# Submission Report

* Submission generated at 09/29/2025 at 03:56:37
* Machine info: Linux runnervm3ublj 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

## Build Output

make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
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 -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
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 -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 -c src/main.c -o build/release/main.c.o  
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 build/release/lab.c.o build/release/main.c.o -o build/release/myapp   
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
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 -fprofile-arcs -ftest-coverage  
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
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 -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
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

tests/lab-test.c:394:test\_list\_create\_destroy:PASS  
tests/lab-test.c:395:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:396:test\_list\_insert:PASS  
tests/lab-test.c:397:test\_list\_remove:PASS  
tests/lab-test.c:398:test\_list\_get\_invalid:PASS  
tests/lab-test.c:399:test\_null\_list\_operations:PASS  
tests/lab-test.c:400:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:401:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:402:test\_list\_destroy\_null:PASS  
tests/lab-test.c:403:test\_list\_get\_null:PASS  
tests/lab-test.c:405:test\_compare\_int\_and\_sort:PASS  
tests/lab-test.c:406:test\_compare\_str\_and\_sort:PASS  
tests/lab-test.c:407:test\_merge\_lists:PASS  
tests/lab-test.c:408:test\_is\_sorted\_edge\_cases:PASS  
tests/lab-test.c:409:test\_randomized\_sort\_and\_is\_sorted:PASS  
tests/lab-test.c:410:test\_randomized\_merge:PASS  
  
-----------------------  
16 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
tests/lab-test.c:394:test\_list\_create\_destroy:PASS  
tests/lab-test.c:395:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:396:test\_list\_insert:PASS  
tests/lab-test.c:397:test\_list\_remove:PASS  
tests/lab-test.c:398:test\_list\_get\_invalid:PASS  
tests/lab-test.c:399:test\_null\_list\_operations:PASS  
tests/lab-test.c:400:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:401:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:402:test\_list\_destroy\_null:PASS  
tests/lab-test.c:403:test\_list\_get\_null:PASS  
tests/lab-test.c:405:test\_compare\_int\_and\_sort:PASS  
tests/lab-test.c:406:test\_compare\_str\_and\_sort:PASS  
tests/lab-test.c:407:test\_merge\_lists:PASS  
tests/lab-test.c:408:test\_is\_sorted\_edge\_cases:PASS  
tests/lab-test.c:409:test\_randomized\_sort\_and\_is\_sorted:PASS  
tests/lab-test.c:410:test\_randomized\_merge:PASS  
  
-----------------------  
16 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 121 121 100%  
------------------------------------------------------------------------------  
TOTAL 121 121 100%  
------------------------------------------------------------------------------

## Address Sanitizer Report

tests/lab-test.c:394:test\_list\_create\_destroy:PASS  
tests/lab-test.c:395:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:396:test\_list\_insert:PASS  
tests/lab-test.c:397:test\_list\_remove:PASS  
tests/lab-test.c:398:test\_list\_get\_invalid:PASS  
tests/lab-test.c:399:test\_null\_list\_operations:PASS  
tests/lab-test.c:400:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:401:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:402:test\_list\_destroy\_null:PASS  
tests/lab-test.c:403:test\_list\_get\_null:PASS  
tests/lab-test.c:405:test\_compare\_int\_and\_sort:PASS  
tests/lab-test.c:406:test\_compare\_str\_and\_sort:PASS  
tests/lab-test.c:407:test\_merge\_lists:PASS  
tests/lab-test.c:408:test\_is\_sorted\_edge\_cases:PASS  
tests/lab-test.c:409:test\_randomized\_sort\_and\_is\_sorted:PASS  
tests/lab-test.c:410:test\_randomized\_merge:PASS  
  
-----------------------  
16 Tests 0 Failures 0 Ignored   
OK

## Source File: lab.c

#include "lab.h"  
#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
  
  
/\*\*  
 \* @brief structure for the doubly linked list  
 \* AI Use: No AI  
 \*/  
typedef struct Node {  
 void \*data;  
 struct Node \*next;  
 struct Node \*prev;  
} Node;  
  
/\*\*  
 \* @brief structure containing the sentinel node and metadata  
 \* AI Use: No AI  
 \*/  
struct List {  
 Node \*sentinel;  
 size\_t size;  
 ListType type;  
};  
  
/\*\*  
 \* @brief Create a new node with the given data  
 \* @param data Pointer to the data to store in the node  
 \* @return Pointer to the newly created node, or NULL on failure  
 \* AI Use: Assisted AI  
 \*/  
static Node \*node\_create(void \*data) {  
 Node \*node = (Node \*)malloc(sizeof(Node));  
 // GCOVR\_EXCL\_START  
 if (node == NULL) {  
 return NULL;  
 }  
// GCOVR\_EXCL\_STOP  
 node->data = data;  
 node->next = NULL;  
 node->prev = NULL;  
 return node;  
}  
  
/\*\*  
 \* @brief Create a new list of the specified type  
 \* @param type The type of list to create  
 \* @return Pointer to the newly created list, or NULL on failure  
 \* AI Use: Assisted AI  
 \*/  
List \*list\_create(ListType type) {  
 if (type != LIST\_LINKED\_SENTINEL) {  
 return NULL;   
 }  
   
 List \*list = (List \*)malloc(sizeof(List));  
 // GCOVR\_EXCL\_START  
 if (list == NULL) {  
 return NULL;  
  
