ARTICLE

Average-case analysis of Leapfrogging samplesort

Eliezer A. Albacea

Institute of Computer Science, University of the Philippines Los Baños
College, Laguna, Philippines

The leapfrogging samplesort was introduced in 1995 but was not analyzed completely. In this paper, we analyze the algorithm in terms of expected number of comparisons. In particular, we obtain an estimated expected number of comparisons of $n \lceil \log (n+1) \rceil - 2^{\lceil \log (n+1) \rceil} - n + \lceil \log (n+1) \rceil + 1$ comparisons using the assumption that we estimate instead the cumulative distribution function of the input sequence from a sample. It should be noted that the results obtained in this paper is an estimate of the true expected number of comparisons. The problem of getting the true expected number of comparisons remains open.

KEYWORDS

Samplesort, quicksort, leapfrogging samplesort, random sample, probability, analysis of algorithms.

INTRODUCTION

Quicksort (Hoare 1962) is considered one of the algorithms that requires a minimal number of comparisons on the average. Under the assumption that all permutations of the input are equally likely, Hibbard 1962 showed that the expected number of comparisons using quicksort on an input sequence of size n is given by $1.39(n+1) \log n - 2.85n + 2.15$ (All logarithms throughout this paper are to base 2, unless otherwise stated explicitly.) Frazer and McKellar 1970 improved on this by introducing a samplesort algorithm that requires $n \log n + O(n \log \log n)$ comparisons on the average. More lately, Chen 2001 introduced PEsort which was analyzed by Cole and Kandathil 2004 to require an average of $n \log n + O(n)$ comparisons. Cole and Kandathil 2004 also introduced partition sort which also requires $n \log n + O(n)$ comparisons on the average. Finally, Chen 2006 introduced another samplesort-based algorithm that also requires $n \log n + O(n)$ comparisons on the average.

In this paper, we show analytically that the leapfrogging samplesort of Albacea 1995 requires an estimated $n \lceil \log (n+1) \rceil - 2^{\lceil \log (n+1) \rceil} - n + \lceil \log (n+1) \rceil + 1$ expected number of comparisons. The average-case analysis of leapfrogging samplesort is very difficult to do, however, with the assumptions and results of Frazer and McKellar 1970, specifically their Lemmas 1 and 2, we were able to finally give an estimate of the expected number of comparisons required in leapfrogging samplesort.

Leapfrogging Samplesort

Leapfrogging samplesort starts with *1* element and this is used as a sorted sample to partition the next *2* elements with the objective of producing a sorted sequence with *3* elements. Then, using the sorted *3* elements as a sample this is used to partition the next *4* elements in order to produce a sorted sequence with *7* elements. This is repeated until the whole sequence is sorted. One will notice that if the size of the sample is s, then this is used to partition the next $s+1$ elements.

Given a sorted sample of size s, the middle element of the

Email Address: eaalbacea@uplb.edu.ph
Submitted: December 6, 2011
Accepted: December 27, 2011
Published: January 30, 2012
Editor-in-charge: Amador C. Muriel
Reviewer:
Amador C. Muriel

sorted sample, say *v*, is used to partition the next *s+1* unsorted elements. When partitioning, those elements in the unsorted part that are less than *v* are moved to the left and those greater than *v* are moved to the right. This procedure is called the Partition procedure. Half of the sorted sample whose elements are greater than *v* are then moved to the left of the partition whose elements are greater than *v* using a procedure called Move Sample procedure. Thus, we produce two sets of elements, namely those greater than *v* and those less than *v*. Each set is prefixed by a sorted sample whose size is half the original size of the sorted sample. The partitioning and moving of a sample are then called recursively on both sets of elements.

