# Submission Report

* Submission generated at 10/26/2025 at 22:33:25
* Machine info: Linux runnervmwhb2z 6.11.0-1018-azure #18~24.04.1-Ubuntu SMP Sat Jun 28 04:46:03 UTC 2025 x86\_64 x86\_64 x86\_64 GNU/Linux

## Note to Students

Please read this report carefully before submission. Ensure that all sections are complete and accurate. Look for any errors in the build or test outputs. If you find any issues, correct them before submitting. Post any questions on the class discussion board for help.

## README

# Project X

* Name: Bowen Moser
* Email: bowenmoser@u.boisestate.edu
* Class: 452-002

## Known Bugs or Issues

No known bugs or issues at this time

## Experience

This was an enjoyable but challening project. Since this is my second time ever writing a multithreaded program, I referred back to my previous program to help remember how to structure the thread creation and joining of my threads back into the main thread. I did use Claude for ideas/help and for writing my tests since it has been helpful in the past when I encountered roadblocks.

I did a bit of research into the POSIX thread standard and found barriers. I worked on implementing one to try and ensure thread synchronization after all threads had succesfully merged their portion of the array. I’m not sure if it was really necessary, but I think it was a nice way of learning another concept and applying it to my program. My thought process was if I implement a barrier, I can make sure threads proceed to joining in an “orderly” fashion with an added layer of control to their behavior, especially since it’s hard (or practically impossible) to verify with just unit tests.

pthread\_barrier doc: https://www.gnu.org/software/gnuastro/manual/html\_node/Implementation-of-pthread\_005fbarrier.html

## Analysis

Overall, my graph doesn’t match the example one as threads increase. I’ve tried a few different things but it sort of stagnates as threads increase, rather than increase in time as the number of threads goes up. I think this might be one or more factors. The first is that I’m only locking when the main thread merges all other threads together. So it’s really just dependent on the main thread merging rather than each thread sorting and context switching between each one. I’ve tried to move the mutex lock around but the results have been essentially the same, so I’m not sure where else to implement the lock. The other is it may have something to do with my barrier causing all threads to wait until each one is done? Again, I’ve tried a few different places to implement the barrier but I think it makes the most sense where it is currently used. That being said, my results are pretty consistent (running on Onyx) and the program is fastest between 14 and 16 threads before it begins to stagnate.

Were you able to generate something close to what the example showed? Why or why not?

Yes and no, the graph does start off by showing my program does run faster and faster as the number of threads increase. However, as the number of threads continue to increase, the graph does not begin to substantially rise in terms of run time. Instead it begins to stagnate across the y axis. While this does indicate that there is a slowdown that correlates with the thread count and my program efficieny, it does not increase greatly in terms of my run time as we got closer to thirty two threads.

Did you see a slowdown at some point? Why or why not?

From my graph, I do see an eventual slowdown. Generally the slowdown appears after about 20 to 25 threads. As seen in the graph, the total amount of time taken begins to plateau after this 20 to 25 thread mark. I think this is because my main thread is really the one being tasked with doing all of the merging, and is the only place a mutex lock is being used. I identified the merging portion as a critical section of code and implemented this lock to protect the sorting action.

Did your program run faster and faster when you added more threads? Why or why not?

Yes my program does run faster and faster as the number of threads for my program increases. This is because I am partitioning the array into an evenly distributed amount of sections that is based on how many threads I entered from main. I tried to replicate the second approach we discussed in class last week. Since I’m able to distribute smaller sections of the total array to each thread, each thread has to sort less of the total array. This is a divide and conquer approach that I think works very well in this scenario where we are giving the program a large, unsorted array to handle. The main thread then handles the final merge of each section of the list, returning it as one sorted array of integers.

What was the optimum number of threads for your machine?

The optimum number of threads for my machine (Onyx) has been consistenly between 14 and 16 total threads.

What was the slowest number of threads for your machine?

Technically the slowest number of threads is when you just have one. But the law of diminishing returns begins to take effect after 20 or so threads. The graph fluctuated slightly between script runs but that may have to do more with Onyx resource utiliziation constantly changing which is making minor changes to the graph output.

