Pergamon

Computers Math. Applic. Vol. 36, No. 6, pp. 19–24, 1998

Printed in Great Britain
0898-1221/98 $19.00 + 0.00

PII: S0898-1221(98)00158-8

3 is a More Promising Algorithmic Parameter than 2

M. KAYKOBAD, MD. M. ISLAM AND M. E. AMYEEN
Department of Computer Science & Engineering
Bangladesh University of Engineering & Technology
Dhaka-1000, Bangladesh
<kaykobad><mamun>@csebuet.agni.com

M. M. MURSHED*
Computer Sciences Laboratory
Research School of Information Sciences & Engineering
Australian National University, ACT 0200, Australia
murshed@cslab.anu.edu.au

(Received April 1997; revised and accepted May 1998)

Abstract—In this paper we have observed and shown that ternary systems are more promising than the more traditional binary systems used in computers. In particular, ternary number system, heaps on ternary trees, and quicksort with three partitions do indicate some theoretical advantages over the more established binary systems. The magic Napierian e plays the crucial role to establish the results. The experimental data, supporting the analysis, have also been presented. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords—Analysis of algorithms, Performance evaluation, Quicksort, Heaps, Divide and conquer technique.

1. INTRODUCTION

With the invention of computers, two-parametric algebra, number system, and graphs among other systems started to flourish with accelerated speed. Boolean algebra got its important applications in computer technology, binary number system has occupied the core of computer arithmetic, and binary trees have become inseparable in mathematical analysis of complexity of algorithms and in the development of efficient algorithms.

Since 2 has been being used as a parameter having significant influence in the efficiency of the concerned algorithms, the claim of its supremacy over other values should be subject to rigorous verification. Megiddo [1] has placed an objection to the standard translation of problems into languages via the binary coding. Extensive works have already been done on optimality of ternary trees [2] and their VLSI embedding [3]. In this paper we have made simplified theoretical analyses on several problems, where 2 is being used as an algorithmic parameter, to see whether some other values are more promising. We have achieved some positive results in favour of 3 as an algorithmic parameter and these results have been supported by the experimental data.

*Author to whom all correspondence should be addressed.
The authors express their thanks to the anonymous referees and M. A. Sattar of CSE Department, BUET for their valuable comments.

In the next section we will establish the optimal value of d in d-ary systems by using some criteria that are suitable for specific cases of applications.

2. MATHEMATICAL ANALYSIS OF d-ARY SYSTEMS

We have chosen the following d-ary systems for analysis:

(i) d-ary number system,
(ii) d-ary heaps, and
(iii) d-Quicksort.

2.1. d-ary Number System

With the advent of computers and popularity of bistable electronic components, binary number system has firmly established its position in computer arithmetic. Although the following analysis must not have gone unnoticed in the literature, for the sake of completeness it has been presented below. Let us assume that the cost of a number system is proportional to the product of number of digits d in the system and number of digits required to express any arbitrary integer n. We wish to find the value of d for which the above product is a minimum. Ignoring integrality condition, we may assume that $\log_d n$ digits are sufficient to express an integer n in d-ary number system. Hence, the function to be minimized in this case is

$$f(d) = d \log_d n = \frac{d \ln n}{\ln d}. \tag{1}$$

Since d is independent of n, it is sufficient to minimize

$$f_1(d) = \frac{d}{\ln d}. \tag{2}$$

Now

$$f_1'(d) = \frac{\ln d - 1}{\ln^2 d} = 0, \tag{3}$$

from where we have $d = e$, the Napierian constant. Again

$$f_1''(d) = \left(\frac{1}{\ln d} - \frac{1}{\ln^2 d} =\right)' = -\frac{1}{d \ln^2 d} + \frac{2}{d \ln^3 d} = \frac{2 - \ln d}{d \ln^3 d}. \tag{4}$$

Hence, $f_1''(e) > 0$.

This means that the only stationary point at $d = e$ must be a minimum. Since we are looking for an integral value of d, it is sufficient to check for which of the values 2 and 3 of d, $d/(\ln d)$ is smaller. It is easy to verify that

$$\frac{2}{\ln 2} > \frac{3}{\ln 3} < \frac{4}{\ln 4}. \tag{5}$$

This supports that ternary number system should be better than the popular binary system, where criterion of optimality is similar to the one assumed here.

2.2. d-ary Heaps

Heapsort is a popular sorting algorithm, since its worst case and average case complexity has the same order of $\mathcal{O}(n \ln n)$ (see [4]). Recent advances in the heapsort algorithm through the works of Carlsson [5,6] and introduction of generalized heapsort by Paulik [7] have firmly challenged the superiority of quicksort or other sorting algorithms over heapsort. For example, heapsort has been proven to be the best sorting algorithm in terms of number of comparisons [6]. Moreover, heaps also have very useful applications in processing priority queues.

