A New Measure of Disorder in Sorting – Entropy

Tadao TAKAOKA

Department of Computer Science
University of Canterbury
Christchurch, New Zealand
E-mail : tad@cosc.canterbury.ac.nz

Abstract. Since information theory was used in establishing the lower bound of the complexity of sorting, there have not been many attempts to use information theory in adaptive sorting. In this paper we introduce the measure of entropy $H(X)$ for disorder in a list X of length n. The entropy measure is defined for ascending runs and up-sequences. Then we present new adaptive sorting algorithms, one called minimal merge sort, and its extension. Minimal mergesort merges the ascending runs in the input list from shorter to longer, that is, merging the shortest two lists each time. The extended algorithm first finds the minimum number of up-sequences and calls minimal mergesort. We show that these algorithms run in $(O(nH(X))$ worst case and expected times respectively, and minmal mergesort is optimal with respect to entropy. Finally we define more entropy measures and discuss their relationships with existing measures in adaptive sorting.

Keywords: adaptive sort, minimal mergesort, ascending runs, up-sequences, entropy

1 Introduction

Adaptive sorting is to sort the list of n numbers into increasing order as efficiently as possible by utilizing the structure of the list which reflects some presortedness. See Estivill-Castro, and Wood [1] for a general survey on adaptive sorting. There are many measures of disorder or presortedness. The simplest one is the number of ascending runs in the list. Let the given list $X = (a_1, a_2, \cdots, a_n)$ be divided into k ascending runs X_i $(i = 1, \cdots, k)$, that is, $X = (X_1, X_2, \cdots, X_k)$ where $X_i = (a_1^{(i)}, \cdots, a_{n_i}^{(i)})$ and $a_1^{(i)}$ is the $|X_1| + \cdots + |X_{i-1}| + 1$-th element in X. We denote the length of list X by $|X|$. Note that $a_1^{(i)} \leq \cdots \leq a_{n_i}^{(i)}$ for each X_i and $a_{n_i}^{(i)} > a_1^{(i+1)}$ if X_i is not the last list. The sort algorithm called natural merge sort [3] sorts X by merging adjacent two lists for each phase halving the number of ascending runs after each phase so that sorting is completed in $O(n \log k)$ time. Mannila [5] proved that this method is optimal under the measure of the number of ascending runs.

In this paper we generalize the measure $RUNS(X)$ of the number of ascending runs into that of the entropy of ascending runs in X, denoted by $H_{RUNS}(X)$.

Then we invent a sorting algorithm, called minimal merge sort, that sorts X by merging two minimal length runs successively until we have the sorted list. We show that the time for this method is $O(nH_{RUNS}(X))$ and is optimal under the measure of $H_{RUNS}(X)$.

Let $SUS(X)$ be the minimum number of up-sequences [4]. We generalize this measure into the entropy of up-sequences, $H_{SUS}(X)$. Our second algorithm first finds the fewest up-sequences, arrange them into ascending runs, and then calls minimal mergesort. We show that this algorithm runs in $O(nH_{SUS}(X))$ expected time and $O(n \log |SUS(X)|)$ worst case time. We refer to H_{RUNS} and H_{SUS} as *runs*-entropy and *sus*-entropy. Finally we define more general entropies and discuss their relationships with other measures in adaptive sort. We do not distinguish between $O(nH(X))$ and $O(n(H(X)+1))$ for entropy H for simplicity, although the latter is more precise. Logarithm is taken with base 2 unless otherwise specified.

2 Entropy of ascending runs

Let $n_i = |X_i|$ and $p_i = n_i/n$. Note that $\sum p_i = 1$. We define the entropy of ascending runs in X, $H_{RUNS}(X)$, by

$$H_{RUNS}(X) = -\sum_{i=1}^{k} p_i \log p_i.$$

Since p_i $(i = 1, \cdots, k)$ can be regarded as a probability measure, we have

$$0 \leq H_{RUNS}(X) \leq \log k$$

and the maximum is obtained when $|X_i| = n/k$ $(i = 1, \cdots, k)$. Hence the measure $O(nH_{RUNS}(X))$ derived from runs-entropy covers $RUNS$ measure $O(n \log k)$.