## Example Image

## Build Output

This section was generated by running make all in the project root directory.

make[1]: Entering directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/lab.c -o build/debug/lab.c.o  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/main.c -o build/debug/main.c.o  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address build/debug/lab.c.o build/debug/main.c.o -o build/debug/myapp\_d -pthread -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
make[1]: Entering directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
mkdir -p build/release  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion -D\_REENTRANT -c src/lab.c -o build/release/lab.c.o  
mkdir -p build/release  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion -D\_REENTRANT -c src/main.c -o build/release/main.c.o  
src/main.c: In function ‘main’:  
src/main.c:47:29: warning: conversion from ‘size\_t’ {aka ‘long unsigned int’} to ‘unsigned int’ may change value [-Wconversion]  
 47 | mergesort\_mt(A, listSize, num\_threads);  
 | ^~~~~~~~~~~  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion -D\_REENTRANT build/release/lab.c.o build/release/main.c.o -o build/release/myapp -pthread  
make[1]: Leaving directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
make[1]: Entering directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
mkdir -p build/tests  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c src/lab.c -o build/tests/lab.c.o  
mkdir -p build/tests  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c src/main.c -o build/tests/main.c.o  
mkdir -p build/tests/harness/  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c tests/harness/unity.c -o build/tests/harness/unity.c.o  
mkdir -p build/tests/  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c tests/lab-test.c -o build/tests/lab-test.c.o  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage build/tests/lab.c.o build/tests/main.c.o build/tests/harness/unity.c.o build/tests/lab-test.c.o -o build/tests/myapp\_t -pthread -fprofile-arcs -ftest-coverage  
make[1]: Leaving directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
make[1]: Entering directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
mkdir -p build/debug-test  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c src/lab.c -o build/debug-test/lab.c.o  
mkdir -p build/debug-test  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c src/main.c -o build/debug-test/main.c.o  
mkdir -p build/debug-test/harness/  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c tests/harness/unity.c -o build/debug-test/harness/unity.c.o  
mkdir -p build/debug-test/  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c tests/lab-test.c -o build/debug-test/lab-test.c.o  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address build/debug-test/lab.c.o build/debug-test/main.c.o build/debug-test/harness/unity.c.o build/debug-test/lab-test.c.o -o build/debug-test/myapp\_td -pthread -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/CS\_452\_P3\_Multithreaded\_Application/CS\_452\_P3\_Multithreaded\_Application'  
Builds completed. You can run the application with: ./build/release/myapp  
You can run the debug build with: ./build/debug/myapp\_d  
You can run the test build with: ./build/tests/myapp\_t  
You can run the debug-test build with: ./build/debug-test/myapp\_td

## Coverage Report

This section was generated by running make report in the project root directory.

Setting up test...  
Tearing down test...  
tests/lab-test.c:100:test\_basic\_sort:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:101:test\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:102:test\_reverse\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:103:test\_array\_with\_duplicates:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:104:test\_single\_element:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:105:test\_different\_thread\_counts:PASS  
  
-----------------------  
6 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
Setting up test...  
Tearing down test...  
tests/lab-test.c:100:test\_basic\_sort:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:101:test\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:102:test\_reverse\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:103:test\_array\_with\_duplicates:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:104:test\_single\_element:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:105:test\_different\_thread\_counts:PASS  
  
-----------------------  
6 Tests 0 Failures 0 Ignored   
OK  
mkdir -p ./build/report/html  
mkdir -p ./build/report/txt  
gcovr -r . --html --html-details --exclude-directories build/tests/harness --exclude '.\*main\.c$' --exclude '.\*test\.c$' -o ./build/report/html/coverage\_report.html  
(INFO) Reading coverage data...  
  
(INFO) Writing coverage report...  
  
gcovr -r . --txt --exclude-directories build/tests/harness --exclude '.\*main\.c$' --exclude '.\*test\.c$'  
(INFO) Reading coverage data...  
  