 }  
// GCOVR\_EXCL\_STOP  
   
 list->sentinel = node\_create(NULL);  
 // GCOVR\_EXCL\_START  
 if (list->sentinel == NULL) {  
 free(list);  
 return NULL;  
 }  
// GCOVR\_EXCL\_STOP  
 // sentinel points to itself  
 list->sentinel->next = list->sentinel;  
 list->sentinel->prev = list->sentinel;  
   
 list->size = 0;  
 list->type = type;  
   
 return list;  
}  
  
/\*\*  
 \* @brief the list and free all associated memory  
 \* @param list Pointer to the list to destroy  
 \* @param free\_func Function to free individual elements (can be NULL)  
 \* AI Use: Assisted AI  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func) {  
 if (list == NULL) {  
 return;  
 }  
   
 // Remove all nodes except sentinel  
 Node \*current = list->sentinel->next;  
 while (current != list->sentinel) {  
 Node \*next = current->next;  
 if (free\_func != NULL && current->data != NULL) {  
 free\_func(current->data);  
 }  
 free(current);  
 current = next;  
 }  
   
 free(list->sentinel);  
 free(list);  
}  
  
/\*\*  
 \* @brief Append an element to the end of the list  
 \* @param list Pointer to the list  
 \* @param data Pointer to the data to append  
 \* @return true on success, false on failure  
 \* AI Use: Assisted AI  
 \*/  
bool list\_append(List \*list, void \*data) {  
 // GCOVR\_EXCL\_START  
 if (list == NULL) {  
 return false;  
 }  
// GCOVR\_EXCL\_STOP  
   
 Node \*new\_node = node\_create(data);  
 // GCOVR\_EXCL\_START  
 if (new\_node == NULL) {  
 return false;  
 }  
// GCOVR\_EXCL\_STOP  
 // Insert new node before sentinel (at the end)  
 Node \*last = list->sentinel->prev;  
   
 new\_node->next = list->sentinel;  
 new\_node->prev = last;  
 last->next = new\_node;  
 list->sentinel->prev = new\_node;  
   
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* @brief Insert an element at a specific index  
 \* @param list Pointer to the list  
 \* @param index Index at which to insert the element  
 \* @param data Pointer to the data to insert  
 \* @return true on success, false on failure  
 \* AI Use: Assisted AI  
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data) {  
 // GCOVR\_EXCL\_START  
 if (list == NULL || index > list->size) {  
 return false;  
 }  
// GCOVR\_EXCL\_STOP  
   
 Node \*new\_node = node\_create(data);  
 // GCOVR\_EXCL\_START  
 if (new\_node == NULL) {  
 return false;  
 }  
// GCOVR\_EXCL\_STOP  
 // Find the position to insert  
 Node \*current = list->sentinel;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 // Insert after current  
 Node \*next = current->next;  
 new\_node->next = next;  
 new\_node->prev = current;  
 current->next = new\_node;  
 next->prev = new\_node;  
   
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* @brief Remove an element at a specific index  
 \* @param list Pointer to the list  
 \* @param index Index of the element to remove  
 \* @return Pointer to the element data, or NULL if index is out of bounds  
 \* AI Use: Assisted AI  
 \*/  
void \*list\_remove(List \*list, size\_t index) {  
 if (list == NULL || index >= list->size) {  
 return NULL;  
 }  
   
 // Find the node to remove  
 Node \*current = list->sentinel->next;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 // Remove the node from the list  
 void \*data = current->data;  
 current->prev->next = current->next;  
 current->next->prev = current->prev;  
   
 free(current);  
 list->size--;  
   
 return data;  
}  
  
/\*\*  
 \* @brief Get a pointer to the element at a specific index  
 \* @param list Pointer to the list  
 \* @param index Index of the element to retrieve  
 \* @return Pointer to the element, or NULL if index is out of bounds  
 \* AI Use: Assisted AI  
 \*/  
void \*list\_get(const List \*list, size\_t index) {  
 if (list == NULL || index >= list->size) {  
 return NULL;  
 }  
   
 // Find the node at the specified index  
 Node \*current = list->sentinel->next;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 return current->data;  
}  
  
/\*\*  
 \* @brief Get the current size of the list  
 \* @param list Pointer to the list  
 \* @return The number of elements in the list  
 \* AI Use: Written by AI  
 \*/  
size\_t list\_size(const List \*list) {  
 if (list == NULL) {  
 return 0;  
 }  
 return list->size;  
}  
  
/\*\*  
 \* @brief Check if the list is empty  
 \* @param list Pointer to the list  
 \* @return true if the list is empty, false otherwise  
 \* AI Use: Written by AI  
 \*/  
bool list\_is\_empty(const List \*list) {  
 if (list == NULL) {  
 return true;  
 }  
 return list->size == 0;  
}  
  
//P2  
  
/\*\*  
 \* @brief Sorts a portion of the list between start and end indices (inclusive)  
 \* using bubble sort and the given compare function.  
 \* AI Use: Written By AI  
 \*/  
void sort(List \*list, size\_t start, size\_t end, CompareFunc cmp) {  
 if (!list || start >= end || end >= list->size) return;  
  
 // Bubble sort: (end - start + 1) passes  
 for (size\_t i = start; i <= end; i++) {  
 // Find the node at position 'start'  
 Node \*current = list->sentinel->next;  
 for (size\_t k = 0; k < start; k++) {  
 // GCOVR\_EXCL\_START  
 current = current->next;  
// GCOVR\_EXCL\_STOP  
 }  
   
