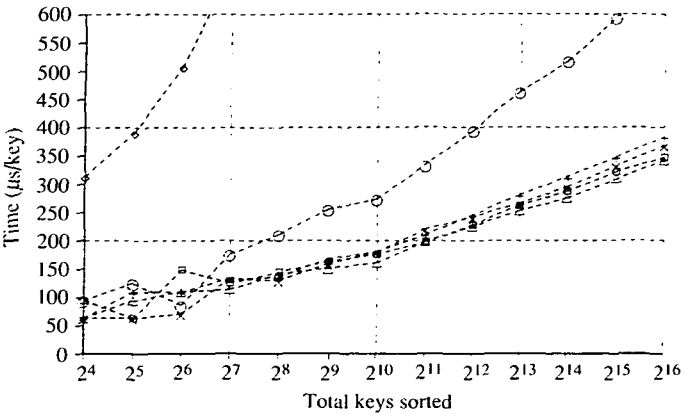

Figure 11. Time to Quicksort a 64-byte key.

6. CONCLUSIONS

This paper has presented an efficient implementation of the historical radix sort algorithm, and has shown that this implementation of a radix sort is superior to the more widely used Quicksort. In almost all experiments the radix sort was considerably faster than Quicksort, regardless of the number of keys being sorted, the size of these keys, and the contents of these keys. Out of a total of 312 different tests, Quicksort performed nominally better in only 5 tests, none of which involved sorting more than 64 keys.

There are cases where our radix sort should not be used. The partitioning is not stable, which may be significant in some applications. When performing external sorting, or in other applications where the cost of exchanging keys (or pointers to keys) is high, our algorithm can be expected to behave poorly. In applications where very little memory is available, the small amount of memory used by our radix sort might be viewed as excessive. In very critical applications, or those known to sort only a limited number of keys, other sort algorithms may be preferred. Finally, when sort keys cannot be divided into smaller keys having decreasing significance, no radix sort can be used.

Our radix sort can be used to sort text, binary values, and floating point values, in ascending or descending sequences, by merely employing encoding schemes which result in keys having the desired collating sequence. Our radix sort can also sort variable-length keys, if minor modifications are made to it. It would therefore be appropriate to consider using this sort in most practical applications.

Acknowledgements

I would very much like to thank Dr H. Bezner for motivating the study presented above, and for subsequently encouraging me to write this paper. I would also like to thank Dr D. J. Taylor for his continued interest in my research activities, and for his assistance in the production of this paper.

This research was funded by Wilfrid Laurier University. The production of this paper was funded, in part, by the Natural Sciences and Engineering Research Council of Canada, under grant A3078.

REFERENCES

1. Programmer's reference manual. In *UNIX System V Release 3*. AT&T (1986).
2. D. C. S. Allison and M. T. Noga, Usort: an efficient hybrid of distributive partitioning sorting. *BIT* **22**, 135–139 (1982).
3. J.-L. Baer and Y.-B. Lin, Improving quicksort performance with a codeword data structure. *IEEE Transactions on Software Engineering* **15** (5), 622–631 (1989).
4. C. M. Davidson, Quicksort revisited. *IEEE Transactions on Software Engineering* **14** (10), 1480–1481 (1988).
5. I. J. Davis, The MSQ database system. *HLSUA Forum XLIII Proceedings*, pp. 764–768 (5–8 October 1986).
6. I. J. Davis, The development of MSQ at Wilfrid Laurier. *Honeywell Bulletin* **5** (8), 6–20 (1987).
7. W. Dobosiewicz, Sorting by distributive partitioning. *Information Processing Letters* **7** (1), 1–6 (1978).
8. W. Dobosiewicz, The practical significance of D. P. sort revisited. *Information Processing Letters* **8** (4), 170–172 (1979).
9. G. H. Gonnet, *Handbook of Algorithms and Data Structures*. Addison-Wesley, London (1984).
10. C. A. R. Hoare, Quicksort. *Computer Journal* **5** (1), 10–15 (1962).
11. D. E. Knuth, Sorting and searching. In *The Art of Computer Programming*, p. 499. Addison-Wesley, Reading, Mass. (1973).
12. R. Loeser, Some performance tests on quicksort and descendants. *Comm. ACM* **17** (3), 143–152 (1974).
13. J. Rohrich, A hybrid of quicksort with $O(n \log n)$ complexity. *Information Processing Letters* **14** (3), 119–123 (1982).
14. R. S. Scowen, Algorithm 271, Quickersort. *Comm. ACM* **9** (5), 354 (1966).
15. R. Sedgewick, The analysis of quicksort programs. *Acta Informatica* **7**, 327–355 (1977).
16. F. Suraweera and J. M. Al-Anzy, Analysis of a modified address calculation sorting algorithm. *The Computer Journal* **31** (6), 561–563 (1988).
17. M. H. VanEmden, Algorithm 402, qsort. *Comm. ACM* **13** (11), 693–694 (1970).

Announcement

18–20 APRIL 1993.

RIDE-IMS '93 Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria. Sponsored by the IEEE Computer Society.

RIDE-IMS '93 is the third of a series of annual workshops on Research Issues in Data Engineering (RIDE). RIDE workshops are held in conjunction with the IEEE CS International Conferences on Data Engineering. Following the successful RIDE-IMS '91 held in Kyoto, Japan, the next RIDE workshop will also focus on interoperability of heterogeneous and autonomous database and knowledge systems.

The proceedings, consisting of the accepted papers, will be published by IEEE Computer Society and will be widely available.

For further information contact:
Elisa Bertino, Dipartimento di Matematica, Università di Genova. Tel.: +39-10-353-8034. Email: bertino@icnucevm.cnuce.cnr.it.
or
Susan Urban, Computer Science Department, Arizona State University. Tel.: +1-602-965-2784. Email: urban@asuvax.eas.asu.edu.