CS 2500 Sample Report

Testing Recursion

Joe Miner, April 1, 2525

**Motivation**

Recursion is a useful method to express certain types of algorithms. However, recursive programs are difficult to understand, analyze, and debug. Recursive programs may be written as loop programs, but these are sometimes more difficult to understand. This lab will implement a common recursive problem in computer science, binary search, by direct recursion in C++ and as a loop program in C++. We want to see if one version is (1) easier to understand than another, (2) more efficient than another, and (3) easier to show correctness. The method of loop invariants for correctness will be extended to recursive programs.

**Background**

Conceptually, programs are step by step instructions to the computer on how to execute the solution to a problem. There have been many programming languages developed over the years (beginning largely with the FORTRAN language in 1958). These early languages supported looping (doing the same instructions over and over), computation, and decision making (branching). A later development was the notion of *recursion*, doing the same program over and over again on differing data sets. Indeed, Merriam-Webster defines recursion as

*a computer programming technique involving the use of a procedure, subroutine, function, or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first*

The definition goes on to compare this with iteration,

*the repetition of a sequence of computer instructions a specified number of times or until a condition is met*

Programs such as Binary Search are naturally expressed via recursion. Binary search involves looking for an item in a sorted list. It splits the list in two, then looks in the left or right hand list based on the values of the element at the split. If less than the split, the item is on the left, otherwise on the right. The program is then called on just the left (or right, but not both) half of the list (recursing). The process continues until the item is found or no more elements of the list exist to look in.

Binary search is known to have an asymptotic complexity of O(lg n). Asymptotic complexity means that as the problem size (n) gets large, the program’s run time takes on time proportional to the O() value. Beyond asymptotic complexity is the notion of program complexity – while two programs may have complexity O(lg n), one program may be less complex than the other, or have an actual run time of 64lg n vs. 2lg n for another. In this case, the constants 64 and 2 are proportional to the number of program statements executed. While there are many measures of program complexity, we will use program statements as our measure.

Correctness is showing that the program results in its postcondition, or system specification. Loop invariants are logical expressions that hold during loop executions or recursive program calls. At program termination, the loop invariant implies the postcondition. For the program to achieve its postcondition, its precondition must be true.

**Procedures**

In this laboratory, the following tasks will be performed:

1. Develop a postcondition and precondition for binary search,
2. show that binary search achieves its postcondition,
3. express binary search, its invariants, and its preconditions and postconditions in pseudocode for both recursive and iterative forms,
4. implement binary search in C++ as a recursive and as an iterative program,
5. implement preconditions, postconditions, and invariants using C++ assert() statements to validate correctness,
6. measure run times of the implemented programs and compare them with the expected O() complexity and experimentally determine the program complexity constants,
7. enumerate problems that were encountered during development,
8. develop and implement a testing plan, and
9. produce a conclusion addressing the efficacy of the methods used.

**Pseudocode and pre/postconditions**

*Here you express binary search, its invariants, and its preconditions and postconditions in pseudocode for both recursive and iterative forms.*

**Problems Encountered**

Constant memory leaks were present in the implemented code, initially. After re-allocating memory, these problems were solved. Another problem was trying to relate 1-based arrays from the pseudocode to 0-based arrays in C++. Implemented invariants detected where these problems were occurring by flagging invalid Initialization conditions.

**Testing Plan**

The following test plan was developed and implemented

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Outputs | Measured Outputs | Corrected Outputs |
| i<0 | Error Message | Segmentation Fault | Added Error Message and retested achieving expected outputs |
| i=0 | Item not found | Null | Item not found |
| i=1000000000 | Max array size exceeded | Segmentation fault | Message created for max array size and retested achieving expected outputs |
| A is n items sorted, v=16 in A[12] | i such that A[i]=v | A[12]=16 | NA |
| A is n items random values | Precondition failed | Item not found | Implemented precondition asserts and retested achieving expected outputs |
| A is n items sorted, v not in A | V not found in A | V not found in A | NA |
| More weird stuff you can think of…. |  |  |  |

**Performance Results**

The plot below shows the implementation of the recursive binary search (measured) vs the plot of lg n scaled by .032. As such, the complexity of the recursive version of binary search is .032. The asymptotic behavior begins (visually) at n0 between 141 and 281. There is one data point at 1600 that does not fit the expected curve, but preceding and ensuing data points fit, so this is probably an anomaly.

n0

*(then add another plot of the iterative form, and, if warranted, plots of best, worst, and average cases)*

**Conclusions**

In this lab, we showed that binary search meets its expected correctness analysis and meets its asymptotic complexity measurements. Testing validated the correctness analysis, although errors were still found in testing. While both versions met the same asymptotic complexity, the iterative version had a lower leading constant, indicating it performed better. However, in examining the psuedocode and C++ code, there are fewer statements in the recursion. This indicates that our proposed measure of complexity is not good.

Appendix A – Source Code

/\* Attach a listing of your code here \*/