For simplicity of analysis, let us consider the worst-case complexity of a top-down variant of heapsort. We know that in order to find the proper place for the last unsorted element, $d-1$ comparisons are required to determine the youngest son and one more comparison to determine whether the element will be pushed down to another level. So in all for each level of push down, d comparisons are necessary. In the worst case, the element will be pushed down to the bottom, i.e., h levels, where $h = \lceil \log_d(n(d-1)+1) \rceil$. We are interested in finding the value of d for which

$$f_2(d) = ndh = nd\lceil \log_d(n(d-1)+1) \rceil \tag{6}$$

is minimized.

Assuming n to be very large and ignoring ceiling function, we have the following modified function:

$$f_2(d) = nd\,(\log_d n + \log_d(d-1)) \approx nd \log_d n = \frac{nd \ln n}{\ln d}. \tag{7}$$

Since now optimal value of d is independent of n, we have to minimize $d/(\ln d)$ for which the result is already known to us. The optimal value of d is again 3.

In Table 1, some experimental data are given in support of the above discussion. These data are taken from 100-run average. We have also considered the performance of four-heap, which was reported by Paulik [7] to outperform traditional two-heap. In both heap and quicksorts, we make comparisons between an element in the register and an element in the memory. So cost of comparison includes a data movement and a comparison. In case of movement, a data element is moved from a register to a memory location. So naturally, the cost of comparison is higher than the cost of movement. We have used 486-DX2 66 MHz machine with math co-processor and 16 KB internal cache and 256 KB external cache. We have found that the number of clock cycles required for 10^7 floating point comparisons and movements using cache are 193 and 121, respectively. So a single floating point comparison is equivalent to 1.6 floating point movement. For integer and double data, the corresponding factors are 1.125 and 1.005, respectively. Without using cache, the corresponding factors for integer, floating point and double data are 1.4, 1.39, and 1.42, respectively. In Table 1, *tmoves* indicates total number of equivalent moves for floating point data using cache. From Table 1, we can see that the more the comparison cost is prominent over movement cost, the more promising is ternary heap over binary heap.

2.3. *d*-Quicksort

Quicksort is a popular sorting algorithm. This algorithm works by partitioning the elements into two subgroups, where the elements of one subgroup is smaller than the partitioning element and the elements of the other subgroup are larger than the partitioning element. Now what will be the performance of the algorithm if the elements were partitioned into three subgroups? We can call this algorithm three-way quicksort. For this algorithm two partitioning elements are required. These elements are used to partition all the elements into three subgroups—first, middle, and last. The elements in the first subgroup are smaller than both the partitioning elements, those in the middle subgroup are larger than or equal to the first element but smaller than or equal to the second partitioning element and those in the last subgroup are larger than both the partitioning elements. To sort a list, its elements are partitioned this way recursively until no more partitioning is possible. Now, to find the average case performance of this three-way quicksort, we consider the following details of implementation of the above algorithm.

We choose the two boundary elements as the partitioning elements. The elements in the first subgroup will be scanned from left to right starting at the leftmost position. For the last subgroup, elements will be scanned from right to left starting from the rightmost position. The elements of the middle subgroup are scanned in both ways alternately starting from the middle of the list. The elements are compared with the partitioning elements to ascertain their position in the list. On the average, $3n/2$ comparisons are required. This can be shown as follows. To

Table 1. Performance of heapsort algorithms.

n	d	Move	Comp.	tmoves
100	2	832	1027	2475
	3	647	1007	2258
	4	580	1088	2321
200	2	1860	2454	5786
	3	1433	2418	5302
	4	1273	1606	3843
500	2	5295	7433	17188
	3	3998	7283	15651
	4	3504	7788	15965
1000	2	11575	16848	38532
	3	8598	16404	34844
	4	7517	17627	35720
2000	2	25153	37700	85473
	3	18521	36777	77364
	4	16005	37560	76101
5000	2	69602	107684	241896
	3	50446	99935	210342
	4	43239	104682	210730
10000	2	149201	235369	525791
	3	106838	214565	450142
	4	91773	233811	465871
15000	2	232446	370342	824993
	3	166319	343191	715425
	4	142019	363844	724169

partition an element into three way, at least one and at most two comparisons are required. For n, again assuming length of each partition to be distributed uniformly, the average number of comparisons, ${C_a}^3(n)$, in partitioning is

$$
\begin{aligned}
{C_a}^3(n) &= \frac{1}{n+1}\sum_{i=0}^{n}[2i+(n-i)] \\
&= \frac{3}{n+1}\sum_{i=0}^{n} i = \frac{3}{2}n.
\end{aligned}
$$

