Project 1: MergeSort & Your Customers

Submission deadline: 9pm on 9/20, 2022
(6% of the total grade + 1% extra credit)

This assignment consists of two parts – Part A and Part B

Part A (3% of the total grade): Implement the MergeSort algorithm that sorts an array of arbitrary records. More specifically, you need to implement the body of

```
void msort(void* base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

The arguments to msort() are identical to those of qsort() in the C runtime library (i.e., read the man page of qsort( )). base holds the input elements of size nmeb (unsorted), and after msort(), the elements in base need to be sorted. Like qsort(), this function allows sorting array elements of any type (e.g., built-in or custom type) as long as one provides the compar() function for comparing two elements. Here is some sample code that shows the usage of msort().

```
struct student {int id; char name[100]; int score};
struct student unsorted1[100], unsorted2[100];
…
int compare_id(const void*A, const void *B) {
    const struct student *pA = (const struct student *)A,
                         *pB = (const struct student *)B;
    return (pA->id > pB->id) ? 1 : ((pA->id < pB->id) ? -1: 0);
}
int compare_score(const void*A, const void *B) {
    const struct student *pA =(const struct student *)A,
                          *pB = (const struct student *)B;
    return (pA->score > pB->score) ? 1 : ((pA->score < pB->score) ? -1: 0);
}
…
msort(unsorted1, 100, sizeof(struct student), compare_id);    // sort by the id
msort(unsorted2, 100, sizeof(struct student), compare_score); // sort by the score
…
```

Logistics for Part A:

- Write the code in C – your code should compile with gcc installed on eelab5 or eelab6
- We provide "msort.c", "msort.h", "test.c", "dist,c"(Part B) "Makefile" in the KLMS page. All you need to do for Part A is to update and submit "msort.c" that includes the complete implementation of msort() described above.
- You can use "test.c" for your own testing & debugging. The current version accepts integers from stdin, sorts the input by msort() and qsort(), and compare the results by printing each element.
  The following commands should build "test", and feed in 100 random integers to "test"

  ```
  $ make
  $ shuf -i 0-10000000 -n 100 | ./test
  ```

- Feel free to modify "test,c" for your own testing and debugging. Note that msort() should work for an input array of an arbitrary custom type.
- We will test your submitted "msort.c" by compiling and linking it to our own test program, so don't assume "test.c" is all that you need to test with.

Part B (3% of the total grade + 1% of extra credit): Every morning, you deliver milk to your customers that live in the same region. Since all of your customers live in the same region, you consider moving to that region as you frequently visit the customers. The address of each customer is different only by the address number, and the distance between two addresses, x and y, is |x - y|. Write a program that reveals the ideal address where you want to move to minimize the sum of distances from your new home. The input is just one line followed by EOF. The first value in the input is the number of customers, and the following numbers represent the address of each customer. For example,

3 4 2 6

says that you have 3 customers, and one customer lives at address 4, 2, 6 each. The output should print out the minimal sum of the distances from the optimal moving location of your home. The program should be named as "dist", so when you run

```
$ ./dist
3 4 2 6
4
```

$./dist < data1 //where data1 has "2 2 4" in it

2

Logistics for Part B:

- Write the code in C – your code should compile with gcc installed on eelab5 or eelab6
- For generality, assume that the total number of customers can be big but smaller than 1,000,000, and the address of any customer is between 1 and 10,000,000. Multiple customers may live at the same address (e.g., sublet, apartment, etc.)
- Your program should print out the result to stdout. Print out one number (min sum of distances) followed by '\n',, like printf("%ld\n", sum);
- Use msort() if you need sorting. If your msort() doesn't work, you can use "msort.c" that we provide at KLMS (that calls qsort() internally).
- Your code should finish in a reasonable amount of time – you'll get penalty if your algorithm is brute-force (e.g., pick the minimum from all possible sums).

- Start with the downloaded “dist.c” from the KLMS page. Edit the code and submit your own “dist.c”.

Collaboration policy:

- You can discuss algorithms and implementation strategies with other students, but you should write the code on your own. That is, you should NOT look at the code of anyone else.
- You can use the code in the textbook or in the slides, but you are NOT allowed to look up the Internet for the merge sort algorithm. You can study the usage of qsort() or command line processing, etc. in the Internet, though.

Submission:

- You need to submit 3 files – your ”msort.c” (part A), “dist.c” (part B), and “readme” in a tarred gzipped file with the name as YourID.tar.gz. If your student ID is 20211234, then 20211234.tar.gz should be the file name.
- ‘make’ should build both binaries, “test” and “dist”.
- You will get **ZERO** point for each part if “msort.c” or “dist.c” doesn’t compile (due to compiling errors). Make sure that your code compiles before submission.
- Suppress all warnings at compiling by turning on the “-W -Wall” options in the CFLAGS (see Makefile). You will get **some penalty** if gcc produces *any* warnings.
- “readme” (a text file with “readme” as the file name) should contain the followings. (1) Explain how msort() works briefly (2) Explain your algorithm for part B. (3) (Optional) prove why your algorithm is optimal (**extra credit of 1%**). (3) List all collaborators with whom you discussed with in this file.

Grading:

- We will evaluate the accuracy for both Part A and Part B.
- We may give penalty if dist is too slow (e.g., brute-force) for a large input.
- We will evaluate your readme file
- We will evaluate the clarity of your code – coding style, comments, etc.