CS 305 Lab 9: Sorting
Fall 2019

The purpose of this lab is to give you experience with sorting routines.

This lab has a total of 100 possible points (30 points from pre-lab and 70 points from lab).

Objectives

Upon completion of the laboratory exercise, you will be able to do the following:

- explain how mergesort, quicksort, selection sort, and insertion sort work
- time various sorting algorithms

Part 1: Logging in (both people)

1. **BE SURE TO USE THE ENGINEERING CLASSROOM machine for this lab, so all the timings are consistent.** Follow the procedure from prior labs to log in and open Mobaxterm. Go into your cs305 directory. Make a new `lab9` directory:

```
cd cs305
mkdir lab9
```

2. Get the lab 9 files. Download `lab9.zip` from moodle to your lab9 folder. Unzip it:

```
unzip lab9.zip
cd starter
```

Part 2: Understand the existing code

In all the sorting implementations, there are two `#defines` that are important for you to know about. `DO_INT` is set to 1. When it is set to 1, the sorting routine operates on ints, using < as the comparator. `LARGE` is set to 1. When it is set to 1, the sorting routine gets the number of ints to generate and sort from the command line.

All the sorting routines can operate on ints or strings, depending if `DO_INT` is set or not. You will see a `#define` for `less` that is defined as ((a) < (b)) for ints and (strcmp((a), (b))<0) for strings.

1. Open `arraySelectionSort.c` to see how selection sort is implemented when the list of data is stored in an array. Look at the definition for `selectionSort`.

Suppose `l` is set to 0 and `r` is set to 99. How many times does the outer `for` loop execute? __________

2. Open `listSelectionSort.c` to see how selection sort is implemented when the list of data is stored in a linked list. This implementation uses a second linked list,

where instead of swapping elements within a single linked list, the chosen value from the original list is inserted at the front on the second list. Which value is found each time (circle answer)?

MAX IN LIST H   MIN IN LIST H

3. Ok, now on to insertion sort. Open `arrayInsertionSort.c`. Look at the `insertionSort` function definition. Which of the loops does the shifting of elements in the array `a` (circle answer)?

OUTER LOOP   INNER LOOP

4. Open `listInsertionSort.c`. To which list are the sorted items inserted (circle answer)?

LIST a   LIST b

5. Open `quicksort.c`. Look at the `partition` function. This implementation of partition is slightly different than the book's and one shown in lecture. Remember, at the end of partition, all values to the right of the pivot are greater than the pivot and all values to the left of the pivot are less than the pivot. How the values get moved around is specific to each implementation of partition.

6. Which value is used as the pivot (circle answer)?

Left-most value in a   Right-most value in a

7. Suppose the array `a` has the following items:

24   50   33   20   10   26

What does the array `a` contain after `partition` is called?

8. Open `mergesort.c`. Does this implementation use arrays or linked lists to store the data for merge sort (circle answer)?

Linked list   Array

9. Is the `merge` function written recursively or iteratively (circle answer)?

Recursively   Iteratively

Part 3: Run the sorting code

1. Create executables for all the sorting routines.

```
make clean
make all
```

You should now have the following executables in your directory: `isa`, `isl`, `ssa`, `ssl`, `qs`, `ms`

isa = insertion sort, array
isl = insertion sort, list
ssa = selection sort, array
ssl = selection sort, list
qs = quicksort
ms = mergesort

Try running isa:

```
./isa
```

What message do you get? ____

Try running:

```
./isa 1000
```

You should see a bunch of numbers printed to the screen, showing the list before and after sorting. Note that the middle set of numbers is not printed (…). The code is written so that it produces 1000 (or whatever the user types) random numbers.

Try running each sorting routine on 1000 numbers. (Keep the output to show for the checkpoint – can copy and paste into another file to save for later.)

2. Now, open `arrayInsertionSort.c` and comment out the `#defines` as appropriate, so that it sorts the preselected list of strings. Compile and run it. (Keep the output to show for the checkpoint.)

3. Now, open `arraySelectionSort.c` and modify the `#defines` as appropriate, so that it sorts the preselected list of integers. (Keep the output to show for the checkpoint.)

<u>Checkpoint 1 [20 points]: Show your lab instructor/assistant the answers to the questions above and results of your programs running.</u>

Part 4: Predictions

For the rest of this lab, you will be timing the sorting routines for various list sizes. Before doing this experiment, make a prediction as to which of the six sorting routines will be the fastest and which will be the slowest. Rank all six routines and put your prediction here (use isa, isl, ssa, ssl, qs, ms):

Fastest ______________

____

____

____

____

Slowest ______________

1. Modify `arrayInsertionSort.c` and `arraySelectionSort.c` so that they are back to the original code (sorting ints, getting a number from the command line).

Unix has a command called `time` that you can use to time commands. Try it:

```
time ./isa 10000
```

What is printed after the sorting results?

2. The <u>real</u> time is the real clock time. The <u>user</u> time is essentially the time the command is using the CPU. The <u>system</u> time is the time that the command is performing system calls in the CPU. You want to look at the <u>*user time*</u> for this lab.

Get **user times** (in seconds) for each executable on 20000 numbers:

isa: ____________________

isl: ____________________

ssa: ____________________

ssl: ____________________

qs: ____________________

ms: ____________________

3. Overall, do you think the implementations of selection sort and insertion sort are faster using arrays or linked lists (circle answer)?

Arrays Linked Lists

<u>**Checkpoint 2 [10 points]: Show your lab instructor/assistant the answers to the questions above.**</u>

Part 5: Timing Experiment

For the remainder of this lab, you will collect user times for the sorts, insert them into a spreadsheet (download from moodle) and graph the times. Note the code includes generating the data plus sorting the data.

You may want to use more shell windows to have multiple tasks executing at once. To get another shell window, type:

```
xterm &
```

Or you can use multiple tabs in Mobaxterm.

You can launch as many xterm windows as you want, so you can keep several commands executing at once and use both machines.

Experiment:

For N = 20000 to 200000 (by <u>increments of your choosing, but should have at least ten different values of N</u>)

Run ssa, isa, qs, and ms on lists of size N

Collect the user times for these (convert all times to total seconds)

Enter data into your excel spreadsheet.

Plot running times (line graph) for each sort, N is on x-axis and time (in seconds) is on y-axis, label each line with the sorting routine (put the plot below the timings in the main worksheet of the spreadsheet)

Name the file `lab9_results_username1_username2.xlsx`. Choose one partner to submit your excel file to moodle. Be sure both name(s) are in the file.

<u>Checkpoint 3 [40 points]: Show your lab instructor/assistant your graph.</u>

Part 5: (if time)

As you can see, quicksort is pretty quick, even though its worst-case performance is $O(N^2)$. How long does it take to quicksort 1 billion numbers?

____

You are finished. Close Mobaxterm and any other applications. Log off the computer.

If any checkpoints are not finished, they are due April 8. You may submit the spreadsheet to moodle. Answers to questions can be submitted via email or by hard copy.