(INFO) Writing coverage report...  
  
------------------------------------------------------------------------------  
 GCC Code Coverage Report  
Directory: .  
------------------------------------------------------------------------------  
File Lines Exec Cover Missing  
------------------------------------------------------------------------------  
src/lab.c 110 93 84% 174,185-186,193-194,217,219-220,222-223,225-226,235,257,267,269-270  
------------------------------------------------------------------------------  
TOTAL 110 93 84%  
------------------------------------------------------------------------------

## Address Sanitizer Report

This section was generated by running make leak-test in the project root directory.

Setting up test...  
Tearing down test...  
tests/lab-test.c:100:test\_basic\_sort:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:101:test\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:102:test\_reverse\_sorted\_array:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:103:test\_array\_with\_duplicates:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:104:test\_single\_element:PASS  
Setting up test...  
Tearing down test...  
tests/lab-test.c:105:test\_different\_thread\_counts:PASS  
  
-----------------------  
6 Tests 0 Failures 0 Ignored   
OK

## Src Files

### lab.c

#define \_GNU\_SOURCE  
#include <stdlib.h>  
#include <stdio.h>  
#include <sys/time.h> /\* for gettimeofday system call \*/  
#include <string.h>  
#include <pthread.h>  
#include "lab.h"  
  
// Global mutex for synchronizing merge operations  
static pthread\_mutex\_t merge\_mutex = PTHREAD\_MUTEX\_INITIALIZER;  
static pthread\_barrier\_t sort\_barrier;  
  
/\*\*  
 \* @brief Standard insertion sort that is faster than merge sort for small array's  
 \*  
 \* @param A The array to sort  
 \* @param p The starting index  
 \* @param r The ending index  
 \*/  
static void insertion\_sort(int A[], int p, int r)  
{  
 int j;  
  
 for (j = p + 1; j <= r; j++)  
 {  
 int key = A[j];  
 int i = j - 1;  
 while ((i > p - 1) && (A[i] > key))  
 {  
 A[i + 1] = A[i];  
 i--;  
 }  
 A[i + 1] = key;  
 }  
}  
  
/\*\*  
 \* @brief Sorts an array of ints into ascending order using the constant  
 \* INSERTION\_SORT\_THRESHOLD internally  
 \*  
 \* @param A A pointer to the start of the array  
 \* @param p The starting index  
 \* @param r The ending index  
 \*/  
void mergesort\_s(int A[], int p, int r)  
{  
 if (r - p + 1 <= INSERTION\_SORT\_THRESHOLD)  
 {  
 insertion\_sort(A, p, r);  
 }  
 else  
 {  
 int q = (p + r) / 2;  
 mergesort\_s(A, p, q);  
 mergesort\_s(A, q + 1, r);  
 merge\_s(A, p, q, r);  
 }  
}  
  
/\*\*  
 \* @brief Merge two sorted sequences A[p..q] and A[q+1..r] and place merged  
 \* output back in array A. Uses extra space proportional to  
 \* A[p..r].  
 \*  
 \* @param A The array to merge into  
 \* @param p The starting index of the first half  
 \* @param q The middle  
 \* @param r The ending index of the second half  
 \*/  
void merge\_s(int A[], int p, int q, int r)  
{  
 // converting to size\_t to fix warnings  
 size\_t size = (size\_t)(r - p + 1);   
 int \*B = (int \*)malloc(sizeof(int) \* size);  
  
 int i = p;  
 int j = q + 1;  
 int k = 0;  
 int l;  
  
 /\* as long as both lists have unexamined elements \*/  
 /\* this loop keeps executing. \*/  
 while ((i <= q) && (j <= r))  
 {  
 if (A[i] < A[j])  
 {  
 B[k] = A[i];  
 i++;  
 }  
 else  
 {  
 B[k] = A[j];  
 j++;  
 }  
 k++;  
 }  
  