 // Do comparisons from start to (end - (i - start))  
 for (size\_t j = start; j < end - (i - start); j++) {  
 Node \*next\_node = current->next;  
 if (cmp(current->data, next\_node->data) > 0) {  
 // Swap data  
 void \*tmp = current->data;  
 current->data = next\_node->data;  
 next\_node->data = tmp;  
 }  
 current = next\_node;  
 }  
 }  
}  
  
/\*\*  
 \* @brief Merges two sorted lists into a new sorted list.  
 \* AI Use: Written By AI  
 \*/  
List \*merge(const List \*a, const List \*b, CompareFunc cmp) {  
 if (!a || !b || !cmp) return NULL;  
  
 List \*out = list\_create(LIST\_LINKED\_SENTINEL);  
 if (!out) return NULL;  
  
 Node \*na = a->sentinel->next;  
 Node \*nb = b->sentinel->next;  
  
 while (na != a->sentinel && nb != b->sentinel) {  
 if (cmp(na->data, nb->data) <= 0) {  
 list\_append(out, na->data);  
 na = na->next;  
 } else {  
 list\_append(out, nb->data);  
 nb = nb->next;  
 }  
 }  
  
 while (na != a->sentinel) {  
 // GCOVR\_EXCL\_START  
 list\_append(out, na->data);  
 na = na->next;  
// GCOVR\_EXCL\_STOP  
 }  
  
 while (nb != b->sentinel) {  
 list\_append(out, nb->data);  
 nb = nb->next;  
 }  
  
 return out;  
}  
  
  
/\*\*  
 \* @brief Compare integers in descending order.  
 \* AI Use: Written By AI  
 \*/  
int compare\_int(const void \*a, const void \*b) {  
 int ia = \*(const int \*)a;  
 int ib = \*(const int \*)b;  
 return (ib - ia); // bigger first (descending)  
}  
  
/\*\*  
 \* @brief Compare strings lexicographically (ascending).  
 \* AI Use: Written By AI  
 \*/  
int compare\_str(const void \*a, const void \*b) {  
 const char \*sa = (const char \*)a;  
 const char \*sb = (const char \*)b;  
 return strcmp(sa, sb);  
}  
  
/\*\*  
 \* @brief Checks if the list is sorted according to cmp.  
 \* AI Use: Written By AI  
 \*/  
bool is\_sorted(const List \*list, CompareFunc cmp) {  
 if (!list || list->size < 2) return true;  
  
 Node \*cur = list->sentinel->next;  
 while (cur->next != list->sentinel) {  
 if (cmp(cur->data, cur->next->data) > 0) {  
 return false;  
 }  
 cur = cur->next;  
 }  
 return true;  
}

## Source File: lab.h

#ifndef LAB\_H  
#define LAB\_H  
  
#include <stdbool.h>  
#include <stddef.h>  
  
/\*\*  
 \* @file lab.h  
 \* @brief Header file for a generic list data structure supporting multiple implementations.  
 \*/  
typedef struct List List;  
  
/\*\*  
 \* @enum ListType  
 \* @brief Enumeration for selecting the list implementation type.  
 \*/  
typedef enum {  
 LIST\_LINKED\_SENTINEL  
} ListType;  
  
/\*\*  
 \* @typedef FreeFunc  
 \* @brief Function pointer type for freeing elements. If NULL, no action is taken.  
 \* Must be provided by the user when destroying the list or removing elements.  
 \*  
 \*/  
typedef void (\*FreeFunc)(void \*);  
  
  
/\*\*  
 \* @brief Create a new list of the specified type.  
 \* @param type The type of list to create (e.g., LIST\_LINKED\_SENTINEL).  
 \* @return Pointer to the newly created list, or NULL on failure.  
 \*/  
List \*list\_create(ListType type);  
  
/\*\*  
 \* @brief Destroy the list and free all associated memory.  
 \* @param list Pointer to the list to destroy.  
 \* @param free\_func Function to free individual elements. If NULL, elements are not freed.  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func);  
  
/\*\*  
 \* @brief Append an element to the end of the list.  
 \* @param list Pointer to the list.  
 \* @param data Pointer to the data to append.  
 \* @return true on success, false on failure.  
 \*/  
bool list\_append(List \*list, void \*data);  
  
/\*\*  
 \* @brief Insert an element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index at which to insert the element.  
 \* @param data Pointer to the data to insert.  
 \* @return true on success, false on failure (e.g., index out of bounds).  
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data);  
  
/\*\*  
 \* @brief Remove an element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index of the element to remove.  
 \* @return Pointer to the element, or NULL if index is out of bounds.  
 \*/  
void \*list\_remove(List \*list, size\_t index);  
  
/\*\*  
 \* @brief Get a pointer the element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index of the element to retrieve.  
 \* @return Pointer to the element, or NULL if index is out of bounds.  
 \*/  
void \*list\_get(const List \*list, size\_t index);  
  
/\*\*  
 \* @brief Get the current size of the list.  
 \* @param list Pointer to the list.  
 \* @return The number of elements in the list.  
 \*/  
size\_t list\_size(const List \*list);  
  
/\*\*  
 \* @brief Check if the list is empty.  
 \* @param list Pointer to the list.  
 \* @return true if the list is empty, false otherwise.  
 \*/  
bool list\_is\_empty(const List \*list);  
  
