EXPECTED TIME ANALYSIS OF A INTERPOLATION MERGE – A SIMPLE NEW MERGING ALGORITHM

Sumanta GUHA
EE & CS Department, University of Wisconsin-Milwaukee, Milwaukee, WI 53201, USA

Arunabha SEN
Department of Computer Science, Arizona State University, Tempe, AZ 85287, USA

The binary merging algorithm is the best general purpose merging algorithm known to date. Binary merge consists of two components: a first component in which an array index is incremented by a number nearly equal to the ratio of the sizes of the two arrays being merged followed by a second component which is binary search. In this paper we formulate a simple algorithm called *interpolation merge*, where the binary search component is replaced with linear search, and analyze its expected behavior over data drawn from a uniform distribution. Our results, both theoretical and experimental, indicate a constant factor (≈ 0.75) speed-up over straight two-way merge. Further, our analysis of interpolation merge, which uses a mechanism of incremental indexing similar to that in binary merge, will hopefully lead to a better understanding of the latter algorithm. Currently, no significant facts are known about the expected behavior of binary merge over data drawn from any standard probability distribution.

Keywords: Analysis of algorithms, merging

1 Introduction

The binary merging algorithm due to Hwang and Lin [3], together with subsequent improvements (see, for example, [1, 6, 7]), is the best general purpose merging algorithm known to date. Binary merge consists of two components: a first component in which an array index is incremented by a number nearly equal to the ratio of the sizes of the two arrays being merged followed by a second component which is binary search.

We formulate a simple algorithm called *interpolation merge*, where the binary search component of the binary merging algorithm is replaced with linear search, and analyse its expected behavior over data drawn from a uniform distribution. Our results, both theoretical and experimental, indicate a constant factor (≈ 0.75) speed-up over straight two-way merge [5].

2 Algorithm

To motivate the algorithm, consider two sorted arrays A and B of m and n numbers, respectively, drawn with uniform probability from an interval $[\alpha, \beta]$. If $m > n$ then, expectedly, approximately $t = \lfloor m/n \rfloor$ elements of A lie between successive elements of B. Hence, having located an element of B in its proper place in the A sequence, we could expect the proper location of the next element of B to be after approximately another t elements of A.

In the following algorithm, doing "linear search downwards" from $A(i)$ to insert an element X in array A means successively comparing X with $A(i-1), A(i-2), \ldots$, until its proper place is located. Similarly define "linear search upwards".

Algorithm Interpolation_Merge

Input: Sorted array A of dimension m, sorted array B of dimension n and an increment t which is determined externally by the data distribution, as discussed above.

Output: Sorted array A which is a merge of the input arrays A and B.

begin

$i := t$; $j := 1$;

while $(j \leq n)$ **do**

begin if $B(j) < A(i)$ **then** do linear search downwards from $A(i)$ to insert $B(j)$ in its proper place between $A(p)$ and $A(p+1)$

else $\{B(j) \geq A(i)\}$ do linear search upwards from $A(i)$ to insert $B(j)$ in its proper place between $A(p)$ and $A(p+1)$;

$i := p + t$; $j := j + 1$

end{while loop}
end{algorithm}

3 Analysis

We make certain simplifying assumptions: A has nt elements, B has n elements, and all the $nt + n$ elements being distinct and drawn with uniform probability from some interval $[\alpha, \beta]$. Imagining A as "fixed", we see the elements of B vary with uniform probability in their location between elements in A. Consequently, the expected number of comparisons made by interpolation merge in this case is the same as that in the following: A is the array $[1, 2, \ldots, nt]$; B is, with equal probability, one of the $|\mathcal{B}|$ elements in the set $\mathcal{B}$ of all sorted subarrays, of length n and with not necessarily distinct elements, chosen from $[0.5, 1.5, \ldots, nt + 0.5]$. Here the fractional values are chosen merely to conveniently locate the elements of B between the elements of A.