 /\* now only at most one list has unprocessed elements. \*/  
 if (i <= q)  
 {  
 /\* copy remaining elements from the first list \*/  
 for (l = i; l <= q; l++)  
 {  
 B[k] = A[l];  
 k++;  
 }  
 }  
 else  
 {  
 /\* copy remaining elements from the second list \*/  
 for (l = j; l <= r; l++)  
 {  
 B[k] = A[l];  
 k++;  
 }  
 }  
  
 /\* copy merged output from array B back to array A \*/  
 k = 0;  
 for (l = p; l <= r; l++)  
 {  
 A[l] = B[k];  
 k++;  
 }  
 free(B);  
}  
  
/\*\*  
 \* @brief The function that is called by each thread to sort their chunk  
 \*  
 \* @param args see struct parallel\_args  
 \* @return void\* always NULL  
 \* AI Use: Assisted by Claude  
 \*/  
void \*parallel\_mergesort(void \*args)  
{  
 struct parallel\_args \*pargs = (struct parallel\_args \*)args;  
 size\_t start = pargs->start;  
 size\_t end = pargs->end;  
   
 if (start < end) {  
 mergesort\_s(pargs->A, (int)start, (int)end);  
 }  
   
 //Using barrier to sync threads after each sort, threads wait here until all have sorted their array portion before proceeding  
 pthread\_barrier\_wait(&sort\_barrier);  
   
 return NULL;  
}  
  
/\*\*  
 \* @brief Sorts an array of ints into ascending order using multiple  
 \* threads  
 \*  
 \* @param A A pointer to the start of the array  
 \* @param n The size of the array  
 \* @param num\_threads The number of threads to use.  
 \* AI Use: Assisted by Claude  
 \*/  
void mergesort\_mt(int \*A, size\_t n, unsigned int num\_threads)  
{  
 int barrier\_initialized = 0;  
 int ret = 0;  
  
 // Handling egde case where array size < num threads  
 if (n < num\_threads) {  
 // If array is smaller than thread count, use only as many threads as elements  
 num\_threads = (unsigned int)n;  
 }  
   
 if (num\_threads > MAX\_THREADS) {  
 // forcing to max threads if necessary  
 num\_threads = MAX\_THREADS;  
 }  
   
 if (num\_threads <= 1 || n <= INSERTION\_SORT\_THRESHOLD) {  
 mergesort\_s(A, 0, (int)(n - 1));  
 return;  
 }  
   
 // Initializing a barrier for my threads   
 ret = pthread\_barrier\_init(&sort\_barrier, NULL, num\_threads);  
 if (ret != 0) {  
 fprintf(stderr, "Failed to initialize barrier: %s\n", strerror(ret));  
 return;  
 }  
 barrier\_initialized = 1;  
  
 // Create thread arguments array  
 struct parallel\_args \*thread\_args = malloc(sizeof(struct parallel\_args) \* num\_threads);  
 if (thread\_args == NULL) {  
 fprintf(stderr, "Memory allocation failed for thread arguments\n");  
 return;  
 }  
  
 // Calculate chunk size for each thread  
 // Using second approach style from class with added remainder handling  
 size\_t chunk\_size = n / num\_threads;  
 size\_t remainder = n % num\_threads;  
 size\_t current\_pos = 0;  
  
 // Create threads and assign work  
 size\_t threads\_created = 0;  
 for (size\_t i = 0; i < num\_threads; i++) {  
 thread\_args[i].A = A;  
 thread\_args[i].start = current\_pos;  
 thread\_args[i].end = current\_pos + chunk\_size - 1;  
   
 // Add remainder elements to last thread  
 if (i == num\_threads - 1) {  
 thread\_args[i].end += remainder;  
 }  
   
 ret = pthread\_create(&thread\_args[i].tid, NULL, parallel\_mergesort, &thread\_args[i]);  
 if (ret != 0) {  
 fprintf(stderr, "Failed to create thread %zu: %s\n", i, strerror(ret));  
 // Clean up already created threads  
 for (size\_t j = 0; j < threads\_created; j++) {  
 pthread\_join(thread\_args[j].tid, NULL);  
 }  
 if (barrier\_initialized) {  
 pthread\_barrier\_destroy(&sort\_barrier);  
 }  
 free(thread\_args);  
 return;  
 }  
 threads\_created++;  
 current\_pos += chunk\_size;  
 }  
  
 for (unsigned int i = 0; i < num\_threads; i++) {  
 ret = pthread\_join(thread\_args[i].tid, NULL);  
 if (ret != 0) {  
 fprintf(stderr, "Failed to join thread %d: %s\n", i, strerror(ret));  
 }  
 }  
   
