BINARY SEARCH ON A LINKED LIST - A RESEARCH PROJECT IN EXPERIMENTAL COMPUTER SCIENCE

Marcin Paprzycki, Firooz Khosraviyani, Mark Wagaman
Department of Mathematics and Computer Science
The University of Texas of the Permian Basin
Odessa, TX 79762

0. INTRODUCTION

This investigation started as a student research project during a junior-level "Programming Algorithms" course. Now it will be presented as a basis for a variety of research projects that can be used in teaching a large group of computer science courses.

There are three major areas of research in computer science (CS). There is hardware design with its main goal of building faster computers to store and process larger amounts of data in shorter time. Theoretical CS; part of which deals with the design of efficient algorithms (so often without any considerations about the feasibility and possible effectiveness of implementation on existing hardware). Finally, there is the most underrated area of CS, experimental computer science. Experimental CS tries to bridge the gap between hardware and software design by establishing the best ways of implementing theoretically designed software on existing computers. It also tries to establish performance characteristics of the existing computers when running varieties of benchmarks.

A variety of such benchmarks can be found in the large area of searching and sorting algorithms. In the present paper, we discuss how a simple task of comparing the performance of different searching techniques can provide valuable insights into computer hardware, compiler design and comparison between programming languages. It is notable that due to the lack of a priori answers such a project can have the flavor of a "real research" project and thus, if only for this reason, attract the attention of students.

Section 1 describes the recently introduced technique of binary search on linked lists and presents some implementation details of this algorithm as well as an efficient sequential search algorithm. Section 2 presents the analysis of the proposed algorithm and shows how this analysis leads to some questions that need to be answered on experimental basis. Section 3 presents our experimental results and points to some research directions worth pursuing.

1. BINARY SEARCH ON LINKED LISTS

Until recently, it was generally agreed that nothing can be gained from implementing binary search algorithms on linked lists. In 1990, Khosraviyani presented a new algorithm and proved that there exists a way of taking advantage of binary search based technique to search linked lists [1]. More recently, a statistical analysis of the new algorithm was presented [2].

The binary search algorithm can be summarized as follows (for more details, the discussion of assumptions and the pseudo-code see [1]). In order to apply the binary search

principle to a linked list the middle of the list must be determined. This means that half of the list has to be sequentially traversed. Assume that the list is ordered and that its current length is known. A counter controlled loop can be used to find the middle of the list. A key field comparison is then used to select the half of the list to which the target key belongs. This process is then repeated on the retained half of the list.

Pointer assignments and key comparisons are the major factors influencing the complexity of the search algorithm. The binary search described above allows to reduce key field comparisons from $O(N)$ to $O(logN)$ but it pays for this reduction by $O(N)$ counter controlled loop iterations. If the counter controlled loop iterations are cheaper than key comparisons (which is usually assumed), this algorithm will be more efficient than a purely sequential search.

In the Fall Semester of 1990, during the "Programming Algorithms" course taught by one of the authors and taken by another, it was noted that a sequential search algorithm [3, pp. 194-197] (more efficient than the one used in [1] for comparison purposes) would outperform the algorithm as presented in [1]. It was only later, when a mathematical analysis of the algorithm allowed us to understand the reasons for the observed results.

2. ALGORITHM ANALYSIS

It is well known that the sequential search as described above uses an average of $N/2$ key comparisons for a successful search ([3] p. 197). The proposed algorithm, on the other hand, will on average have $O(logN)$ key comparisons and $N-1$ loop iterations ([1] p. 8). Let us denote by A the time for the key comparison, and by B — time for updating the loop counter. The average time of a successful search is described by the following formulas

$$Time_{Sequential} = (N/2)A$$
$$Time_{Binary} = AlogN + (N-1)B.$$

We would like to establish the conditions under which the binary search is more efficient than the sequential search. This leads to solving the following inequality

$$(N/2)A > AlogN + (N-1)B,$$

which can be reformulated as

$$(N/2 - logN)A > (N-1)B.$$

Let us now assume that $A = cB$ and try to estimate the value of c (c determines the point from which the binary search is more efficient than the sequential search). Simple calculations show that

$$c > \frac{N-1}{N/2 - logN}.$$

Assuming $N \to \infty$, we can see that in the limit $c > 2$. This result, has a very interesting implication. For the binary search to be more efficient than the sequential search, the cost of key comparison must be at least *twice* the cost of updating loop counter. It is likely that it is because c did not satisfy this condition that the sequential search proved superior to the binary search in the student research project.

This analysis leaves open the question about the actual values of c, which can lead to research in *different directions*. Below, we present some suggestions supporting them by the results of our experiments.