An implementation of leapfrogging samplesort is given below.

```
void LFSamplesort(int first, int last)
{
   int s;
   if (last > first) {
      s = 1;
      while (s <= (last-first-s)) {
         Leapfrog(first, first+s-1, first+s+s);
         s += (s+1);
         }
      Leapfrog(first, first+s-1, last);
      }
}


void Leapfrog(int s1, int ss, int u)
{
   int i,j,k, sm, v,t;
   if (s1 > ss) LFSamplesort(ss+1, u);
   else
      if (u > ss) {
         sm = (s1+ss) / 2;
         /* Partition */
         v = A[sm];
          j = ss;
          for(i=ss+1; i <= u; i++) {
             if (A[i] < v) {
                j++;
                t = A[j];
                A[j] = A[i];
                A[i] = t;
                }
             }
         /* Move Sample */
         if (j > ss) {
            for (k=j, i=ss; i >= sm; k- -, i- -) {
               t = A[i];
               A[i] = A[k];
               A[k] = t;
               }
            }
         Leapfrog(s1, sm-1,sm+j-ss-1);
         Leapfrog(sm+j-ss+1, j, u);
         }
}
```

It is worth noting that the stages in leapfrogging samplesort, that is, a sorted sample is inserted in an unsorted part, is exactly the same as the second stage of the samplesort algorithm of Frazer and McKellar 1970 where a sorted sample is inserted in the remaining unsorted elements.

Expected Number of Comparisons

The analysis in terms of expected number of comparisons assumes that every time the algorithm is executed the elements of the sample denoted by *S*, which are used to partition the unsorted elements *U*, is a random sample of $(S \cup U)$. The sampling approach used in the sorting algorithm of Frazer and McKellar 1970 could be used to insure that we have a random sample. However, note that this assumption can easily be incorporated in the algorithm without additional comparisons. Assuming we have $n = 2^i - 1$ elements. First, we randomly choose a subset of size $2^{i-1}-1$ and place this in the first $2^{i-1}-1$ locations. Then, using this $2^{i-1}-1$ elements we randomly choose a subset of size $2^{i-2}-1$ and place this in the first $2^{i-2}-1$. We repeat this process until we have *3* elements where we choose a random sample of size *1* and place this in the first location.

With this assumption holding, we can use the following result of Frazer and McKellar 1970.

Lemma 1: When inserting the sample *S* of size *s* to an unsorted part *U* of size *u*, the expected number of elements in each partition is $u/(s+1)$.

Proof: This is a consequence of Lemma 1 of Frazer and McKellar 1970.

Let $S = \{s_1, s_2, ..., s_s\}$ be a randomly chosen subset of $S \cup U$ numbered such that $s_i < s_{i+1}$. Let $U = \{u_1, u_2, ..., u_u\}$ be the set of unsorted elements with items numbered such that $u_i < u_{i+1}$. When inserting the sample *S* to set *U*, this partitions the set *U* into *s+1* subsets, $U_0, U_1, ..., U_s$, where:

$$U_0 = \{ u \mid u < s_1 \},$$
$$U_i = \{ u \mid s_i < u < s_{i+1} \}, 1 \leq i < s,$$
$$U_s = \{ u \mid s_s < u \}.$$

In Lemma 1, the number of elements in U_i, where $0 \leq i \leq s$, is $u/(s+1)$.

Lemma 2: Leapfrogging samplesort will partition with a pivot element an unsorted sequence of size *u* using *u* comparisons.

Proof: This is obvious from the part of the code where partitioning is done. Specifically, the following part of the code that does partitioning is the following:

```
/* Partition */
v = A[sm];
j = ss;
for(i=ss+1; i <= u; i++) {
   if (A[i] < v) {
      j++;
      t = A[j];
      A[j] = A[i];
      A[i] = t;
      }
   }
```

Clearly, the for loop will iterate u times.

Theorem 1: The expected number of comparisons involved in leapfrogging samplesort is $n \lceil \log (n+1) \rceil - 2^{\lceil \log (n+1) \rceil} - n + \lceil \log (n+1) \rceil + 1$ comparisons.

Proof: Consider one stage of the sorting process where the Leapfrog procedure is given a sorted sample of size s and an unsorted part of size $s+1$. This means that the input size is $n = 2s+1 = 2^i-1$, where $i>0$.

The sorting process will first use the middle (median) of the sorted sample as a pivot to partition the $s+1$ unsorted elements and by Lemma 2 this will require $s+1$ comparisons. By Lemma 1, it is expected that the pivot element will divide the unsorted $s+1$ elements into two equal parts. Then, these two equal unsorted parts are each partitioned using the first quarter and third quarter elements of the sorted sample, respectively. This process is repeated until there is only one element in the sorted part and two elements in the unsorted part. By Lemma 1 after partitioning the unsorted part using all the elements of the sorted sample the expected size of each partition is 1.

