UTD, CS 4348 Operating Systems
Multithreaded Programming Project
Sorting Application using Threads
Due Date: Thursday, July 03, 2025
100 Points

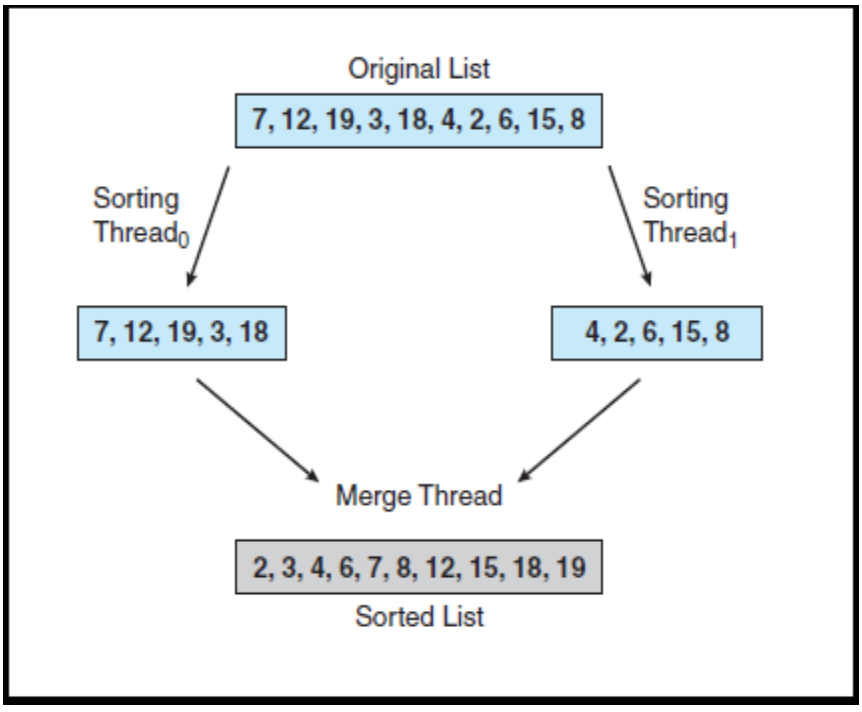
In this project, you should work in a group of two students. You will learn and practice developing a multithreaded application using **either Java or C/C++**.

Description:

Task #1: Writing a multithreading sorting function:

Write a sorting function that sorts a list (i.e., an array) of size N containing integer values. This list is divided into two smaller sublists of equal size. Two separate threads should be created (referred to as *sorting threads*), each of which will sort one of the sublists using any sorting algorithm (such as Bubble Sort). Then, a third thread should be created (referred to as the *merging thread*) to merge the two sorted sublists produced by the sorting threads into a single sorted list using the Merge Sort algorithm.

Because global data is shared across all threads, perhaps the easiest approach is to create a global array to hold the list of integer values. Each sorting thread will work on one half of this array. The merging thread will then merge the two sorted sublists into a second global array of the same size. Graphically, the structure of this function is illustrated below.



This function requires passing parameters to each of the sorting threads. Specifically, it is necessary to identify the starting and ending indices of the elements in the global array that each thread is responsible for sorting.

Task #2: Writing a non-threading sorting function:

Write a second version of the sorting function that does not use threads. This function should sort each half of the array using the same sorting algorithm used in the multithreaded version (from Task #1) to produce two sorted sublists. It should then merge these sublists using the merge sort algorithm.

Task #3:
Define a global variable N that will be assigned a different value in each run of your program. This variable represents the size of the list (i.e., the array) that needs to be sorted.
In the main function of your program, generate N random integer values in the range [1, 10,000] and store them in the unsorted array.

The main function will first call the function defined in Task #1 to sort the array using multithreading and measure the time taken to complete the sorting process. Then, it will call the function defined in Task #2 to sort the array using the non-threaded method and also measure the time taken for that sorting process.

Example of running this program (named myproj.c):

Suppose you defined N to be 1000 at the top of your program as follows:

#define N 1000

Compile your program as follows:

$ gcc myproj.c -o myproj

Then run your program as follows:

$ ./myproj
You may get the following output:

Sorting 1000 integers is done in 10.0 msec using threads
Sorting 1000 integers is done in 20.0 msec without-threads

The numbers 10.0 and 20.0 are just examples! Your actual numbers will be different.

You must run your program four times, each time with a different value of N. You must use the following values of N:

1000, 5000, 10000, and 20000

Collect the resulting output from each run in a table (one row for each run), as shown in the following sample output table:

Sample output table:

N	Using Non-Threading	Using Multithreading
1000	20 msec	10 msec
5000	60 msec	25 msec
10000	95 msec	40 msec
20000	140 msec	90 msec

The results in your table may be completely different from results in this table

What to submit:

- The Source code of your program:
- How to run it
- The results produced from each run in a table as shown above.