3. EXPERIMENTAL RESULTS

We performed our experiments on three different computers: **VAX 8200, MicroVax II**

and DEC Station 5000/200. We implemented both search algorithms in Fortran and Pascal. In both cases, the linked list structure was emulated using parallel arrays. All results presented here are averages of multiple runs (the differences between the runs were, however, rather small and varied from 3% for small to about 1% for large list sizes). Timings on VAX 8200 and MicroVax II were obtained using system timers (VAX VMS Library **LIB$**). MicroVax II was executing only our programs whereas VAX 8200 was performing standard duties as the central academic computer. On the DEC Station, our programs were running one at a time on an empty machine; the timings were obtained using the system provided routines. In all cases, the data was generated using system random number generators (**MTH$RANDOM** routine from the **MTH$** library on Vax'es and the **rand** routine on the DEC Station); we varied the value of the seed for different runs.

Our major goal was to compare the performance of the sequential search with the performance of the binary search. We used the search performed for every single element on the list of a given size (lists were presorted) for comparative purposes (and as a means of averaging the performance). Tables 1 and 2 present results obtained on the VAX 8200 for different data types used as keys. Table 1 presents the timings for the Pascal programmed searches. Table 2 presents results obtained when the algorithms were programmed in FORTRAN (for cross-language comparison purposes, only 4 byte integers are reported; the conclusions, however, carry over to FORTRAN's short integers). N specifies the list size; results are presented in seconds and parts of seconds.

	Binary Searches			Sequential Searches		
N	Integer	Real*4	Real*8	Integer	Real*4	Real*8
1000	10.9	10.8	11.5	7.3	7.9	9.6
2000	41.6	41.7	42.8	29.3	31.1	35.7
3000	96.9	95.6	96.2	69.7	72.9	88.1
4000	172.7	171.4	172.6	132.0	137.8	154.0
5000	272.9	270.3	277.3	235.0	243.2	237.4
10000	1110.6	1100.2	1107.8	976.9	1009.7	1011.5
15000	2493.4	2487.7	2503.8	2198.1	2266.6	2410.1
20000	4431.4	4421.6	4456.9	3904.6	4049.4	4511.0

Table 1. Total search implemented in Pascal on the VAX 8200.

	Binary Searches			Sequential Searches		
N	Integer	Real*4	Real*8	Integer	Real*4	Real*8
1000	3.7	4.2	3.6	5.2	5.6	6.3
2000	16.4	16.4	16.4	22.7	24.8	26.7
3000	39.6	39.5	39.1	54.9	61.0	61.2
4000	71.4	72.4	72.0	105.7	120.6	109.2
5000	114.1	115.4	115.6	172.6	198.6	171.0
10000	473.7	483.6	484.4	748.9	881.5	782.1
15000	1072.9	1092.8	1089.7	1722.9	2039.7	1940.5
20000	1910.3	1948.9	1945.2	3115.1	3661.2	3748.0

Table 2. Total search implemented in Fortran on the VAX 8200.

As was expected, tables 1 and 2 show that binary search behaves almost uniformly for different data types with only minimal increase of time as the key comparison time increases; whereas the sequential searches are key-comparison-time sensitive.

In Pascal, there is no advantage of using binary search over the sequential search for

both integers and short reals. Only for long reals and a large list size is there some advantage of using binary search. We calculated the ratio of time used by the binary search and the time of the sequential search. For both integers and short reals, the ratio remains stable for all list sizes suggesting that further increase in problem size will make no difference. For long reals, on the other hand, the ratio decreases steadily suggesting that further increase in problem size can increase the advantage of using binary search over the sequential search.

Results in Table 2 show that Fortran compiler produces a more optimized code. A substantial improvement in the loop efficiency can be observed. The binary search is much faster than the sequential one in all cases. Interestingly enough, the difference between the sequential searches in Pascal and Fortran is relatively small, whereas the binary search in Fortran is more than twice as fast as its Pascal counterpart.

The results on MicroVax II were similar to the results on VAX 8200. There were certain differences, however. In Pascal, the binary searches for all data types were slower than those on VAX 8200 with the biggest performance decline observed for long reals. All the sequential searches, on the other hand, were faster than those on VAX 8200. This caused all the sequential searches to be more efficient than the corresponding binary searches. The time ratio suggests that this relationship is independent of the problem size.

For the Fortran programs, both binary and sequential searches were faster on MicroVax II than those on VAX 8200. As for Pascal, binary search was not as consistent as on VAX 8200. A sudden performance drop was observed for both real data types. Binary search remains, however, more efficient than its sequential counterpart for all data types. As for VAX 8200 the sequential searches in Pascal and Fortran remained closer to each other than the binary searches.