Let $C(2s+1)$ be the expected number of comparisons required for executing the Leapfrog procedure on a sample of size s and unsorted part of size $s+1$. This gives

$$C(2s+1) = (s+1)+2\ ((s+1)/2\ )+4\ ((s+1)/4\ )+ \ldots + 2^{\log (s+1) - 1}\ ((s+1)/2^{\log (s+1) - 1}\ )$$
$$= (s+1) \log (s+1)$$

This is consistent with Lemma 2 of Frazer and McKellar 1970.

Note that leapfrogging samplesort does sorting for a sorted sample of size $s=1$ and unsorted part of size $s+1=2$, then repeated on sample size $s=3$ and unsorted part of size $s+1=4$, and so on until the sample of size $s=2^{\lceil \log (n+1) \rceil -1} -1$ and unsorted part of size h where $1 \le h \le 2^{\lceil \log (n+1) \rceil -1}$.

Hence, for any input n, the whole sorting process will require an expected number of comparisons of

$$C(n) = 2 \log 2 + 4 \log 4 + 8 \log 8 + \ldots + h \log 2^{\lceil \log (n+1) \rceil - 1}$$
$$= 2 \log 2 + 4 \log 4 + 8 \log 8 + \ldots + 2^{\lceil \log (n+1) \rceil - 1} \log 2^{\lceil \log (n+1) \rceil - 1} - (2^{\lceil \log (n+1) \rceil -1} - h)(\log 2^{\lceil \log (n+1) \rceil - 1})$$
$$= \sum_{i=1}^{\lceil \log (n+1) \rceil - 1} i\, 2^i - (2^{\lceil \log (n+1) \rceil -1} - h)(\lceil \log (n+1) \rceil - 1)$$
$$= \lceil \log (n+1) \rceil\ 2^{\lceil \log (n+1) \rceil} - 2(2^{\lceil \log (n+1) \rceil}) + 2 - (2^{\lceil \log (n+1) \rceil -1} - (n - 2^{\lceil \log (n+1) \rceil -1} +1))(\lceil \log (n+1) \rceil - 1)$$
$$= \lceil \log (n+1) \rceil\ 2^{\lceil \log (n+1) \rceil} - 2\ (2^{\lceil \log (n+1) \rceil}) + 2 - (2^{\lceil \log (n+1) \rceil} - n - 1)(\lceil \log (n+1) \rceil - 1)$$
$$= n \lceil \log (n+1) \rceil - 2^{\lceil \log (n+1) \rceil} - n + \lceil \log (n+1) \rceil + 1 .$$

CONCLUSIONS

What is crucial in the analysis of the algorithm is the generation of the random sample. As mentioned in Frazer and McKellar 1970, the objective is to generate a sample whose cumulative distribution function provides an unbiased estimate for the cumulative distribution function of the input sequence and thus ensure that the probability distribution of Lemma 1 of Frazer and McKellar 1970 is the appropriate one. But as mentioned earlier, this random sample can be obtained without additional comparison.

In this paper, we showed that the expected number of comparisons of leapfrogging samplesort is estimated to be $n \lceil \log (n+1) \rceil - 2^{\lceil \log (n+1) \rceil} - n + \lceil \log (n+1) \rceil + 1$ comparisons.

The problem of analyzing the algorithm in terms of exact number of comparisons on the average remains open.

REFERENCES

Albacea EA. Leapfrogging Samplesort, Proceedings of the 1st Asian Computing Science Conference, in Lecture Notes in Computer Science. 1995; 1023:1-9.

Chen JC. Proportion extend sort. SIAM Journal of Computing. 2001; 31(1):323-330.

Chen JC. Efficient sample sort and the average case analysis of PEsort. Theoretical Computer Science. 2006; 369:44-66.

Cole R, Kandathil DC. The average case analysis of partition sorts. in Proc of European Symposium on Algorithms, Bergen, Norway, 2004.

Frazer WD, McKellar AC. Samplesort: A sampling approach to minimal storage tree sorting. Journal of the ACM 1970; 17:496-507.

Hibbard TN. Some combinatorial properties of certain trees with applications to searching and sorting. Journal of the ACM 1962; 9:13-28.

Hoare CAR. Quicksort. Computer Journal 1962; 5:10-15.