 // locking for main thread safety  
 pthread\_mutex\_lock(&merge\_mutex);  
  
 // Merge sorted chunks  
 // This is a very critical section!  
 for (size\_t size = chunk\_size; size < n; size = size \* 2) {  
 for (size\_t i = 0; i < n - size; i += size \* 2) {  
 int mid = (int)(i + size - 1);  
 int right\_end = (int)((i + size \* 2 - 1) < (n - 1) ? (i + size \* 2 - 1) : (n - 1));  
 merge\_s(A, (int)i, mid, right\_end);  
 }  
 }  
 pthread\_mutex\_unlock(&merge\_mutex);  
  
 // Check for barrier  
 if (barrier\_initialized) {  
 ret = pthread\_barrier\_destroy(&sort\_barrier);  
 if (ret != 0) {  
 fprintf(stderr, "Failed to destroy barrier: %s\n", strerror(ret));  
 }  
 }  
 free(thread\_args);  
}  
  
/\*\*  
 \* @brief retuns the current time as milliseconds  
 \* @return the number of milliseconds  
 \*/  
double getMilliSeconds() {  
 struct timeval now;  
 gettimeofday(&now, (struct timezone \*)0);  
 return (double)now.tv\_sec \* 1000.0 + (double)now.tv\_usec / 1000.0;  
}

### lab.h

#ifndef LAB\_H  
#define LAB\_H  
  
// Threading support  
#ifndef \_REENTRANT  
#define \_REENTRANT  
#endif  
#include <pthread.h>  
  
// Test mode specific configurations  
#ifdef TEST  
#define ENABLE\_VERIFICATION  
#endif  
  
  
 // The threshold that we will use to switch to insertion sort, make sure that  
 // you use test arrays bigger than 5 so you are testing the merge sort  
#define INSERTION\_SORT\_THRESHOLD 2  
#define MAX\_THREADS 32  
 /\*\*  
 \* @brief Sorts an array of ints into ascending order using the constant  
 \* INSERTION\_SORT\_THRESHOLD internally  
 \*  
 \* @param A A pointer to the start of the array  
 \* @param p The starting index  
 \* @param r The ending index  
 \*/  
 void mergesort\_s(int \*A, int p, int r);  
  
 /\*\*  
 \* @brief Merge two sorted sequences A[p..q] and A[q+1..r] and place merged  
 \* output back in array A. Uses extra space proportional to  
 \* A[p..r].  
 \*  
 \* @param A The array to merge into  
 \* @param p The starting index of the first half  
 \* @param q The middle  
 \* @param r The ending index of the second half  
 \*/  
 void merge\_s(int A[], int p, int q, int r);  
  
 /\*\*  
 \* @brief Sorts an array of ints into ascending order using multiple  
 \* threads  
 \*  
 \* @param A A pointer to the start of the array  
 \* @param n The size of the array  
 \* @param num\_threads The number of threads to use.  
 \*/  
 void mergesort\_mt(int \*A, size\_t n, unsigned int num\_threads);  
  
 /\*\*  
 \* @brief retuns the current time as milliseconds  
 \* @return the number of milliseconds  
 \*/  
 double getMilliSeconds();  
  
 /\*\*  
 \* @brief Represents a chunk of the array to be sorted by a thread  
 \*  
 \*/  
 struct parallel\_args  
 {  
 int \*A;  
 size\_t start;  
 size\_t end;  
 pthread\_t tid;  
 };  
  
