In other words, you need to insert this successor into the min-heap.

If the min element was the largest in its sublist (meaning all the elements in this particular sublist have been merged into the output), move directly on to extracting the next min element in the min heap. Repeat this process until all sublists have been merged into the 1D, sorted output list. Rather than implementing a min heap from scratch, you may instead use the priority queue from the C++ queue library (`http://en.cppreference.com/w/cpp/container/priority_queue`), or you may use the standard heap functions in the C++ algorithm library (`http://en.cppreference.com/w/cpp/algorithm`).

The tricky part of this implementation is that, when you delete the min/root from the heap, you need to somehow remember which sublist it originally comes from, as you need to take its successor in the same sublist as the replacement. One way to achieve this would be: when you insert an number into the min-heap, associate it with its sublist ID and insert the combined structure. This combined structure can be implemented using *struct* or *STL map*, among others.

You have been provided a driver program that reads in the test input, makes calls to the quicksort and multiway merge functions, and outputs the final sorted results. Do not modify the main function and do not rewrite any of the existing function headers. You may however implement extra helper functions or data structs if you feel it necessary.

As stated in our CS 3610 syllabus, in all projects, homeworks and exams, it is

expected that the work you submit is your own. Do NOT copy code from the Internet or other sources. While you could work together with your classmates to develop an algorithm, make sure you implement the algorithm and the code by yourself.

Input

Input is read from the keyboard. The first line of input is the number of unordered sublists n. The second line is the number of integers m comprising each unordered sublist. The third line is an unordered, space-delimited list of $n * m$ integers.

Output

Output the externally sorted list of integers using space as a delimiter.

Sample Test Case

Use input redirection to redirect commands written in a file to the standard input, e.g. `$ ./a.out < input1.dat`.

Input 1

```
4
5
100 28 83 22 3 4 1 0 222 425 42 49 599 58 79 934 41 23 444 422
```

Output 1

```
0 1 3 4 22 23 28 41 42 49 58 79 83 100 222 422 425 444 599 934
```

Turn In

Submissions are through blackboard. If you have multiple files, package them into a zip file.

Grading

Total: 100 pts.

- **10**/100 - Code style, commenting, general readability.
- **05**/100 - Compiles.
- **05**/100 - Follows provided input and output format.
- **80**/100 - Successfully implemented external sort.