Lemma 1. *Any sorting algorithm takes at least $\Omega(nH_{RUNS}(X))$ time when the entropy of ascending runs in X is $H_{RUNS}(X)$ and $|X_i| \geq 2$ for $i = 1, \cdots, k$.*

Proof. Sorting X into $X' = (a'_1, \cdots, a'_n)$ where $a'_1 \leq \cdots \leq a'_n$ means that X' is a permutation of X. To establish a lower bound, we can assume that all elements in X are different. Let $X_i = (a_1^{(i)}, \cdots, a_{n_i}^{(i)})$. Let $a_{n_i}^{(i)}$ $(i = 1, \cdots, k)$ be fixed to be the i-th largest element in X. Then there are $\binom{n-k}{n_1-1}$ possibilities of X_1 being scattered in X'. Since the constraint of $a_{n_1}^{(1)} > a_1^{(2)}$ is satisfied by the choice of $a_{n_1}^{(1)}$, we have $\binom{n-k-n_1+1}{n_2-1}$ possibilities of X_2 being scattered in X'. Repeating this calculation yields the number of possibilities N as

$$\begin{aligned} N = & \frac{(n-k)!}{(n_1-1)!\,(n-k-n_1+1)!} \\ & \frac{(n-k-n_1+1)!}{(n_2-1)!\,(n-k-n_1-n_2+2)!} \\ & \cdots \frac{(n_{k-1}-1)!}{(n_k-1)!\,0!} \\ = & \frac{n!}{n_1! \cdots n_k!} \cdot \frac{n_1 \cdots n_k}{n(n-1)\cdots(n-k+1)}. \end{aligned}$$

Since the number of possible permutations is not fewer than this, we have the lower bound T on the computing time based on the binary decision tree model approximated by

$$\begin{aligned} T = \log N \geq n \log n - \textstyle\sum_{i=1}^{k} n_i \log n_i \\ + \textstyle\sum_{i=1}^{k} (\log n_i - \log(n-i+1)) \end{aligned}$$

where we use Stirling's formula. T is evaluated by

$$\begin{aligned} T \geq \textstyle\sum_{i=1}^{k} n_i \log \frac{n}{n_i} - k \log n + 2(k-1) \\ + \log(n-2k+2), \end{aligned}$$

since $\sum \log n_i$ is minimum when $n_1 = \cdots = n_{k-1} = 2$ and $n_k = n - 2(k-1)$. Now noting that the first term is minimum with the same condition, we have

$$\begin{aligned} & 2T - nH_{RUNS}(X) \\ & \geq \sum n_i \log \frac{n}{n_i} - 2k \log n + 4(k-1) \\ & \quad + 2 \log(n-2k+2) \\ & \geq 2(k-1) \log \frac{n}{2} + (n-2k+2) \log \frac{n}{n-2k+2} \\ & \quad - 2k \log n + 4(k-1) + 2 \log(n-2k+2) \\ & = (n-2k) \log \frac{n}{n-2k+2} + 2(k-1) \\ & \geq 0, \end{aligned}$$

since $1 \leq k \leq n/2$. Thus we have $T \geq nH_{RUNS}(X)/2 = \Omega(nH_{RUNS}(X))$. If we do not use Stirling's formula for small n_i's, we can still establish the theorem by using the integral approximation $n \log n - n + 1 \leq \sum_{j=1}^{n} \log j \leq n \log n - n + (1/2) \log n$ for n (1st inequality) and n_i (2nd inequality) and increasing the constant 2 of $2T$ at the lefthand side. ■

Note that if n_i=1 for all i, that is, sorted in reverse order, we have $O(nH_{RUNS}(X))$ $= O(n \log n)$, but we can sort the given list in $O(n)$ time.

3 Minimal mergesort

All lists are maintained in linked list structures in this section. Let $X = (X_1, \cdots, X_k)$ be the given input list such that each X_i is sorted in ascending order. Rearrange X into $X' = (X_{i_1}, \cdots, X_{i_k})$ in such a way that $|X_{i_j}| \leq |X_{i_{j+1}}|$ $(j = 1, \cdots, k-1)$, that is, $(X_1, \cdots, X_k)$ is sorted with $|X_i|$ as key. We call this "meta-sort." Since each $|X_{i_j}|$ is an integer we can obtain X' in $O(n)$ time by radix sort. Now we sort X' by merging two shortest lists repeatedly. Formally we have the following. Let M and L are lists of lists, whereas W_i $(i = 1, 2)$ and W are ordinary lists. By the operation $M \Leftarrow L$, the leftmost list in L is moved to the rightmost part of M. By the operation $W_i \Leftarrow M$ $(i = 1, 2)$ the leftmost list of M is moved to W_i. By the operation $M \Leftarrow W$, W is moved to the rightmost part of M. $First(L)$ is the first list in L.