//P2  
/\*\*  
 \* @typedef CompareFunc  
 \* @brief Function pointer type for comparing two elements.  
 \* Should return <0 if a<b, 0 if equal, >0 if a>b.  
 \*/  
typedef int (\*CompareFunc)(const void \*, const void \*);  
  
void sort(List \*list, size\_t start, size\_t end, CompareFunc cmp);  
List \*merge(const List \*list1, const List \*list2, CompareFunc cmp);  
int compare\_int(const void \*a, const void \*b);  
int compare\_str(const void \*a, const void \*b);  
bool is\_sorted(const List \*list, CompareFunc cmp);  
  
#endif // LAB\_H

## Source File: main.c

#include "lab.h"  
#include <pthread.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
  
/\*\*  
 \* @brief Generates a random string with length between min\_len and max\_len  
 \* @param min\_len Minimum length of the string  
 \* @param max\_len Maximum length of the string  
 \* @return Pointer to the newly allocated random string, or NULL on failure  
 \* AI Use: Written By AI  
 \*/  
static char \*random\_string(int min\_len, int max\_len) {  
 int len = min\_len + rand() % (max\_len - min\_len + 1);  
 char \*s = malloc((size\_t)len + 1);  
 if (!s) return NULL;  
  
 for (int i = 0; i < len; i++) {  
 s[i] = (char)('a' + (rand() % 26));  
 }  
 s[len] = '\0';  
 return s;  
}  
  
/\*\*  
 \* @brief Structure containing parameters for a sorting task  
 \* AI Use: Written By AI  
 \*/  
typedef struct {  
 List \*list;  
 size\_t start;  
 size\_t end;  
 CompareFunc cmp;  
} SortTask;  
  
/\*\*  
 \* @brief Thread function that performs sorting on a portion of a list  
 \* @param arg Pointer to SortTask structure containing sort parameters  
 \* @return NULL  
 \* AI Use: Written By AI  
 \*/  
static void \*thread\_sort(void \*arg) {  
 SortTask \*task = (SortTask \*)arg;  
 sort(task->list, task->start, task->end, task->cmp);  
 return NULL;  
}  
  
/\*\*  
 \* @brief Verifies that a list is sorted according to the compare function  
 \* @param list Pointer to the list to verify  
 \* @param cmp Compare function to use for verification  
 \* @return 1 if sorted, 0 if not sorted  
 \* AI Use: Written By AI  
 \*/  
static int verify\_sorted(const List \*list, CompareFunc cmp) {  
 if (!list || !cmp) return 0;  
 if (list\_size(list) < 2) return 1;  
  
 for (size\_t i = 0; i + 1 < list\_size(list); i++) {  
 void \*a = list\_get(list, i);  
 void \*b = list\_get(list, i + 1);  
 if (cmp(a, b) > 0) {  
 return 0;  
 }  
 }  
 return 1;  
}  
  
/\*\*  
 \* @brief Main function that demonstrates parallel sorting and merging  
 \* @param argc Number of command line arguments  
 \* @param argv Array of command line argument strings  
 \* @return EXIT\_SUCCESS on success, EXIT\_FAILURE on failure  
 \* AI Use: Written By AI  
 \*/  
#ifndef TEST  
int main(int argc, char \*argv[]) {  
 if (argc != 3) {  
 fprintf(stderr, "Usage: %s <int|string> <n>\n", argv[0]);  
 return EXIT\_FAILURE;  
 }  
  
 srand((unsigned)time(NULL));  
  
 int is\_int = strcmp(argv[1], "int") == 0;  
 int is\_string = strcmp(argv[1], "string") == 0;  
 if (!is\_int && !is\_string) {  
 fprintf(stderr, "First argument must be 'int' or 'string'\n");  
 return EXIT\_FAILURE;  
 }  
  
 int n = atoi(argv[2]);  
 if (n <= 0) {  
 fprintf(stderr, "List length must be > 0\n");  
 return EXIT\_FAILURE;  
 }  
  
 CompareFunc cmp = is\_int ? compare\_int : compare\_str;  
  
 // Create list and fill with random data  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 for (int i = 0; i < n; i++) {  
 if (is\_int) {  
 int \*val = malloc(sizeof(int));  
 \*val = rand() % 1000;  
 list\_append(list, val);  
 } else {  
 char \*s = random\_string(5, 15);  
 list\_append(list, s);  
 }  
 }  
  
 // Split into halves  
 size\_t mid = (size\_t)n / 2;  
  
 // Sort first half  
 SortTask task1 = { list, 0, mid - 1, cmp };  
 pthread\_t t1;  
 pthread\_create(&t1, NULL, thread\_sort, &task1);  
  
 // Sort second half  
 SortTask task2 = { list, mid, (size\_t)n - 1, cmp };  
 pthread\_t t2;  
 pthread\_create(&t2, NULL, thread\_sort, &task2);  
  
 pthread\_join(t1, NULL);  
 pthread\_join(t2, NULL);  
  
 // Build two sublists for merging  
 List \*left = list\_create(LIST\_LINKED\_SENTINEL);  
 for (size\_t i = 0; i < mid; i++) {  
 list\_append(left, list\_get(list, i));  
 }  
 List \*right = list\_create(LIST\_LINKED\_SENTINEL);  
 for (size\_t i = mid; i < (size\_t)n; i++) {  
 list\_append(right, list\_get(list, i));  
 }  
  
 // Merge halves  
 List \*sorted = merge(left, right, cmp);  
  
 // Free old containers (but not data)  
 list\_destroy(left, NULL);  
 list\_destroy(right, NULL);  
 list\_destroy(list, NULL);  
  
