Lab 08: Recursive Binary Search

CSci 515 Spring 2015

2015-02-18

Dates:

Due: In Lab, Wednesday March 25, by 4:05 pm (lab end time)

Objectives

- Become more familiar with performing searches and sorts.
- Practice passing arrays to and from functions.

Description

In class today we gave an example of a binary search implementation using a straight forward iterative approach. We kept track of the unsearched region of the array, and calculated the midpoint, and decided on a comparison with that midpoint whether to search to the lower or upper part of the array. Binary search can also be fairly easily described as a recursive algorithm. The basic idea of the algorithm is:

- Recursive condition: Calculate the midpoint of the current array and

compare the value to the search value. If the value is greater than the search value, recursively call the function on the subarray to the left of the midpoint. If the value is less than the search value, recursively call the function on the subarray to the right of the midpoint.

- Base case 1: If the value is equal to the search item, we found it so return the midpoint.
- Base case 2: If the subarray is empty, return a failure.

A specific description of the function/tasks to perform may make this clearer. You need to write a single function for this lab called `binarySearchRecursive`. This function takes an array of integers, and two indexes as its input parameters `low` and `high`. It also takes and integer value which is the value to search for within the indicated portion of the array. As in class, the `low` and `high` represent the indexs of the unsearched portion of the array. The function can implement a binary search recursively by performing the following steps:

1. If `low` > `high` then there is nothing left to search and the search has failed (base case 2). In this case, return -1 to indicate a failure of the search.

2. Calculate the midpoint location between `low` and `high` in the same was as was done in class.

3. If the value at the midpoint is equal to the value being searched for then return the midpoint location.

4. If the value at the midpoint is greater than the value being searched for then call `binarySearchRecursive` with `low` and `midpoint` - 1 (e.g. search the lower subportion of the array recursively).

5. If the value at the midpoint is less than the value being search for then call `binarySearchRecursive` with `midpoint` + 1 and `high`.

Your program output should look something close to the following when I run your program. In this example, I prompt the user for a value to search for and then display the location where the item was found. I run the program twice so we can see an example of a successful and a failed search.

```
$ ./lab08
Array, before being sorted:
000:  10
001:  11
002:  13
003:   1
004:  14
005:   9
006:   1
007:  13
008:   7
```

```
009:   4

Array, after being sorted sorted:
000:   1
001:   1
002:   4
003:   7
004:   9
005:  10
006:  11
007:  13
008:  13
009:  14

Enter a value and I will search for it in the array: 9
I found the value at location: 4

$ ./lab08
Array, before being sorted:
000:  19
001:   7
002:  15
003:  10
004:   3
005:  11
006:   4
007:  13
008:  16
009:  16

Array, after being sorted sorted:
000:   3
001:   4
002:   7
003:  10
004:  11
005:  13
006:  15
007:  16
008:  16
```

```
009:  19

Enter a value and I will search for it in the array: 5
Search failed, value: 5 not located in array
```

In your `main` function, show an example of calling your recursive binary search. Create an array of 10 values, presorted (you can use a static initializer to do this, or create an array of random values and copy and use a sort function from class today). Then call your recursive binary search on your array of 10 values, and show that it finds the location of the item being searched for.

NOTE: Now that our programs have more functions than just the `main()` function, the use of the function headers becomes meaningful and required. Make sure that all of your functions have function headers preceding them that document the purpose of the functions, and the input parameters and return value of the function.

Lab Submission

An eCollege dropbox has been created for this lab. You should upload your version of the lab by the end of lab time to the eCollege dropbox named `Lab 06 Processing Arrays`. Work submitted by the end of lab will be considered, but after the lab ends you may no longer submit work, so make sure you submit your best effort by the lab end time in order to receive credit.

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.

2. 50+ pts. Your program must have the required named function, that accepts the required input parameters and return the required values (if any).

3. 20+ pts. The function must be implemented correctly. The function must be working.

4. 30+ pts. Your main function must create an array and demonstrate calling the recursive binary search function correctly.

Program Style

Your programs must conform to the style and formatting guidelines given for this course. The following is a list of the guidelines that are required for the lab to be submitted this week.

1. The file header for the file with your name and program information and the function header for your main function must be present, and filled out correctly.

2. A function header must be present for all functions you define. You must document the purpose, input parameters and return values of all functions. Your function headers must be formatted exactly as shown in the style guidelines for the class.

3. You must indent your code correctly and have no embedded tabs in your source code. (Don't forget about the Visual Studio Format Selection command).

4. You must not have any statements that are hacks in order to keep your terminal from closing when your program exits (e.g. no calls to system()).

5. You must have a single space before and after each binary operator.

6. You must have a single blank line after the end of your declaration of variables at the top of a function, before the first code statement.

7. You must have a single blank space after , and ; operators used as a separator in lists of variables, parameters or other control structures.

8. You must have opening `{` and closing `}` for control statement blocks on their own line, indented correctly for the level of the control statement block.

9. All control statement blocks (if, for, while, etc.) must have `{ }` enclosing them, even when they are not strictly necessary (when there is only 1 statement in the block).

10. You should attempt to use meaningful variable and function names in your program, for program clarity. Of course, when required, you must name functions, parameters and variables as specified in the assignments. Variable and function names must conform to correct `camelCaseNameingConvention` .

Failure to conform to any of these formatting and programming practice guidelines for this lab will result in at least 1/3 of the points (33) for the assignment being removed for each guideline that is not followed (up to 3 before getting a 0 for the assignment). Failure to follow other class/textbook programming guidelines may result in a loss of points, especially for those programming practices given in our Deitel textbook that have been in our required reading so far.