Let $\mathcal{D}$ be the set of all sequences $D = [D(1), D(2), \ldots, D(n)]$ of n non-negative integers such that $\sum_{i=1}^{n} D(i) \leq nt$. Then there is a one-to-one correspondece between $\mathcal{B}$ and $\mathcal{D}$ given by $D \leftrightarrow B(D)$ where $B(D)(k) = 0.5 + \sum_{i=1}^{k} D(i)$.

It is easy to check that the number of comparisons made by interpolation merge with inputs A, $B(D)$ and t is precisely

$$Comp(D) \quad = \quad \sum_{\substack{1 \leq i \leq n \\ D(i) \geq t}} (D(i) - t + 2) + \sum_{\substack{1 \leq i \leq n \\ D(i) < t}} (t - D(i) + 1) \; .$$

In fact, the insertion of $B(D)(i)$ in its proper place in A requires $D(i) - t + 2$ or $t - D(i) + 1$ comparisons after the insertion of $B(D)(i-1)$—imagining the insertion of $B(D)(0)$ to be start of the algorithm—according as $D(i) \geq t$ or $D(i) < t$.

Hence, the expected number of comparisons made is

$$\begin{aligned}
Comp_{av} &= \frac{1}{|\mathcal{D}|}\sum_{D\in\mathcal{D}} Comp(D) \\
&= \frac{1}{|\mathcal{D}|}\sum_{D\in\mathcal{D}}\left(\sum_{\substack{1\le i\le n\\ D(i)\ge t}}(D(i)-t+2)+\sum_{\substack{1\le i\le n\\ D(i)<t}}(t-D(i)+1)\right) \\
&\le 2n+\frac{1}{|\mathcal{D}|}\sum_{D\in\mathcal{D}}\sum_{1\le i\le n}|D(i)-t| \\
&= 2n+\frac{1}{|\mathcal{D}|}\sum_{1\le i\le n}\sum_{D\in\mathcal{D}}|D(i)-t| \\
&= 2n+\frac{1}{|\mathcal{D}|}\sum_{1\le i\le n}\left(\sum_{j=0}^{nt}|j-t|\,|\{D\in\mathcal{D}:D(i)=j\}|\right) \\
&= 2n+\frac{1}{|\mathcal{D}|}\sum_{1\le i\le n}\left(\sum_{j=0}^{t-1}(t-j)\,|\{D\in\mathcal{D}:D(i)=j\}|+\sum_{j=t}^{nt}(j-t)\,|\{D\in\mathcal{D}:D(i)=j\}|\right). \quad (1)
\end{aligned}$$

Using elementary combinatorics we can check that

$$|\mathcal{D}| = \binom{nt+n}{n} \quad \text{and} \quad |\{D\in\mathcal{D}:D(i)=j\}| = \binom{nt-j+n-1}{n-1}. \quad (2)$$

From (1) and (2) we have

$$Comp_{av} \le 2n+\frac{n}{\binom{nt+n}{n}}\left(\sum_{j=0}^{t-1}(t-j)\binom{nt-j+n-1}{n-1}+\sum_{j=t}^{nt}(j-t)\binom{nt-j+n-1}{n-1}\right). \quad (3)$$

After some tedious manipulation of binomial cooefficients (see [4], especially formula (11) on page 54) we verify that

$$\begin{aligned}
\sum_{j=0}^{t-1}(t-j)\binom{nt-j+n-1}{n-1} &= (t-nt-n)\left(\binom{nt+n}{n}-\binom{nt-t+n}{n}\right) \\
&\quad + n\left(\binom{nt+n-1}{n+1}-\binom{nt-t+n+1}{n+1}\right) \quad (4)
\end{aligned}$$

and

$$\sum_{j=t}^{nt}(j-t)\binom{nt-j+n-1}{n-1} = (nt+n-t)\binom{nt-t+n}{n}-n\binom{nt-t+n+1}{n+1}. \quad (5)$$

Adding (4) and (5) and simplifying gives

$$\sum_{j=0}^{t-1}(t-j)\binom{nt-j+n-1}{n-1}+\sum_{j=t}^{nt}(j-t)\binom{nt-j+n-1}{n-1} = \frac{t(nt+n)!}{(n+1)!(nt)!}+\frac{2(nt-t+n)!}{(n+1)!(nt-t-1)!}. \quad (6)$$