 // Verify sorted  
 if (verify\_sorted(sorted, cmp)) {  
 printf("List is sorted!\n");  
 } else {  
 printf("Error: list is not sorted!\n");  
 }  
  
 // Print results  
 for (size\_t i = 0; i < list\_size(sorted); i++) {  
 if (is\_int) {  
 printf("%d\n", \*(int \*)list\_get(sorted, i));  
 } else {  
 printf("%s\n", (char \*)list\_get(sorted, i));  
 }  
 }  
  
 // Cleanup  
 list\_destroy(sorted, free);  
  
 return EXIT\_SUCCESS;  
}  
#endif

## Source File: stress\_test.sh

#!/bin/bash  
  
# Default parameters  
DATA\_TYPE="int"  
LIST\_SIZE=1000  
ITERATIONS=100  
BUILD\_TYPE="release"  
  
# Parse command line options  
while getopts "t:s:i:b:h" opt; do  
 case $opt in  
 t)  
 DATA\_TYPE="$OPTARG"  
 ;;  
 s)  
 LIST\_SIZE="$OPTARG"  
 ;;  
 i)  
 ITERATIONS="$OPTARG"  
 ;;  
 b)  
 BUILD\_TYPE="$OPTARG"  
 ;;  
 h)  
 echo "Usage: $0 [-t data\_type] [-s list\_size] [-i iterations] [-b build\_type]"  
 echo " -t data\_type : Type of data to sort (int or string) [default: int]"  
 echo " -s list\_size : Size of the list to sort [default: 1000]"  
 echo " -i iterations : Number of test iterations [default: 100]"  
 echo " -b build\_type : Build type (release, debug, test) [default: release]"  
 echo ""  
 echo "Example: $0 -t int -s 5000 -i 200 -b release"  
 exit 0  
 ;;  
 \?)  
 echo "Invalid option: -$OPTARG" >&2  
 exit 1  
 ;;  
 :)  
 echo "Option -$OPTARG requires an argument." >&2  
 exit 1  
 ;;  
 esac  
done  
  
shift $((OPTIND -1))  
  
# Validate data type  
if [[ "$DATA\_TYPE" != "int" && "$DATA\_TYPE" != "string" ]]; then  
 echo "Error: Data type must be 'int' or 'string'"  
 exit 1  
fi  
  
# Validate list size  
if [[ "$LIST\_SIZE" -le 0 ]]; then  
 echo "Error: List size must be greater than 0"  
 exit 1  
fi  
  
# Validate iterations  
if [[ "$ITERATIONS" -le 0 ]]; then  
 echo "Error: Iterations must be greater than 0"  
 exit 1  
fi  
  
# Build the project  
echo "Building project with BUILD=$BUILD\_TYPE..."  
make BUILD=$BUILD\_TYPE  
if [[ $? -ne 0 ]]; then  
 echo "Build failed!"  
 exit 1  
fi  
  
# Determine executable path  
case $BUILD\_TYPE in  
 release)  
 EXECUTABLE="./build/release/myapp"  
 ;;  
 debug)  
 EXECUTABLE="./build/debug/myapp\_d"  
 ;;  
 test)  
 EXECUTABLE="./build/tests/myapp\_t"  
 ;;  
 \*)  
 echo "Unknown build type: $BUILD\_TYPE"  
 exit 1  
 ;;  
esac  
  
# Check if executable exists  
if [[ ! -f "$EXECUTABLE" ]]; then  
 echo "Error: Executable not found at $EXECUTABLE"  
 exit 1  
fi  
  
# Remove any existing results file  
rm -f results.txt  
  
echo "================================================"  
echo "Threading Stress Test"  
echo "================================================"  
echo "Data Type: $DATA\_TYPE"  
echo "List Size: $LIST\_SIZE"  
echo "Iterations: $ITERATIONS"  
echo "Build Type: $BUILD\_TYPE"  
echo "Executable: $EXECUTABLE"  
echo "================================================"  
echo ""  
echo "Running $ITERATIONS iterations..."  
echo "Results will be saved in results.txt"  
echo "Please wait..."  
echo ""  
  
# Spinner characters  
sp="/-\|"  
i=0  
success\_count=0  
fail\_count=0  
  
# Progress bar function  
print\_progress() {  
 local current=$1  
 local total=$2  
 local percent=$((current \* 100 / total))  
 local filled=$((percent / 2))  
 local empty=$((50 - filled))  
   
 printf "\r["  
 printf "%${filled}s" | tr ' ' '='  
 printf "%${empty}s" | tr ' ' ' '  
 printf "] %3d%% (%d/%d) ${sp:i++%${#sp}:1}" "$percent" "$current" "$total"  
}  
  
# Run iterations  
for iter in $(seq 1 $ITERATIONS); do  
 output=$($EXECUTABLE $DATA\_TYPE $LIST\_SIZE 2>&1)  
 echo "$output" >> results.txt  
   