ALGORITHM 3.1 (Minimal mergesort)

```
1  Meta-sort X into X' by length of X_i;
2  Let L = X';
3  M := ∅;
4  M ⇐ L;
5  if L ≠ ∅ then M ⇐ L;
6  for i := 1 to k − 1 do begin
7     W_1 ⇐ M;
8     W_2 ⇐ M;
9     W := merge(W_1, W_2);
10    while L ≠ ∅ and |W| > |first(L)| do
         M ⇐ L;
11    M ⇐ W
12 end
   {W is the sorted list}.
```

Theorem 2. *The algorithm minimal mergesort sorts $X = (X_1, \cdots, X_k)$ where each X_i is an ascending sequence in $O(nH_{RUNS}(X))$, precisely speaking $O(n(H_{RUNS}(X) + 1))$, time.*

Proof. Consider the moment when W_1 and W_2 are merged at line 9. Note that M and L are meta-sorted throughout the computation. From Lemma 3 (see below), we have $|W_2| \leq \frac{2}{3}|W|$ if W_2 is not an original list X_i for any i. Since $|W_1| \leq |W_2|$, W_1 and W_2 will go to a wider list, at least $3/2$ times as wide, if they are not original X_i's. Therefore each element in X_i will go to wider lists at most $\lceil \log_{3/2}(n/|X_i|) \rceil + 1$ times. Since at each merge at most $|W_1| + |W_2| - 1$ comparisons are performed, we can charge 1 comparison on each element in X_i if it is in one of the merged lists. Thus the total number of comparisons can be bounded by

$$\sum_{i=1}^{k} n_i \log_{3/2}(\frac{n}{n_i} + 1) = O(n(H_{RUNS}(X) + 1)).$$

■

Lemma 3. *If W_2 is not an original X_i for any i at line 9, it holds that $|W_2| \leq \frac{2}{3}|W|$.*

Proof. Suppose to the contrary that $|W_2| > 2|W_1|$. Then for the previously merged lists V_1 and V_2, that is, $W_2 = \text{merge}(V_1, V_2)$, we have $|V_1| > |W_1|$ or $|V_2| > |W_1|$. Thus V_2 or V_1 must have been merged with W_1 or a shorter list, a contradiction. ■

Example. Let $|X_1| = 2$, $|X_i| = 2^{i-1}$ $(i = 2, \cdots, k-1)$ and $n = 2^k$. Then minimal mergesort sorts X in $O(n)$ time, since $H(X) = const$, whereas natural mergesort takes $O(n \log \log n)$ time to sort X.

4 Finding up-sequences

Let X be decomposed into k up-sequences X_1, ..., X_k. Each $X_i = (a_1^{(i)}, \cdots, a_{n_i}^{(i)})$ is a monotone non-decreasing sublist of X, that is, elements are taken not necessarily consecutively from X and the union of X_i's is equal to X in the meaning of mutiset. The measure $SUS(X)$ is the minmum number of such sequences [4]. Let the entropy of up-sequences be defined smilarly to that of ascending runs. That is, for $p_i = n_i/n$ where $n_i = |X_i|$, we define

$$H_{SUS}(X) = -\sum_{i=1}^{k} p_i \log p_i.$$

Note that $H_{SUS}(X) \leq \log k$. We can similarly define $H_{SDS}(X)$ by decomposing X into the minimum number, $SDS(X)$, of down-sequences. Let $X_{SUS} = \{X_1, ..., X_k\}$ for $k = SUS(X)$. We show we can compute X_{SUS} in $O(nH_{SUS}(X))$ expected time. Then we can call minimal mergesort, so we can sort the given list in $O(nH_{SUS}(X))$ expected time. In the following algorithm, $FRONT$ is a one dimensional array containing endpoints of up-sequences so far obtained. Elements in up-sequences and endpoints are organized as pairs like $(i, a[i])$ to record the position of $a[i]$ and its value. We visualize up-sequences and $FRONT$ by putting $(i, a[i])$'s on a two-dimensional plane where i and $a[i]$ represent horizontal and vertical co-ordinates.