 /\*\*  
 \* @brief The function that is called by each thread to sort their chunk  
 \*  
 \* @param args see struct parallel\_args  
 \* @return void\* always NULL  
 \*/  
 void \*parallel\_mergesort(void \*args);  
  
 /\*\*  
 \* @brief Entry point for the main function  
 \*  
 \* @param argc The argument count  
 \* @param argv The argument array  
 \* @return The exit code  
 \*/  
 int myMain(int argc, char \*\*argv);  
  
#ifdef \_\_cplusplus  
  
#endif  
  
#endif

### main.c

#include <stdio.h>  
#include <stdlib.h>  
#include "lab.h"  
  
#ifndef TEST  
/\*\*  
 \* @brief Main function to test the mergesort\_mt function with command line arguments  
 \* @param argc The number of command line arguments  
 \* @param argv The array of command line arguments  
 \* @return 0 on success, 1 on failure  
 \* AI Use: None  
 \*/  
int main(int argc, char \*\*argv)  
{  
  
 if (argc < 3)  
 {  
 printf("usage: %s <array\_size> <num\_threads>", argv[0]);  
 return 1;  
 }  
 // parsing command line args  
 int temp\_size = atoi(argv[1]);  
 if (temp\_size <= 0) {  
 fprintf(stderr, "Array size must be positive!\n");  
 return 1;  
 }  
 size\_t listSize = (size\_t)temp\_size;  
   
 int temp\_threads = atoi(argv[2]);  
 if (temp\_threads <= 0) {  
 fprintf(stderr, "Num threads must be positive!\n");  
 return 1;  
 }  
 size\_t num\_threads = (size\_t)temp\_threads;  
 int \*A = (int \*)malloc(sizeof(int) \* listSize);  
 if (A == NULL)  
 {  
 fprintf(stderr, "Memory allocation failed!\n");  
 return 1;  
 }  
 for (size\_t i = 0; i < listSize; i++)  
 {  
 A[i] = rand() % 1000; // rand nums up to 1000  
 }  
  
 double start = getMilliSeconds();  
 mergesort\_mt(A, listSize, num\_threads);  
 double end = getMilliSeconds();  
 // printing sorted array for verification  
 // for (size\_t i = 0; i < listSize; i++)  
 // {  
 // printf("%d ", A[i]);  
 // }  
 // printf("\n");  
 free(A);  
  
 printf("%zu %f\n", num\_threads, end - start);  
  
 return 0;  
}  
#endif

## Tests Files

### lab-test.c

#include <stdlib.h>  
#include <stdio.h>  
#include "harness/unity.h"  
#include "../src/lab.h"  
#include <string.h>  
  
// Helper function to verify array is sorted  
static int is\_sorted(int arr[], int size) {  
 for (int i = 1; i < size; i++) {  
 if (arr[i] < arr[i-1]) return 0;  
 }  
 return 1;  
}  
  
void setUp(void) {  
 printf("Setting up test...\n");  
}  
  
void tearDown(void) {  
 printf("Tearing down test...\n");  
}  
  
// Test basic array sorting  
void test\_basic\_sort(void) {  
 int arr[] = {64, 34, 25, 12, 22, 11, 90};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 mergesort\_mt(arr, size, 2);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr, size), "Array should be sorted in ascending order");  
 TEST\_ASSERT\_EQUAL\_INT(11, arr[0]);  
 TEST\_ASSERT\_EQUAL\_INT(90, arr[size-1]);  
}  
  
// Test already sorted array  
void test\_sorted\_array(void) {  
 int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 mergesort\_mt(arr, size, 2);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr, size), "Array should remain sorted");  
 TEST\_ASSERT\_EQUAL\_INT(1, arr[0]);  
 TEST\_ASSERT\_EQUAL\_INT(8, arr[size-1]);  
}  
  
// Test reverse sorted array  
void test\_reverse\_sorted\_array(void) {  
 int arr[] = {8, 7, 6, 5, 4, 3, 2, 1};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 mergesort\_mt(arr, size, 2);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr, size), "Reverse sorted array should be sorted ascending");  
 TEST\_ASSERT\_EQUAL\_INT(1, arr[0]);  
 TEST\_ASSERT\_EQUAL\_INT(8, arr[size-1]);  
}  
  