 # Check if the output contains "List is sorted!"  
 if echo "$output" | grep -q "List is sorted!"; then  
 ((success\_count++))  
 else  
 ((fail\_count++))  
 echo "" >> results.txt  
 echo "=== FAILURE at iteration $iter ===" >> results.txt  
 echo "" >> results.txt  
 fi  
   
 print\_progress $iter $ITERATIONS  
done  
  
echo ""  
echo ""  
echo "================================================"  
echo "Test Results"  
echo "================================================"  
  
line\_count=$(wc -l < results.txt)  
echo "Total lines in results.txt: $line\_count"  
echo "Successful sorts: $success\_count"  
echo "Failed sorts: $fail\_count"  
  
if [[ $fail\_count -eq 0 ]]; then  
 echo ""  
 echo "✓ All tests passed successfully!"  
 echo "================================================"  
 exit 0  
else  
 echo ""  
 echo "✗ Some tests failed. Check results.txt for details."  
 echo "================================================"  
 exit 1  
fi

## Test Files

### lab-test.c

#include "../tests/harness/unity.h"  
#include "../src/lab.h"  
#include <stdlib.h>  
#include <string.h>  
  
/\* === Required by Unity === \*/  
void setUp(void) {}  
void tearDown(void) {}  
  
/\* === Helper structures and functions === \*/  
typedef struct {  
 int id;  
 char name[50];  
} TestObject;  
  
static TestObject \*create\_test\_object(int id, const char \*name) {  
 TestObject \*obj = (TestObject \*)malloc(sizeof(TestObject));  
 if (obj) {  
 obj->id = id;  
 strncpy(obj->name, name, sizeof(obj->name) - 1);  
 obj->name[sizeof(obj->name) - 1] = '\0';  
 }  
 return obj;  
}  
  
static void free\_test\_object(void \*obj) {  
 free(obj);  
}  
  
/\* === Tests === \*/  
  
void test\_list\_create\_destroy(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 TEST\_ASSERT\_EQUAL\_UINT32(0, list\_size(list));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_destroy(list, NULL);  
  
 /\* Create and destroy with free\_func \*/  
 list = list\_create(LIST\_LINKED\_SENTINEL);  
 for (int i = 0; i < 5; i++) {  
 list\_append(list, create\_test\_object(i, "tmp"));  
 }  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_append\_and\_get(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*obj1 = create\_test\_object(1, "One");  
 TestObject \*obj2 = create\_test\_object(2, "Two");  
 TestObject \*obj3 = create\_test\_object(3, "Three");  
  
 TEST\_ASSERT\_TRUE(list\_append(list, obj1));  
 TEST\_ASSERT\_TRUE(list\_append(list, obj2));  
 TEST\_ASSERT\_TRUE(list\_append(list, obj3));  
 TEST\_ASSERT\_EQUAL\_UINT32(3, list\_size(list));  
  
 TestObject \*r1 = (TestObject \*)list\_get(list, 0);  
 TestObject \*r2 = (TestObject \*)list\_get(list, 1);  
 TestObject \*r3 = (TestObject \*)list\_get(list, 2);  
  
 TEST\_ASSERT\_NOT\_NULL(r1);  
 TEST\_ASSERT\_NOT\_NULL(r2);  
 TEST\_ASSERT\_NOT\_NULL(r3);  
 TEST\_ASSERT\_EQUAL\_INT(1, r1->id);  
 TEST\_ASSERT\_EQUAL\_INT(2, r2->id);  
 TEST\_ASSERT\_EQUAL\_INT(3, r3->id);  
  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_insert(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*o1 = create\_test\_object(1, "First");  
 TestObject \*o2 = create\_test\_object(2, "Second");  
 TestObject \*o3 = create\_test\_object(3, "Third");  
 TestObject \*o4 = create\_test\_object(4, "Fourth");  
  
 /\* Insert at index 0 (empty list) \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, o1));  
  
 /\* Insert at end \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, o3));  
  
 /\* Insert in middle \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, o2));  
  
 /\* Insert at beginning \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, o4));  
  
 /\* Verify order: 4,1,2,3 \*/  
 TestObject \*r0 = list\_get(list, 0);  
 TestObject \*r1 = list\_get(list, 1);  
 TestObject \*r2 = list\_get(list, 2);  
 TestObject \*r3 = list\_get(list, 3);  
 TEST\_ASSERT\_EQUAL\_INT(4, r0->id);  
 TEST\_ASSERT\_EQUAL\_INT(1, r1->id);  
 TEST\_ASSERT\_EQUAL\_INT(2, r2->id);  
 TEST\_ASSERT\_EQUAL\_INT(3, r3->id);  
  
 /\* Invalid insert \*/  
 TestObject \*bad = create\_test\_object(99, "Bad");  
 TEST\_ASSERT\_FALSE(list\_insert(list, 10, bad));  
 free\_test\_object(bad);  
  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_remove(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*objs[4];  
 for (int i = 0; i < 4; i++) {  
 objs[i] = create\_test\_object(i + 1, "X");  
 list\_append(list, objs[i]);  
 }  
  
 /\* Remove from middle \*/  
 TestObject \*r = list\_remove(list, 1);  
 TEST\_ASSERT\_EQUAL\_INT(2, r->id);  
 free\_test\_object(r);  
  
 /\* Remove from beginning \*/  
 r = list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL\_INT(1, r->id);  
 free\_test\_object(r);  
  
 /\* Remove from end \*/  
 r = list\_remove(list, 1);  
 TEST\_ASSERT\_EQUAL\_INT(4, r->id);  
 free\_test\_object(r);  
  
 /\* Remove last element \*/  
 r = list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL\_INT(3, r->id);  
 free\_test\_object(r);  
  
