Week 6 Lab: Fast Sorting

This Week

This week, we will be starting a new backlog: Linked Lists. This backlog will have you implement the algorithms that were covered in weeks 5 and 6. To get the files you need for the next two demos, download `LinkedList.zip` from `ds.nathanielgmartin.com`. That is, use the command `wget ds.nathanielgmartin.com/LinkedList.zip` and unzip it. In the directory produced you will find functions to complete for the next two weeks. Next week you will complete stories 1-6; the following week you will complete stories 7-12. You should be able to leave the stories for next week blank and still have the file compile and run without errors. You may, or course, start working on the stories for the following week since the material will be covered this week.

Tip

These functions involve pointers, which makes them difficult. Test driven development is even more important with pointers than it is in most programs because you cannot see what they are doing and they fail without given you any information.

Use `printf` to track what is going on at each step. Unfortunately `printf` sometimes fails when pointers fail to making debugging even harder. What happens is that pointer errors cause the operating system to stop your program immediately. The `printf` function buffers information before it prints it out because input and output take a long longer than computation. However, this means that all of the buffered information is lost when the operating system shuts down your program.

You can solve this problem, and we recommend that you do, by using the `fflush` function. This function causes the operating system to put whatever printf has buffered on the screen immediately. Here is an example:

```
printf("Testing story 2: add_sll\n");
print_list("Printing NULL", test_list);
fflush(stdout);
test_list = add_sll(test_list, 3);
```

Here we are testing `add_sll`. Before we call the function we call `fflush(stdout)` to make sure that we will see what we printed immediately before. If the program crashes we know that the add_ssl function failed.

The `fflush` function takes a `FILE *` parameter. Here we want to print out the standard output, so we pass in `stdout`.

Tests Provided

In this assignment, we provide you with a set of tests to run against the program. The tests are found in

test.c. You might find it easier to develop the program using by these tests. For example, the first test tests printing by running two tests. First, it prints the null list, then it prints a list with three elements in it. Start by commenting out the second test, and get the first test to run. Then you can uncomment the second test and get it to run. Once you have gotten the tests to run, try the program from the user interface to make sure it runs on other lists.

To compile the tests, use the command `make test`. To run them use `./tst`.

Demo

You will be demoing the merge sort and quick sort functions you did last week.

Story 14 (15 Marks)

If you enter an 'S' command you are prompted to enter the type of sort. When you enter 'm' the array entered earlier is then sorted using an merge sort.

1. Basics
    1. Formatting is correct (2 Marks)
    2. Compiles without errors or warnings (3 Marks)
2. Correctness (10 Marks)
    1. Sort {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (3 Marks)
    2. Sort {1, 4, 2, 3, 5, 8, 6, 7, 9, 0} (3 Marks)
    3. No visible errors (4 Marks)

Story 15 (15 Marks)

If you enter an 'S' command you are prompted to enter the type of search. When you enter 'q' the array entered earlier is then sorted using a quick sort.

1. Basics (5 Marks)
    1. Formatting is correct (3 Marks)
    2. Compiles without errors or warnings (2 Marks)
2. Correctness (10 Marks)
    1. Search for 0 in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (2 Marks)
    2. Search for 0 in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (2 Marks)
    3. Search for 0 in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (2 Marks)
    4. No visible errors (4 Marks)