Now, combining (3) and (6) we get

$$\begin{aligned} Comp_{av} &\leq 2n + \frac{n!nt!n}{(nt+n)!}\left(\frac{t(nt+n)!}{(n+1)!(nt)!} + \frac{2(nt-t+n)!}{(n+1)!(nt-t-1)!}\right) \\ &= 2n + \frac{n}{n+1}\left(t + \frac{2(nt)!(nt-t+n)!}{(nt+n)!(nt-t-1)!}\right). \end{aligned} \tag{7}$$

Using Stirling's formula [4] it may be calculated that, for large n,

$$\frac{(nt)!(nt-t+n)!}{(nt+n)!(nt-t-1)!} \approx (n-1)t\left(1+\frac{1}{t}\right)^{-t}. \tag{8}$$

So finally, with (7) and (8) we have, for large n,

$$\begin{aligned} Comp_{av} &\leq 2n + t + 2(n-1)t\left(1+\frac{1}{t}\right)^{-t} \\ &\approx 2n + t + \frac{2}{e}(n-1)t \qquad \text{(when t is large)} \\ &\approx 0.75nt + 2n + 0.25t. \end{aligned} \tag{9}$$

4 Conclusions

Equation (9) suggests that the expected speed-up of interpolation merge over two-way merge (where the expected time is almost exactly $nt+n$) is asymptotically a constant factor of about 0.75, at least with a uniform distribution. To confirm this we ran a simulation experiment on a Harris computer and obtained results that, in fact, agree well with equation (9). The process was as follows: ten sets, each of 50 pseudorandom numbers chosen between 0 and 1000, were generated with the Unix random number generator function Rand. Each set was sorted and merged, using an interpolation merge program written in C, with the sequence [1,2,...,1000] and the number of comparisons noted. The results of the experiment are shown in the following table.

Data set number	Comparisons in interpolation merge	Comparisons in two-way merge	Ratio of comparisons in interpolation merge to comparisons in two-way merge
1	767	1049	0.73
2	832	1049	0.79
3	637	1049	0.61
4	784	1049	0.75
5	831	1049	0.79
6	798	1049	0.76
7	887	1049	0.85
8	747	1049	0.71
9	714	1049	0.68
10	956	1049	0.91

It seems rather surprising that even when nt, the size of array A, is much larger than n, the size of array B, interpolating elements of B at intervals of length t in array A (the location we could "expect" to be about right?) followed by even linear search does not give more than a constant factor gain. Binary search seems to be the only winning factor in binary merge and even the worst-case guarantee of binary merge, which is $\lceil lg\binom{m+n}{m}\rceil + min(m,n)$ for arrays of size m and n [3], looks relatively satisfactory. We hope that, using techniques similar to those described here, a precise expected-case analysis of binary search itself may be obtained (something that is currently not known). Another interesting question is whether there is some naturally occuring distribution where the expected performance of interpolation merge (or some variant) is comparable to that of binary merge. In this case, the simplicity of implementing interpolation merge would make it an attractive alternative.

References

[1] C. Christen, Improving the bounds on optimal merging, *Proc 19th IEEE FOCS* (1978) 259-266.

[2] F.K. Hwang and S. Lin, Optimal merging of 2 elements with n elements, *Acta Informatica* **1** (1971) 145-158.

[3] F.K. Hwang and S. Lin, A simple algorithm for merging two disjoint linearly ordered sets, *SIAM J. Comp.* **1** (1972) 31-39.

[4] D.E. Knuth, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[5] D.E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[6] G.K. Manacher, Significant improvements to the Hwang-Lin merging algorithm, *J. ACM* **26** (1979) 434-440.

[7] M. Thanh and T.D. Bui, An improvement of the binary merge algorithm, *BIT* **22** (1982) 454-462.

[8] N.Y. Yousif and D.J. Evans, Merging by the parallel binary search algorithm, *Intern. J. Computer Math.* **22** (1987) 239-248.