ALGORITHM 4.1 (Finding up-sequences)

```
 1  k := 1; { k : current number of upsequences}
 2  FRONT := {(1, a[1])}; X_1 := ((1, a[1]));
 3  for i := 2 to n do begin
 4  if a[i] < min{a[j]|(j, a[j] ∈ FRONT} then
 5    begin k := k + 1;
 6      Add (i, a[i]) to the end of FRONT;
 7      X_k := ((i, a[i])); mem[i] := k end
 8    else begin
 9      Let a[j] = max{a[l]|(l, a[l]) ∈ FRONT and a[l] ≤ a[i]};
10      m := mem[j];
11      Append (i, a[i]) to X_m;
12      Replace (j, a[j]) by (i, a[i]) in FRONT;
13    end
14  end
```

Let us name elements $(j, a[j])$ of $FRONT$ such that $a[j] \leq a[i]$ to be candidates. Any candidate $(j, a[j])$ can be connected to $(i, a[i])$ to grow the up-sequence to which $(j, a[j])$ belongs as an endpoint. We take the largest such $a[j]$ among candidates. The array mem shows the membership of each $(j, a[j])$ such that $j < i$. Let the current endpoint of X_l be $(i_l, b[l])$. Then we have

$b[1] > ... > b[k]$. Using this property, line 8 can be implemented in $O(\log k)$ time by binary search. Although Algorithm 4.1 is well known [2], the time analysis is only known to be $O(n \log n)$. We show its correctness and a sharper analysis in our terminology.

Theorem 4. *At the end of Algorithm 4.1, k is the minimum number of up-sequences.*

Proof. We prove by induction that at the end of the i-th iteration, k is the minimum number of up-sequences for array $a[1..i]$. Basis is obvious for $i = 1$. Assume the theorem is true up to $i - 1$. If $a[i]$ is not smaller than the smallest candidate, then we come to line 9 and $(i, a[i])$ will be appended to X_m without increasing the number of up-sequences, hence the theorem is true. Suppose $a[i]$ is smaller than the smallest candidate. Then we create a new up-sequence at lines 5-6. If the number of up-sequences is not minimum at this stage, we must be able to append $(i, a[i])$ to a previously formed up-sequence, say, X_l. Then X_l is split into two; Y_1 and Y_2 to form $(Y_1, (i, a[i]))$ and Y_2. Since Y_2 is not empty, we must be able to append Y_2 somewhere else. Whether we split Y_2 or not, we must be able to append at least the leftmost element of Y_2, say, $(j, a[j])$, to another up-sequence. When we appended this element to Y_1 at a previous step of the algorithm, there were no endpoints $(j', a[j'])$ such that $j' < j$ and $a[j'] \leq a[j]$. This follows from the fact that there are no endpoints $(i', a[i'])$ such that $a[i'] \leq a[i])$ and $(j, a[j])$ was appended to the largest, or highest in a geometrical sense, candidate. Now we face the same situation with that for $(i, a[i])$, that is, $(j, a[j])$ must be appended to a non-endpoint in order not to increase the number of up-sequences. This make-up process will proceed to the left with no success in reducing the number of up-sequences. ■

Theorem 5. *The expected running time of Algorithm 4.1 is $O(nH_{SUS}(X))$, and the worst case time is $O(n \log k)$ where $k = SUS(X)$.*

