IE 170 Laboratory 1: Search Algorithms

Dr. T.K. Ralphs

Due January 31, 2005

1 Laboratory Description and Procedures

1.1 Learning Objectives

1. Understand each of the key terms listed below.
2. Understand why algorithms are important in systems engineering.
3. Understand the three important factors in determining the execution time of an algorithm.
4. Understand how to analyze a simple algorithm, both empirically and theoretically.

1.2 Key Words

1. Algorithm
2. Efficiency
3. Running time
4. Search algorithm
5. Comparison

1.3 Scenario

You have just been hired as a consultant by the Internet start-up **ExoticBeer.com**, an on-line catalog supplying exotic imported beers and homebrewing supplies. Your first job is to help them improve the response time for queries to their on-line catalog. Specifically, you are to write a program implementing a *search algorithm* that determines whether a given search word that has been typed in by a customer matches any of the entries in an alphabetized list of key words describing items from the catalog (note that generating this alphabetized list requires a *sorting algorithm*, to be discussed in Laboratory 4). The company plans to carry thousands of items and each item may be referred to by several key words, so the number of key words in the list that must be searched can be extremely large. The goal is to return a response to the customer's query as quickly as possible and the number of queries received could be in the thousands per hour. If the execution time is too slow, then customers may be lost.

1.4 Designing and Analyzing the Algorithm

The *efficiency* of an algorithm is generally measured by determining the *running time* as a function of the *input size*. The running time for a given algorithm can be determined either *theoretically*, by counting the number of *fundamental operations* needed to execute the algorithm in theory, or *empirically*, by applying the algorithm in practice and determining the amount of time needed to execute it for a variety of inputs.

Empirically, the time it takes to execute an algorithm depends on three primary factors

- the problem instance,
- the efficiency of the algorithm (i.e., software), and
- the speed of the computer on which it is run (i.e., hardware).

We will take the speed of the hardware on which the algorithm will be executed as a given constant and will concern ourselves only with the efficiency of the algorithm itself. Theoretical analysis can provide us with an assessment of an algorithm's efficiency that is independent of hardware. However, performing this analysis usually requires simplifying assumptions and may not give us the complete picture. In this laboratory, we will perform both empirical and theoretical analyses and compare them to see if they agree.

As we just discussed, the theoretical analysis of an algorithm requires counting the number of *fundamental operations* that must be performed to execute the algorithm. Hence, the analysis depends inherently on what we consider to be a *fundamental operation*. In this laboratory, we will assume that comparing two strings is a fundamental operation that takes one time unit. We will consider the theoretical running time to be the number of comparisons required to carry out the search. Although this is a simplified model of the true situation, it will still allow us to effectively compare the efficiency of the two algorithms under study in this laboratory.

The number of comparisons needed in the worst case depends on the number of items in the list to be searched. Hence, we will take the input size to simply be the number of items in the list. Again, this is a simplified model, but acceptable for our purposes. How long it takes to search the list for a specific item depends on whether that items is, in fact, in the list, as well as where the item falls in the list.

1.5 Program Specifications

1.5.1 Program Function

Your program should perform the following functions:

1. Read in a (sorted) list of key words from a file (once).
2. Prompt the user for a search term.
3. Determine whether the search term is in the list.
4. Report the number of comparisons required to respond to the query.
5. When the key word submitted is "quit" or "exit," the program should end.

1.5.2 Algorithms

For this lab, you will implement and test two different search algorithms.

1. *Sequential search*: Compare each item in the list to the search term, in order, stopping when there is a match.
2. *Binary search*: Let n be the total number of items in the list.
   (a) If n is less than 4, perform a sequential search.
   (b) Otherwise, compare the search term to an item approximately in the middle of the list, e.g., item number $m = \lfloor \frac{n}{2} \rfloor$.
   (c) If the search term matches item m, then STOP.
   (d) Otherwise determine whether the search word occurs before or after item m in alphabetical order.
   (e) If the search term appears *before* item m, then perform binary search recursively on the first half of the list, i.e., from the beginning of the list to item $m - 1$.
   (f) If the search term appears *after* item m, then perform binary search recursively on the second half of the list, i.e., items $m + 1$ to the end of the list.

1.5.3 Data Structures

The basic data structure required for this laboratory is an array.

2 Laboratory Files

The test files for this laboratory are in the zip archive `Lab1.zip` available on the course Web site. The archive will unpack into a directory called `Lab1` with subdirectories `data`, `generator`, and `shell`. In the `shell` directory is a file named `search.cpp`, containing parts of the code need to complete the lab. You must fill in the missing pieces. In the `data` subdirectory, the files named `input*.txt` contain lists of randomly generated 10 letter "words" that can be searched to test your code. The name of the file indicates the number of words in the list. The number of words in the list is also the entry on the first line of the file. The files `target*.txt` contain randomly selected words from the list in each input file. You can test your code by searching for these words and ensuring that they are correctly found.

3 Laboratory Assignments

3.1 Programming (50 points)

1. Complete the file `search.cpp` and compile it to obtain a program that will perform either sequential or binary search, as desied.
2. Verify that your program compiles and runs correctly using the test files provided for the lab.

3.2 Analysis (40 points)

1. (10 points) Create graphs showing the number of comparisons needed to execute each of the algorithms as a function of the number of words in the list for the case when the word to be searched for is not in the list (to insure this, type in a word that is not 10 letters). You should include all the files `input*.txt` except for `input1M.txt` in your test. To create the graphs, use Excel and then paste the graph into your laboratory write-up.

2. (10 points) Determine theoretically the number of comparisons that would need to be made for each algorithm in the *worst case* as a function of the number of items in the list to be searched.

3. (5 points) Compare your empirical results from the first question to the theoretical number of comparisons computed in the second question. Do they agree? Why or why not?

4. (5 points) For the list `input1M.txt`, create a table showing the number of comparisons needed to locate each of the search terms in the file `target1M.txt`. Comment on how the running times change for different search terms for each of the algorithms.

3.3 Follow-up Questions (10 points)

1. (10 points) CLRS 1-1