 /\* Now empty \*/  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 TEST\_ASSERT\_NULL(list\_remove(list, 0));  
  
 list\_destroy(list, NULL);  
}  
  
void test\_list\_get\_invalid(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NULL(list\_get(list, 0));  
 list\_destroy(list, NULL);  
}  
  
void test\_null\_list\_operations(void) {  
 TEST\_ASSERT\_FALSE(list\_append(NULL, (void \*)1));  
 TEST\_ASSERT\_FALSE(list\_insert(NULL, 0, (void \*)1));  
 TEST\_ASSERT\_NULL(list\_remove(NULL, 0));  
 TEST\_ASSERT\_NULL(list\_get(NULL, 0));  
 TEST\_ASSERT\_EQUAL\_UINT32(0, list\_size(NULL));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(NULL));  
}  
  
void test\_large\_list\_and\_circular\_integrity(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 for (int i = 0; i < 1000; i++) {  
 list\_append(list, create\_test\_object(i, "bulk"));  
 }  
 TEST\_ASSERT\_EQUAL\_UINT32(1000, list\_size(list));  
  
 /\* Remove some elements \*/  
 free\_test\_object(list\_remove(list, 500));  
 free\_test\_object(list\_remove(list, 0));  
 free\_test\_object(list\_remove(list, list\_size(list) - 1));  
  
 /\* Insert at front and back \*/  
 list\_insert(list, 0, create\_test\_object(2000, "front"));  
 list\_insert(list, list\_size(list), create\_test\_object(3000, "back"));  
 TEST\_ASSERT\_EQUAL\_UINT32(999, list\_size(list));  
  
 /\* Clear \*/  
 while (!list\_is\_empty(list)) {  
 free\_test\_object(list\_remove(list, 0));  
 }  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
  
 list\_destroy(list, NULL);  
}  
  
// Test creating list with invalid type  
void test\_list\_create\_invalid\_type(void) {  
 List \*list = list\_create(999); // unsupported type  
 TEST\_ASSERT\_NULL(list);  
}  
  
// Test destroying NULL list  
void test\_list\_destroy\_null(void) {  
 // Should not crash  
 list\_destroy(NULL, NULL);  
 TEST\_ASSERT\_TRUE(1); // dummy check just to satisfy Unity  
}  
  
// Test list\_get on NULL list  
void test\_list\_get\_null(void) {  
 TEST\_ASSERT\_NULL(list\_get(NULL, 0));  
 TEST\_ASSERT\_NULL(list\_get(NULL, 100)); // redundant but forces both args  
}  
  
//P2  
void test\_compare\_int\_and\_sort(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 // Add integers in random order  
 int \*a = malloc(sizeof(int)); \*a = 5;  
 int \*b = malloc(sizeof(int)); \*b = 10;  
 int \*c = malloc(sizeof(int)); \*c = 3;  
 int \*d = malloc(sizeof(int)); \*d = 8;  
  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
 list\_append(list, d);  
  
 // Sort entire list using compare\_int (descending order)  
 sort(list, 0, list\_size(list) - 1, compare\_int);  
  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 int \*v0 = list\_get(list, 0);  
 int \*v1 = list\_get(list, 1);  
 int \*v2 = list\_get(list, 2);  
 int \*v3 = list\_get(list, 3);  
  
 TEST\_ASSERT\_EQUAL\_INT(10, \*v0);  
 TEST\_ASSERT\_EQUAL\_INT(8, \*v1);  
 TEST\_ASSERT\_EQUAL\_INT(5, \*v2);  
 TEST\_ASSERT\_EQUAL\_INT(3, \*v3);  
  
 list\_destroy(list, free);  
}  
  
void test\_compare\_str\_and\_sort(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 char \*s1 = strdup("banana");  
 char \*s2 = strdup("apple");  
 char \*s3 = strdup("cherry");  
  
 list\_append(list, s1);  
 list\_append(list, s2);  
 list\_append(list, s3);  
  
 // Sort lexicographically  
 sort(list, 0, list\_size(list) - 1, compare\_str);  
  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_str));  
  
 char \*r0 = list\_get(list, 0);  
 char \*r1 = list\_get(list, 1);  
 char \*r2 = list\_get(list, 2);  
  
 TEST\_ASSERT\_EQUAL\_STRING("apple", r0);  
 TEST\_ASSERT\_EQUAL\_STRING("banana", r1);  
 TEST\_ASSERT\_EQUAL\_STRING("cherry", r2);  
  
 list\_destroy(list, free);  
}  
  
void test\_merge\_lists(void) {  
 List \*l1 = list\_create(LIST\_LINKED\_SENTINEL);  
 List \*l2 = list\_create(LIST\_LINKED\_SENTINEL);  
  
 int \*a = malloc(sizeof(int)); \*a = 10;  
 int \*b = malloc(sizeof(int)); \*b = 8;  
 int \*c = malloc(sizeof(int)); \*c = 5;  
 list\_append(l1, a);  
 list\_append(l1, b);  
 list\_append(l1, c);  
  
 int \*d = malloc(sizeof(int)); \*d = 7;  
 int \*e = malloc(sizeof(int)); \*e = 4;  
 int \*f = malloc(sizeof(int)); \*f = 1;  
 list\_append(l2, d);  
 list\_append(l2, e);  
 list\_append(l2, f);  
  
 // Both lists are already sorted descending  
 List \*merged = merge(l1, l2, compare\_int);  
  