Proof. At the end of the i-th iteration, if $a[i]$ is smaller than the minimum of $FRONT$, we put $(i, a[i])$ to the right end of $FRONT$, taking $O(1)$ time. Suppose $a[i]$ is not smaller than the minimum. Then our probability assumption is that $a[i]$ is appended to X_l $(l = 1, ..., k)$ with equal probability $1/k$, spending $O(\log k)$ time by binary search. The value $i - j$ at line 9 is then expected to be not less than $k/2$. This is because $a[i]$ is chosen at random and has an equal probability of being connected with each of k endpoints. Let $pos(a_l^{(i)})$ be the position of $a_l^{(i)}$ in X. Let $pos(a_{l+1}^{(i)}) - pos(a_l^{(i)}) = N_{il}$. Then it takes $O(\log N_{il})$ time on average to expand X_i of length l by one. Since $\sum_{l=1}^{n_i} N_{il} \leq n$, we have

$$\sum_{l=1}^{n_i} \log N_{il} \leq n_i \log(n/n_i).$$

Since our assumption also regards the occurrence of $a[i]$ at line 4 as an independent trial, we can take this summation as the sum of expected times. Taking summation over i, we have the total expected time T as

$$T \leq \sum_{i=1}^{k} n_i \log(n/n_i) = O(nH_{SUS}(X)).$$

The worst case time is obvious. ■

Combining Algorithm 3.1 and Algorithm 4.1 yields the following algorithm for sorting.

ALGORITHM 4.2 (Sorting with up-sequences)

1 Call Algorithm 4.1 to obtain fewest up-sequences X_1,...,X_k;
2 Call Algorithm 3.1 to sort $X = (X_1, ..., X_k)$.

Theorem 6. *The lower bound for worst case time for sorting under decision tree model is* $\Omega(nH_{SUS}(X))$.

Proof. If we prepare ascending runs in Section 2 for Algorithm 4.1, these will be found to be the fewest up-sequences as well. Then the theorem follows in a similar way. ■

5 Relationships with other measures

In this section we discuss the relationships of entropy measures with other measures of presortedness or disorder in adaptive sorting.

Lemma 7. *Let* $X_{SUS} = \{X_1, ..., X_k\}$ *and* $n_i = |X_i|$. *Let* I *be the number of inversions in* X, *that is,* I *is the number of pairs* $(a[i], a[j])$ *such that* $a[i] > a[j]$. *Let* $X_{SUS} = \{X_1, ..., X_k\}$ *and* $n_i = |X_i|$. *Then we have*

$$\sum_{i=1}^{k} n_i \log i \leq n \log(I/n + 1).$$

Proof. Let I_j be the number of inversions caused by $a[j]$, that is, I_j is the number of $a[i]$ such that $a[i] > a[j]$. Then we have $I = \sum_{j=1}^{n} I_j$ and

$$\sum_{j=1}^{n} \log(I_j + 1) \leq n \log(I/n + 1).$$

Equality holds when all I_j are equal. Now observe the endpoints of X_1, ..., X_k are in decreasing order and $a[i]$ is appended to $a[j]$, the endpoint of X_m. That is, $I_j \geq m - 1$ if $a[j]$ belongs to X_m. Thus we have

$$\sum_{i=1}^{k} n_i \log i \leq \sum_{j=1}^{n} \log(I_j + 1) \leq n \log(I/n + 1).$$

■

Theorem 8.

$$O(H_{SUS}(X)) \leq O(\log(I/n + 1) + 1).$$

Proof. Let $\sum_{i=1}^{\infty} 1/i^2 = C$, where C is a constant, and $q_i = (1/i^2)/C$. Then $\sum_{i=1}^{k} q_i \leq 1$. Let $p_i = n_i/n$. Then, noting that $\log x \leq x - 1$,

$$\begin{aligned}
&-\sum_{i=1}^{k} p_i \log p_i - (-\sum_{i=1}^{k} p_i \log q_i) \\
&= \sum_{i=1}^{k} p_i \log(q_i/p_i) \\
&\leq \sum_{i=1}^{k} p_i (q_i/p_i - 1) \\
&= \sum_{i=1}^{k} q_i - \sum_{i=1}^{k} p_i \leq 0
\end{aligned}$$

Thus, using the lemma, we have

$$\begin{aligned}
&H_{SUS}(X) \leq -\sum_{i=1}^{k} p_i \log q_i \\
&= (1/n) \sum_{i=1}^{k} n_i \log C i^2 \\
&= (1/n) \sum_{i=1}^{k} 2n_i \log i + \log C \leq 2\log(I/n + 1) + \log C.
\end{aligned}$$

■

Corollary. Since the measure of inversions is given by $O(n(\log(I/n + 1) + 1))$, the measure of *sus*-entropy, that is, $O(n \log(H_{SUS}(X)))$ covers the inversion measure. Furthermore Algorithm 4.2 is inversion-optimal in the expected time sense.

