CS 392: Homework Assignment 3
Due: March 8, 11:55pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

1 Objective: Out-of-core Merge Sort

Create a program that implements the merge sort algorithm without storing all the data in memory. This will be accomplished by using files to store intermediate data. Note that for the merging step of merge sort, only the current element of each of the two arrays needs to be available at a given time.

Your program should be called `oocmerge` and take two mandatory command line arguments:

```
oocmerge <N> <output filename>
```

N is the number of floats to be sorted, which will be randomly generated. The output filename is of the file that will store the sorted array of floats (in binary format).

This algorithm is useful in practice for sorting massive amounts of data that cannot fit in RAM. (It would be combined with an in-core algorithm like selection sort in practice, but this is out of scope here.)

2 Requirements

1. The program should begin by generating N floating point numbers between -100 and 100 using `rand()`. Floats can be generated by dividing the output of `rand()`, which is an integer, by `RAND_MAX` and then transforming the result appropriately. Do not generate integer values only. These data should be written to a file, since this is an out-of-core algorithm.

2. Assume N is positive and less than 1000 and that sorting will be in ascending order.
3. The program should NOT allocate any arrays statically or dynamically. All data should be held in files in binary format. Allocating memory for strings (filenames) is allowed.
4. Files and directories should be manipulated using systems calls: `open()`, `read()`, `wite()`, etc.
5. The program should create a directory named temp. The permissions of this directory should be changed so that it is completely inaccessible to anyone but the user.
6. Temporary files storing intermediate arrays to be merged should be written in the above directory. To avoid mistakes, the permissions of these files should be changed so that they can only be written when they are used as output and they can only be read from when they are used as input. Anyone but the user should not have access to them.
7. Temporary files should be deleted when they are no longer needed.
8. The program should end by verifying that the numbers in the output file are indeed sorted. This should be done also without storing the more than the minimum amount of data necessary for the verification in RAM. (Reading two consecutive numbers from the file at a time to test that they are in the right order, and looping through the entire file would work.) To assist grading, the program should also print the contents of the sorted final file.

3 Hints

1. Worry about permissions after your code works, but do not forget to implement this part even if your code does not completely work.
2. I would start by implementing the flow of the algorithm without the actual sorting. Print useful messages for debugging, such as: ”Round R: merging arrays A and B.”
3. See how to use `sprintf()` to generate the filenames of the temporary files. For example `sprintf(fname, "array_round%d_split%d.dat", 5, 0);`
4. There is a recursive and an iterative implementation. If you pick the former, make sure that it terminates. See second hint and start with 8 numbers.
5. Think about what the split step actually does.

Deliverables A zip file containing:

1. Sources files, including at least one header file.
2. A makefile.