CS3610 Project 4

Dr. Jundong Liu

In this project, you will develop a program to simulate an *external sorting* algorithm. External sorting is used when the dataset size 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 loaded from disk to memory and sorted individually using an efficient internal sorting algorithm, such as quicksort. Finally, the sorted sublists are merged efficiently on disk to produce a single, fully 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 51, 98, and 67, the median of them would be 67. 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 one-dimensional, 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 one-dimensional, 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`).

An Analogy (based on information from the Forbes 400 list on April 4, 2023 `https://www.forbes.com/forbes-400/`)

You can imagine that we are trying to rank the richest people or families (more accurately, their wealth) in the US. Each sublist can be thought as a state, and thus we have 50 sublists (n = 50).

Firstly, we use quicksort to rank the individuals or families within each state. For quicksort(), you can modify and reuse the code from the textbook. Next, we take the wealthiest person or family from each of the 50 states to create a heap – one representative per state. This heap would include figures like Jeff Bezos from Washington, Mark Zuckerberg from California, Warren Buffett from Nebraska, Michael Bloomberg from New York, and Les Wexner from Ohio. The heap thus has a size of 50.

The root of this heap should be the wealth of the richest person/family in the entire country (i.e., Elon Musk's \$251B). Musk is from the state of Texas. After his number is removed from the heap and output to the output_list, you will need to find his successor – the 2nd richest person/family in Texas and insert his/her/their wealth (i.e., Alice Walton's \$55.7B) into the heap. Updating the heap, you will get the wealth of the 2nd overall richest person (i.e., Jeff Bezos' \$151B) on top of the heap. After Bezos' number is removed, you should go to his home state (Washington) to insert the next person's wealth (Bill Gates' number) into the heap.

As you need to remember which state each person comes from, **the elements in this heap should have a structure of (number, list_ID), e.g., (\$251B, Texas). This structure can be implemented using a *struct* or *STL map*, among others.** For the heap itself, you can use the priority queue class or heap functions available in C++ libraries, as I mentioned above. As for how to use a struct in priority queue, you can follow the examples under `http://www.geeksforgeeks.org/stl-priority-queue-for-structure-or-class/`.

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 they're 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 may collaborate with your classmates to develop an algorithm, make sure you implement the algorithm and the code by yourself.

Note: This project, along with the analogy, was first developed in 2017. The Forbes 400 list is used solely for illustration purposes.

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.
- **10**/100 - Compiles.
- **10**/100 - Follows provided input and output format.
- **20**/100 - Successfully implemented quicksort.
- **50**/100 - Successfully implemented k-way merge using a heap.