Next we define more general entropies. Let $UP(X) = \{X_1, ..., X_k\}$ be an arbitrary decomposition of X into up-sequences X_1, ..., X_k. Let $\Gamma_{UP}(X)$ be the set of all such decompositions. Now the entropy of $UP(X)$ is defined by

$$H_{UP(X)}(X) = -\sum_{i=1}^{k} p_i \log p_i,$$

where $p_i = n_i/n$ and $n_i = |X_i|$. The up-entropy of X, $H_{UP}(X)$, is defined by

$$H_{UP}(X) = min\{H_{UP}(X) | UP(X) \in \Gamma_{UP}(X)\}.$$

Let $H_{DOWN}(X)$ and $H_{MONO}(X)$ be similarly defined by using down-sequences and monotone sequences. We call these entropies down-entropy and monotone-entropy. Obviously $H_{MONO}(X) \leq H_{UP(X)}$ and $H_{MONO}(X) \leq H_{DOWN}(X)$. It is also clear that $H_{UP}(X) \leq H_{SUS}(X)$ and $H_{DOWN}(X) \leq H_{SDS}(X)$, meaning that $O(nH_{UP}(X))$ covers the invertion measure. Furthermore we have the following.

Theorem 9. *The Rem measure is defined by $O(n+r \log r)$ where r is the number of the remaining elements after the longest up-sequence is removed [1]. Then it holds that $O(nH_{UP}(X))$ covers the Rem measure.*

Proof. We use natural logarithm in this proof for convenience. This only causes difference in constant factor. Let a decomposition $UP(X) = \{X_1, ..., X_k\}$ be given in such a way that X_k in the longest up-sequence in X. Let $n_i = |X_i|$. Then

$$\begin{aligned}
&H_{UP}(X) \leq H_{UP(X)}(X) \\
&= \sum_{i=1}^{k-1} n_i \log n/n_i + (n - r) \log n/(n - r)
\end{aligned}$$

$$\leq r \log((k-1)n/r) + (n-r) \log 1/(1-r/n)$$
$$\leq r \log n + (n-r)(r/n + r^2/(2n^2))$$
$$\leq r \log n + r.$$

Now subtracting the right-hand side from $n + r \log r$, we have

$$n + r \log r - r - r \log n = n - r + r \log(r/n)$$
$$= n(1 - r/n + (r/n) \log(r/n)) \geq 0,$$

since $0 \leq r/n \leq 1$ and $-x \log x \leq 1 - x$ for $0 \leq x \leq 1$. ■

6 Concluding remarks

If an algorithm runs in $O(nH(X))$ time for some entropy $H(X)$ and $O(nH(X))$ is the lower bound for sorting, we say the algorithm is $H(X)$ optimal.

When we scan X, we can identify ascending runs and descending runs alternatingly. By reversing descending runs, we can satisfy the condition of $|X_i| \geq 2$ in Lemma 2.1. This modified version of Algorithm 3.1 with this prescanning is thus optimal with respect to runs-entropy. It is open whether there is an algorithm for obtaining fewest up-sequences in $O(nH_{SUS}(X))$ worst case time, whereby Algorithm 4.2 can be $H_{SUS}(X)$ optimal. We also showed that the entropy measures cover several existing measures in adaptive sorting.

Future research will seek entropy optimal algorithms for various entropies.

References

1. V. Estivill-Castro and D. Wood, A survey of adaptive sorting algorithms, *ACM Computing Surveys* **24** (1992), 441–476.
2. M.C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, 1980.
3. D. E. Knuth, "The Art of Computer Programming, Vol.3, Sorting and Searching," Addison-Wesley, Reading, Mass., 1974.
4. C. Levcopoulos and O. Petersson, Sorting shuffled monotone sequences, Proc. 2nd Scandinavian Workshop on Algorithm Theory, Bergen, Sweden, July 1990, LNCS 447, 181-191.
5. H. Mannila, Measures of presortedness and optimal sorting algorithms, *IEEE Trans. Comput. C-34* (1985), 318–325.