# ECE36800 Programming Assignment #2

*Due Sunday, July 2nd, 2023, 11:59pm*

**Description**

This project is to be completed on your own. You will implement the following two sorting algorithms.

- Mergesort
- Quicksort

The goal is to have efficient implementations of both algorithms.

**What you are given**

You will be given two files, `sorting.c` and `sorting.h`. Do *not* modify the `sorting.h` file. You are not submitting the `sorting.h` file and any changes you make will not be reflected when we compile your code.

**Functions to complete**

For this assignment, you will complete the following functions.

- `void mergeSort(long* arr, int size)`
- `void merge(long* src, long* dest, int lb, int mid, int ub)`
- `void quickSort(long* arr, int size)`
- `int partition(long* arr, int lb, int ub)`

If you need additional helper functions, it is best to declare these helper functions as static. Do not name these help functions with a prefix of two underscores "__". You may assume every array a function is given as input will have a length less than `INT_MAX`.

Do not modify the provided `sorting.h` file because you are not submitting it. Any modifications you have made to the provided `sorting.h` will not be reflected in the `sorting.h` file that we use to evaluate your submission.

The `merge` function accepts two arrays of `long` s and 3 integers denoting start and endpoints of two subarrays in `src`. You may assume the arrays are non-`NULL` and have the same size and type. However, you may not assume they have the same contents. We break these arrays into two subarrays. The left subarray starts at index `lb` and ends at index `mid`, `src[lb, mid]`. The right subarray starts at index `mid+1` and ends at `ub`, `src[mid+1, ub]`. If the two subarrays are sorted in ascending order in `src`, the `merge` function merges them together in the `dest` array. In otherwords,

 

the merge function takes two sorted subarrays, src[lb, mid] and src[mid+1, ub] and merges them together to form one larger sorted subarray dest[lb, ub]. If the two subarrays (src[lb, mid] and src[mid+1, ub]) are not sorted, behavior is undefined. If ub $\leq$ lb, the function should return without making any changes to the arrays. The function should also return without making any changes if mid $<$ lb or mid+1 $>$ ub.

The mergeSort function accepts an array of long s and the size of the array. The function should perform Mergesort on the given array. If you wish to perform a recursive version of Mergesort, you will have to create a helper function. While performing Mergesort, you should make calls to the merge function. Your version of Mergesort should be compatible with any version of the merge function, as long as it meets the criteria listed above. When grading, we should be able to replace your version of merge with our version of merge, and no functionality should be lost.

The partition function accepts an array of long s and 2 integers denoting the start and endpoint of a subarray, arr[lb, ub]. If ub < lb, your function should return -1. This function should choose an element in the subarray and use it as the partition element (partition_el). When finished executing, the partition function should return the partition index (partition_idx). Every element to the left of the partition index should be less than or equal to the partition element and every element to the right of the partition index should be greater than the partition element. In otherwords, after a call to partition each of the following is true:

- arr[i] $\leq$ arr[partition_idx], such that i $\in$ [lb, partition_idx - 1]
- arr[i] $>$ arr[partition_idx], such that i $\in$ [partition_idx + 1, ub]
- arr[partition_idx] = partition_el

The quickSort function accepts an array of long s and the size of the array. The function should perform Quicksort on the given array. If you wish to perform a recursive version of Quicksort, you will have to create a helper function. While performing Quicksort, you should make calls to the partition function. Your version of Quicksort should be compatible with any version of the partition function, as long as it meets the criteria listed above. When grading, we should be able to replace your version of partition with our version of partition, and no functionality should be lost.

**Other functions you may want to write**

You will definitely want to write a main function to test your program. However, **do not include a main function in** sorting.c. This will prevent us from compiling your code. You should create a separate file that has your main function then you should #include "sorting.h" in that file.

You should also write functions that create test cases for your code. I would recommend making use of the rand() function (https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm). I would also think about edge cases such as potential worst cases and best cases.

It will also be helpful to write a function that checks if an array is sorted or unsorted. This will help you test your code to make sure it is functional.

| Function | Weight |
|---|---|
| `mergeSort` | 35% |
| `quickSort` | 35% |
| `merge` | 15% |
| `partition` | 15% |
| Total | 100% |

Table 1: Grading Breakdown

The function `void printArr(long* arr, int size)` is defined in `sorting.h`. This function is completely optional. You do *not* have to implement it and it will *not* be graded. It is simply there to help you test things. It is defined in `sorting.h` so that you are able to access it from other files (such as a file containing your `main` function).

**Submission and Grading**

To submit the assignment, upload your version of `sorting.c` to Brightspace. Do *not* zip the file, do *not* upload other files, just your version of `sorting.c`. The grading breakdown can be viewed in Table 1. If we detect any memory errors during the execution of a test case, you will lost 50% for that test case.

**Compiling**

We will use the following command to compile your code:

```
gcc -std=c99 -Wall -Wshadow -Wvla -pedantic -O3 sorting.c pa2.c -o pa2
```

`pa2` is the file containing the `main` function. Remember, you are not providing this file, just the `sorting.c` file.

Good luck and happy coding!