CS 103 Lab - Rehearse To Recurse

1 Introduction

In this lab you will use recursion to implement a binary search algorithm to find a value in an array of data

2 What you will learn

After completing this lab you should be able:

- Implement a recursive function
- Understand the efficiency aspect of binary search vs. linear search
- Implement a 'selection' sort algorithm (maybe recursively)

3 Background Information and Notes

Binary Search

Review your lecture slides/notes regarding Recursion and the approach to binary search.

Remember we will specify the range of indexes to search by specifying the start and end index; start is inclusive, end is ***exclusive*** (i.e. 1 beyond the last location).

Be sure when you code your recursive functions:

1. Check your base case(s) first and 'return' an appropriate value (if any)
2. Be sure you return something appropriate from each function call, not just the base case
3. Be sure each recursive call ***makes progress*** (i.e. the range of indexes to search gets smaller for ***EVERY*** recursive call.

Selection Sort

Search any online resources available to you (i.e. search 'selection sort') to understand the basic idea behind the selection sort algorithm as you will be asked to code up a selection sort algorithm in this lab. Essentially, we search through the array to find the largest element and move it to the last index in the array (swapping with whatever number was already located there). Then we repeat the process of finding the maximum number but only through first n-1 elements (since the largest is already correctly located at the last element. We keep finding the maximum but of smaller and smaller array portions until the whole array is sorted. You may NOT directly copy/paste code from the Internet but you can certainly reference online implementations.

For extra credit you can attempt to write the selection sort algorithm recursively by using a loop to find the maximum and put it at the last location but then using recursion to process the list of size n-1 each time.

4 Procedure

4.1 [7 pts.] Binary Search

Download the binary search code skeleton.

```
$ wget http://bytes.usc.edu/files/cs103/lab-binsearch.tar
$ tar xvf lab-binsearch.tar
$ cd lab-binsearch
```

This will extract the following files:

```
binsearch.cpp           // skeleton code you will modify
sorted_data.in          // sorted list of integers to search
unsorted_data.in        // unsorted list of integers to search
```

Now make the following alterations:

1. Examine the list of integers in `sorted_data.in and unsorted_data.in` which your code will need to search through for a value provided by the user (via cin).
2. Open binsearch.cpp and examine and understand the skeleton code. As a point of interest understand the code that counts how many integers are in the file before reading them in.
3. Implement a recursive binary search algorithm in the binsearch() function.
    a. You may NOT use any explicit while/for loops…you must use recursion
    b. Remember the start index is inclusive but the end index is exclusive (i.e. 1 beyond the actual last index to be searched).
4. Compile your code (check for any warnings that may be generated).

    ```
    $ compile binsearch.cpp –o binsearch
    ```

5. Run your code and test it with some values that are and aren't present in the sorted_data.in file.

    ```
    $ ./binsearch sorted_data.in
    ```

6. Check the results to ensure it matched your expectation (make sure you have an expectation of what should be output).

Demo your code to your TA/CP.

4.2 [3 pts. + 2 EC for recursive implementation] Selection Sort

7. Recall that binary search only works on sorted data. So if we are given unsorted data we would first have to sort it before we can call binary search.
8. Uncomment line 48 which calls the sort() function (the skeleton code is shown below…uncomment that call to sort).

```
// Uncomment the line below for part 2
// sort(data, count);
```

9. Implement a selection sort algorithm in the sort() function that sorts the data in the array passed as an argument. It should sort the data in place (i.e. DON'T declare another blank array and fill that array in with sorted data).
    - Selection sort is easily implemented with two nested loops (for or while)
10. For 2 extra credit points implement a recursive implementation of selection sort. [Note: Sorting a list of size N can be achieved by placing the maximum value in the list at the last location (and moving the value there to some other location) and then sorting a list of size N-1 that doesn't include that last item.]
11. You are not allowed to change the prototype (add or remove parameters or return values) of the sort() function. You may however define a new helper function with alternate parameters/return values to help implement the recursion.

12. Compile your code (check for any warnings that may be generated).

```
$ compile binsearch.cpp –o binsearch
```

13. Run your code and test it with some values that are and aren't present in the unsorted_data.in file.

```
$ ./binsearch unsorted_data.in
```

14. Check the results to ensure it matched your expectation.

Demo your code to your TA/Sherpa.