Assignment 2

Due Date: Sunday, March 19th 2017, 11:59pm

Objectives

The purpose of this assignment is twofold. First, I'd like to increase your familiarity with stacks, queues, and linked lists. Second, I would like to expose you to an idea for sorting data not discussed in class.

This assignment will challenge your implementation skills, building on the skills obtained in the last homework and developing them further. You will also have a tricky sub-problem to tackle. How do you combine runs efficiently? This will challenge you to think creatively.

Deliverables

Submit two files.

The file named "`zigzag.h`" is your header file. Place all function declarations in this header with comments on how you designed the header. You may use the header I provide or design your own. In either case, include a header.

The file named is "`zigzag.c`" is your source file. Implement all functions defined in the header file. Include a main function in this file.

All input should be read from a file named "`zigzag.txt`". The input specification will define precisely what is allowed for input to your program. You need not guard against cases not allowed in the input.

Scoring

I will run your program multiple times on multiple test files. Your grade will be awarded based on the number of tests you pass with some points allocated to fulfilling the special requirements for sorting. Be sure to test your program on more than just the sample cases provided.

To guard against infinite loops and inefficient implementations, your program will be run for x seconds per test case. Upon grading, you will receive a "`results.txt`" file detailing which tests where passed and which failed. Here is a table of possible results:

Error Code	Meaning
AC	Accepted! :D All is wonderful in the world and your code worked correctly.
WA	Wrong Answer: The code executed but produced faulty output.
RTE	Runtime Error: The code crashed spewing memory and guts everywhere.
TLE	Time Limit Exceeded: I killed the program after x seconds. I feel betrayed.
CTE	Compile Time Error: Your code did not compile. ☹

Zigzag

`filename: zigzag`
`timelimit: 3 seconds`

In class, we discussed numerous techniques for sorting data. In this problem, we will use a clever idea to speed up the best case in sorting algorithms. With any lucky, we will also keep the worst case of sort as $O(n \log n)$. (At the very least, that is my challenge to you.)

In sorting, we explored ordering sequences of integers. In particular, the merge sort algorithm took two sorted sequences and combined them as a sub problem to sorting. But often in real world data, pieces of the data are partially sorted. Observe:

$$1, 3, 7, 100, 8, 100, 200, 7, 8, 9, 10$$

Notice we can split the sequence into three contiguous regions where the data is already sorted:

$$1, 3, 7, 100, 8, 100, 200, 7, 7, 8, 9, 10$$

Aha! Now we can form a modified merge sort were we can avoid sorting already sorted data. Notice that because we are decomposing the data into contiguous *runs*, the algorithm runs best case in $O(n)$ on already sorted data.

But our algorithm has a weakness, an Achilles heel of sorts. By making the runs strictly **decreasing**, everything is a run! Uh oh!

$$5, 4, 3, 2, 1$$

We can solve this problem as well. When identifying runs, we can look at the first two numbers to determine if the run will be increasing or decreasing. From then on, continue adding the next numbers to the run until it changes direction. Now we have a solution that works in $O(n)$ best case on data sorted in reverse order! Observe:

$$5, 4, 3, 2, 1$$

It also improves its average runtime on cases that zigzag back and forth:

$$1, 2, 3, 4, 5, 4, 3, 2, 1, 8, 9, 10, 3, 2$$

Though it's probably prudent to put the reverse runs in sorted order first, for easy combination:

$$1, 2, 3, 4, 5, 1, 2, 3, 4, 8, 9, 10, 2, 3$$

So now your challenge is: combine these sorted runs of data efficiently to sort all the data. Keep in mind what happened in class with our "dumb" merge sort that ran in $O(n^2)$. Try to find a way around that behavior. What other types of cases might be tricky for such an approach?

Oh, and one more thing. When I said check the direction based on the first two elements in the run, I lied. That will work in most cases, but won't work here:

$$4, 4, 4, 7, 8, 9, 4, 4, 4, 3, 2, 1$$

The above sequence should be decomposed into two runs:

$$4, 4, 4, 7, 8, 9, \; 4, 4, 4, 3, 2, 1$$

Additional Requirements

- You must read the data into a linked list data structure and sort that structure. Don't use an array!
- When reversing runs, don't uses recursion. Use a stack instead!

Input "zigzag.txt"

The first line of the input file contains a single integer n ($1 \leq n \leq 10^5$) representing the number values in the sequence of data.

This is followed by a line with n integers v_i ($1 \leq v_i \leq 10^9$), separated by spaces, representing the data to be sorted

Output (stdout)

Output n integers in sorted order using the algorithm described in the specification.

Sample 1

Input	`5`
	`5 2 3 4 1`

Output	`1 2 3 4 5`

Sample 2

Input	`10`
	`2 3 3 10 9 8 7 8 9 10`

Output	`2 3 3 7 8 8 9 9 10 10`

Tips

Try getting a version that works in $O(n^2)$ time first. Such a solution will decompose the data into runs, but won't combine them in a smart way. What kinds of cases cause your algorithm to run in $O(n^2)$? If you cannot get the program to work faster than $O(n^2)$, turn that version in! You may pass a subset of the tests and still get a good grade.

I recommend making a suite of tests for your program. Try making tests that cause your approach to be slow. Is your algorithm fast on other types of data? It is surely better than bubble sort! Write comments in your code detailing what kinds of data is fast versus slower data.

If something in the problem appears unclear, please come see me or send me an email. I'll do my best to clarify either the specification or provide further clarification in class or through WebCourses.

Don't use the whole header file immediately. Only copy in what you need while you're getting things working. Use it as a guide for how to design your program.

Checkpoints

This problem requires several major steps. When dealing with a larger project, it is best to make checkpoints for yourself. Try to set a schedule for when you can expect each piece to be complete. Do your best to meet each milestone.

- (02/24) Get a working stack and queue.
- (03/03) Work on reading in the data and identifying runs. Reverse each of the decreasing runs using a stack.
- (03/10) Use the ideas from Mergesort to combine runs together. Get a working sort algorithm even if it is slow.
- (03/19) Hopefully you have identified the slow cases. Attempt to find a smart way to handle the merging efficiently.

If you don't break up the work like this or set your own milestones, you will fail to complete this project. The trick is to break things into small manageable chunks and get pieces working systematically. Use what you learned from Assignment 1 about testing small blocks of your code before moving on to more complex sub-problems.