// Test array with duplicates  
void test\_array\_with\_duplicates(void) {  
 int arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 mergesort\_mt(arr, size, 3);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr, size), "Array with duplicates should be sorted");  
}  
  
// Test single element array  
void test\_single\_element(void) {  
 int arr[] = {42};  
 int size = 1;  
 mergesort\_mt(arr, size, 2);  
 TEST\_ASSERT\_EQUAL\_INT(42, arr[0]);  
}  
  
//different thread counts  
void test\_different\_thread\_counts(void) {  
 int arr[] = {9, 8, 7, 6, 5, 4, 3, 2, 1};  
 int size = sizeof(arr) / sizeof(arr[0]);  
   
 // 1 thread  
 int arr1[size];  
 memcpy(arr1, arr, sizeof(arr));  
 mergesort\_mt(arr1, size, 1);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr1, size), "Array should be sorted with 1 thread");  
   
 // 4 threads  
 int arr4[size];  
 memcpy(arr4, arr, sizeof(arr));  
 mergesort\_mt(arr4, size, 4);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arr4, size), "Array should be sorted with 4 threads");  
   
 // max threads  
 int arrMax[size];  
 memcpy(arrMax, arr, sizeof(arr));  
 mergesort\_mt(arrMax, size, MAX\_THREADS);  
 TEST\_ASSERT\_TRUE\_MESSAGE(is\_sorted(arrMax, size), "Array should be sorted with MAX\_THREADS");  
   
 for (int i = 0; i < size; i++) {  
 TEST\_ASSERT\_EQUAL\_INT(arr1[i], arr4[i]);  
 TEST\_ASSERT\_EQUAL\_INT(arr1[i], arrMax[i]);  
 }  
}  
  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_basic\_sort);  
 RUN\_TEST(test\_sorted\_array);  
 RUN\_TEST(test\_reverse\_sorted\_array);  
 RUN\_TEST(test\_array\_with\_duplicates);  
 RUN\_TEST(test\_single\_element);  
 RUN\_TEST(test\_different\_thread\_counts);  
 return UNITY\_END();  
}

## Scripts Files

### createplot.sh

#!/usr/bin/env bash  
#NOTE!!! THIS needs to be in the scripts directory!!!  
function usage() {  
 echo "$0 usage:" && grep " .)\ #" $0  
 exit 0  
}  
[ $# -eq 0 ] && usage  
while getopts "hs:f:" arg; do  
 case $arg in  
 s) # The size of the array to sort.  
 size=${OPTARG}  
 ;;  
 f) # The plot file name  
 name=${OPTARG}  
 ;;  
 h | \*) # Display help.  
 usage  
 exit 0  
 ;;  
 esac  
done  
MYAPP="../build/release/myapp"  
  
if [ "$name" == "" ] || [ "$size" == "" ]  
then  
 usage  
 exit 0  
fi  
if [ -e $MYAPP ]; then  
 if [ -e "data.dat" ]; then  
 rm -f data.dat  
 fi  
 echo "Running myprogram to generate data"  
 echo "#Time Threads" >> data.dat  
 for n in {1..32}; do  
 echo -ne "running $n thread \r"  
 $MYAPP "$size" "$n" >> data.dat  
 done  
 gnuplot -e "filename='$name.png'" graph.plt  
 echo "Created plot $name.png from data.dat file"  
else  
 echo "myapp is not present in the build directory. Did you compile your code?"  
fi

Report generated on 10/26/2025 at 22:33:26

## End of Report

SHA-256 Hash of the report: 9858c6d9e9b5c8d773ff76f83ac96ebf635aa9027c9042a760af056f76cb4dae

Do not edit the generated report. Any changes will be reported as academic dishonesty

## GitHub Info

* GitHub repo name: bmoserEDU/CS\_452\_P3\_Multithreaded\_Application
* The repository visibility is public.
* The workflow was triggered by bmoserEDU