Separately, we would like to mention the performance of the searching algorithms for the quadruple precision reals. This data type is an addition provided in the VAX VMS environment for both Fortran and Pascal. On VAX 8200, the performance of all programs followed the results presented in Tables 1 and 2. In the largest case (N=20,000), Pascal's binary search took 4483.1 seconds, whereas Fortran's binary search took 1951.2 seconds. Whereas the results for the sequential searches were 5669.2 (Pascal) and 4648.9 seconds (Fortran).

The outcomes were different on MicroVax II. When the binary searches were consistent with the remaining cases (though slightly slower than the double precision reals) an extremely large drop in performance was observed for the sequential search. When the total time of a sequential search (in the largest case — for both languages) remained below 1.5 CPU hours for the double precision reals, the quadruple precision sequential search took about 29.5 CPU hours — for both Pascal and Fortran! When contacted, Digital's customer service suggested, that the MicroVax II we used is equipped with a software emulated floating point accelerator and has some problems with quadruple precision arithmetics. In contrast, VAX 8200 has a hardware floating point accelerator which does not experience such problems. It looks as if this may also explain the remaining performance differences between the two computers.

Finally, we will present some results obtained on the DEC Station 5000. Tables 3 and 4 summarize the results that correspond to ones presented in Tables 1 and 2 above. Since the DEC Station RISC compilers provide 4 levels of optimization (from 0 to 3) and for the problem in question we found no difference between optimization 2 and 3, the results of optimization level 2 are presented.

	Binary Searches			Sequential Searches		
N	Integer	Real*4	Real*8	Integer	Real*4	Real*8
1000	0.17	0.18	0.19	0.25	0.27	0.32
2000	0.68	0.68	0.69	1.25	1.79	1.24
3000	1.52	1.50	1.52	2.20	2.66	2.77
4000	2.70	2.67	2.68	4.48	5.53	6.62
5000	4.16	4.11	4.17	8.71	9.52	10.92
10000	16.65	16.66	16.69	48.44	52.74	64.82
15000	37.52	37.44	37.54	132.47	137.15	172.83
20000	113.61	113.62	113.38	273.21	280.93	346.20

Table 3. Total search implemented in Pascal on the DEC 5000.

	Binary Searches			Sequential Searches		
N	Integer	Real*4	Real*8	Integer	Real*4	Real*8
1000	0.17	0.17	0.17	0.23	0.33	0.33
2000	0.67	0.67	0.68	0.95	1.97	2.06
3000	1.50	1.50	1.53	2.39	4.91	5.45
4000	2.68	2.67	2.70	5.49	7.82	9.53
5000	4.17	4.16	4.20	9.74	14.99	14.73
10000	16.58	16.62	16.64	47.86	62.79	66.63
15000	37.45	37.47	37.59	122.38	144.79	176.95
20000	113.44	113.24	113.61	249.45	289.16	355.93

Table 4. Total search implemented in Fortran on the DEC 5000.

It is clear that the binary searches are more efficient than the sequential ones. In addition, the results from binary searches are remarkably consistent and, surprisingly, there was no difference between Pascal and Fortran. The sequential search on integers was slightly faster when programmed in Fortran whereas Pascal outperformed Fortran somewhat for both short and long reals. For lower levels of optimization, however, Fortran was consistently more efficient than Pascal. It is also clear that the DEC Station 5000/200 is about 40 times faster than either VAX 8200 or MicroVax II.

4. CONCLUSIONS

We presented a set of experimental results comparing the performance of the binary search and the sequential search algorithms implemented on three computer architectures in two programming languages. We believe that a study of this kind can and should be used in teaching computer science as it exposes the student to the realities of programming. Since there are no predetermined answers students may find such a project interesting and challenging. As it is rather simple to program students can concentrate on analyzing the results, which is what they often do not but should focus their attention on.

Since no particular computer environment is required a project of this type can be completed in almost all schools. Depending on the course and the available computers, such projects may be focused on, for instance, algorithm comparison, hardware comparison, cross-language comparison, compiler-related comparisons, etc. Even in the smallest schools it is enough to have a PC and two programming languages to complete the simplest version of a similar project. It can be connected with the introduction of basic statistical techniques (and introduction to statistical packages) used in the analysis of results. We hope that this paper may inspire some teachers to develop groups of similar problems that will allow students get exposed to variety of important issues in the areas of experimental computer science.

REFERENCES

1. Khosraviyani, F., "Using Binary Search On A Linked List," *SIGCSE Bulletin*, Vol. 22, No. 3, Sept., 1990, pp. 7-10

2. Khosraviyani, F., Moadab, M.H., Hale, D.F., "Time Distribution Analysis for Binary Search of a Linked List," *SIGCSE Bulletin*, Vol. 23, No. 4, Dec., 1991, pp. 7-12

3. Sedgewick, R., *Algorithms*, Second Edition, Addison-Wesley, 1988