If the elements are within their subgroup, they are skipped; otherwise a flag is set to indicate the subgroup to which the element belongs. Initially all the subgroups are *activated*. All the activated subgroups will be scanned. When an element not belonging to the subgroup is found, that subgroup is deactivated and scanning begins at the next activated subgroup. After all the subgroups have been scanned, the elements are moved to their proper position according to the flags. Then the subgroups involved in the move are activated and the scanning is restarted. This process continues until all elements are compared and moved to their proper position. Then, the

three subgroups pass through the same procedure recursively until the list is sorted. The total cost for sorting n-elements satisfies the following relation:

$$T_a{}^3(n) = C_a{}^3(n) + \frac{2}{n(n-1)} \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} \left[T_a{}^3(i) + T_a{}^3(j) + T_a{}^3(n-2-i-j)\right]$$

$$= \frac{3}{2}n + \frac{6}{n(n-1)} \sum_{i=0}^{n-2} (n-1-i) T_a{}^3(i).$$

After simplification this equation reduces to

$$n(n-1)T_a{}^3(n) = 2(n-1)(n-2)T_a{}^3(n-1) - n(n-5)T_a{}^3(n-2) + 3(3n-4).$$

To find an approximation to $T_a{}^3(n)$, we assume $T_a{}^3(n) = \alpha n \ln n$ and find the best fit value for α using regression analysis. Using different values of n ranging from 1,000 to 10,000,000, the best value of α has been found to be approximately 1.8. If we use logarithm to the base 2, this figure comes to 1.248. For 2-quicksort [8], the value of α is 1.386, which is significantly higher than 1.248, the value of α for 3-quicksort.

To compare the relative performance of 2-quicksort and 3-quicksort, some experiments have been done, and the results are shown in Table 2. We have used i486SX 33 MHz based machine with 16 KB internal cache and 256 KB external cache. Floating point random data was used for the experiments. t_{tick} in Table 2 denotes the total clockticks required for execution. One clock tick is equivalent to 0.0346 sec.

Table 2. Performance of quicksort algorithms.

n	d	Moves	S.D.	Comp.	S.D.	t_{tick}
5000	2	37312	39.77	77342	330.34	4292
	3	49735	234.69	69594	280.89	4215
7000	2	54647	60.14	112674	437.19	6257
	3	72267	306.98	102237	371.38	6169
9000	2	72626	67.14	150573	640.89	8365
	3	95427	393.50	136402	475.87	8221
11000	2	91253	76.11	186958	715.01	10383
	3	119324	534.76	169504	541.88	10205
13000	2	110094	101.42	225870	852.61	12544
	3	142365	630.06	205494	743.56	12350
15000	2	129249	120.55	264727	1069.47	14709
	3	166372	627.89	240521	796.83	14413

This new algorithm requires larger number of moves, as elements of the middle subgroup cannot be moved to a definite position in the list. If the actual partitioning medians were known, this problem could not have arisen. Table 2 shows that 3-quicksort clearly outperforms the traditional 2-quicksort. It may be observed that the costlier the comparison with respect to movement is, the better the performance of the 3-quicksort than that of 2-quicksort.

3. CONCLUSION

Although analyses in the previous section have been rather simplified, thorough analysis of each example with efficient implementation of the corresponding algorithm is expected to retain the theoretical advantages of ternary systems over the more traditional and popular binary systems.

REFERENCES

1. N. Megiddo, Is binary encoding appropriate for the problem-language relationship?, *Theoretical Computer Science* **19**, 337–341 (1982).
2. F. Göbel and C. Hoede, On an optimality property of ternary trees, *Information and Control* **42**, 10–26 (1979).
3. S.S. Pinter and Y. Wolfstahl, Embedding ternary trees in VLSI arrays, *Information Processing Letters* **26**, 187–191 (1987).
4. R. Schaffer and R. Sedgewick, The analysis of heapsort, *Journal of Algorithms* **15**, 76–100 (1993).
5. S. Carlsson, Average-case results on heapsort, *BIT* **27**, 2–17 (1987).
6. S. Carlsson, An optimal algorithm for deleting the root of a heap, *Information Processing Letters* **37**, 117–120 (1991).
7. A. Paulik, Worst-case analysis of a generalized heapsort algorithm, *Information Processing Letters* **36**, 159–165 (1990).
8. E.M. Reingold and W.J. Hansen, *Data Structures*, Little Brown and Company, (1983).