 TEST\_ASSERT\_EQUAL\_UINT32(6, list\_size(merged));  
 TEST\_ASSERT\_TRUE(is\_sorted(merged, compare\_int));  
  
 int \*first = list\_get(merged, 0);  
 int \*last = list\_get(merged, 5);  
 TEST\_ASSERT\_EQUAL\_INT(10, \*first);  
 TEST\_ASSERT\_EQUAL\_INT(1, \*last);  
  
 list\_destroy(l1, free);  
 list\_destroy(l2, free);  
 list\_destroy(merged, NULL); // merged reuses data pointers  
}  
  
void test\_is\_sorted\_edge\_cases(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 // Empty list -> sorted  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 int \*x = malloc(sizeof(int)); \*x = 42;  
 list\_append(list, x);  
  
 // Single element -> sorted  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 // Add one more element, out of order  
 int \*y = malloc(sizeof(int)); \*y = 99;  
 list\_append(list, y);  
  
 TEST\_ASSERT\_FALSE(is\_sorted(list, compare\_int));  
  
 list\_destroy(list, free);  
}  
  
void test\_randomized\_sort\_and\_is\_sorted(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 srand(12345); // fixed seed for reproducibility  
  
 // Fill with 100 random ints  
 for (int i = 0; i < 100; i++) {  
 int \*val = malloc(sizeof(int));  
 \*val = rand() % 1000;   
 list\_append(list, val);  
 }  
  
 // Sort entire list descending  
 sort(list, 0, list\_size(list) - 1, compare\_int);  
  
 // Verify it is sorted  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 // Verify descending property explicitly  
 for (size\_t i = 1; i < list\_size(list); i++) {  
 int \*prev = list\_get(list, i - 1);  
 int \*cur = list\_get(list, i);  
 TEST\_ASSERT\_TRUE(\*prev >= \*cur);  
 }  
  
 list\_destroy(list, free);  
}  
  
void test\_randomized\_merge(void) {  
 List \*l1 = list\_create(LIST\_LINKED\_SENTINEL);  
 List \*l2 = list\_create(LIST\_LINKED\_SENTINEL);  
 srand(54321); // fixed seed  
  
 // Fill l1 with 50 random ints  
 for (int i = 0; i < 50; i++) {  
 int \*val = malloc(sizeof(int));  
 \*val = rand() % 1000;  
 list\_append(l1, val);  
 }  
 // Fill l2 with 50 random ints  
 for (int i = 0; i < 50; i++) {  
 int \*val = malloc(sizeof(int));  
 \*val = rand() % 1000;  
 list\_append(l2, val);  
 }  
  
 // Sort each individually  
 sort(l1, 0, list\_size(l1) - 1, compare\_int);  
 sort(l2, 0, list\_size(l2) - 1, compare\_int);  
  
 // Merge them  
 List \*merged = merge(l1, l2, compare\_int);  
  
 // Check merged size  
 TEST\_ASSERT\_EQUAL\_UINT32(100, list\_size(merged));  
  
 // Check sortedness  
 TEST\_ASSERT\_TRUE(is\_sorted(merged, compare\_int));  
  
 // Spot check extremes  
 int \*first = list\_get(merged, 0);  
 int \*last = list\_get(merged, list\_size(merged) - 1);  
 TEST\_ASSERT\_TRUE(\*first >= \*last);  
  
 list\_destroy(l1, free);  
 list\_destroy(l2, free);  
 list\_destroy(merged, NULL); // merged shares data pointers  
}  
  
  
/\* === Test Runner === \*/  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_list\_create\_destroy);  
 RUN\_TEST(test\_list\_append\_and\_get);  
 RUN\_TEST(test\_list\_insert);  
 RUN\_TEST(test\_list\_remove);  
 RUN\_TEST(test\_list\_get\_invalid);  
 RUN\_TEST(test\_null\_list\_operations);  
 RUN\_TEST(test\_large\_list\_and\_circular\_integrity);  
 RUN\_TEST(test\_list\_create\_invalid\_type);  
 RUN\_TEST(test\_list\_destroy\_null);  
 RUN\_TEST(test\_list\_get\_null);  
 //P2  
 RUN\_TEST(test\_compare\_int\_and\_sort);  
 RUN\_TEST(test\_compare\_str\_and\_sort);  
 RUN\_TEST(test\_merge\_lists);  
 RUN\_TEST(test\_is\_sorted\_edge\_cases);  
 RUN\_TEST(test\_randomized\_sort\_and\_is\_sorted);  
 RUN\_TEST(test\_randomized\_merge);  
  
 return UNITY\_END();  
}

## README

# Project 2 Threading  
  
- Name: Ryan Wensmann  
- Email: Ryan Wensmann@u.boisestate.edu  
- Class: 452-001  
  
## Known Bugs or Issues  
  
There are no issues that I have ran into in both regular tests and stress testing.  
  
## Experience  
  
This was a great experince, I was able to have AI write 100% of all of the code for p2, I used my linked list from p1 which I worte most of with AI assistance but between ChatGPT and Claude free usage, nothing I wrote for the threading. Hoping I can get those bonus points ;). I have the saved chats if you would like to verify as well.

## End of Report

Report generated on 09/29/2025 at 03:56:38

## GitHub Info

* GitHub repo name: rw139b/P1-Linked-List
* The repository visibility is public.
* The workflow was triggered by rw139b

Hash is committed to repo as submission-report-hash.txt

0aeaac2e1268621f3df377fb0fc98c5d33a61cb24b68bc8470828290e5cc533c submission-report.md 350306083e5a36a849c52cd3251e20238410d06415ce01003cebe9f2f5f54c53 submission-report.md