Universal Pseudo Random List Sorting Algorithm

Daniel Diaz-Gonzalez[1,2]

[1] *UCL DCAL Research Centre - IT Officer*
[2] *University of London, Goldsmiths, University of London - Undergraduate BSc Computer Science*

November 1st, 2019

Proof of *feasibility* for any given input sequence **S**

Let define functions to build a finite sequence where the order of the elements is such as no two consecutive elements have the same property $c^{\dagger}$. The input sequence is partitioned in smaller finite sequences by the *commonality* of the property c in all elements of any sub-sequence.

Let h: A $\rightarrow$ D where A, D are finite sequences and h is a function that reverses the order of the elements in A. Such as:

$$h(a_1, a_2, a_3, \ldots, a_n) \rightarrow (a_n, a_{n-1}, a_{n-2}, \ldots, a_2, a_1) \text{ [1]}$$

Let g: E $\times$ F $\rightarrow$ H where E, F and H are finite sequences, $|E| \geq |F|$ and g is a function that pairs an element of E, e_i, i = 1 to $|F|$, with an element of F, f_i, up to the element $|F|$-*th*. In addition, concatenate the remainder elements of E, e_i, $i = |F| + 1$ to $|E|$ to that sequence. Such as:

$$g((e_1, e_2, \ldots, e_{|E|}), (f_1, f_2, \ldots, f_{|F|})) \rightarrow ((e_1, f_1), (e_2, f_2), \ldots, (e_{|F|}, f_{|F|}), (e_{|F|+1}, \ldots, e_{|E|})) \text{ [2]}$$

Thus $|H| = |E| + |F|$ [3].

It follows that the pairing function g can build a sequence where no two consecutive elements share the property c *if and only if* $|E| = |F|$ or $|E| = |F| + 1$. [4]

Let f: A $\times$ B $\rightarrow$ C where A, B and C are finite sequences, $|A| \geq |B|$ and f is defined as:

$$f(A, B) = g(h(A), B) \text{ [5]}$$

Let an initial *partition* of a finite sequence with *n finite sub-sequences* such as S = (S_1, S_2, S_3, ..., S_n), where every element in the *same sub-sequence* shared the property c, every sub-sequence has a *different* property c from each other and the *order of the sub-sequences* are of *decreasing cardinality*, thus:

$$|S_1| \geq |S_2| \geq |S_3| \geq \ldots \geq |S_n| \quad [6]$$

The function f can build a sequence from this initial sequence, where f is applied recursively to the sub-sequences of S in increasing cardinality, from S_n to S_1 [6], following the pattern:

$$f(S_{n-1}, S_n) \rightarrow T_{j,\, j=1 \text{ to } n-1};$$

$$\text{if } |T_j| \geq |S_{n-2}| \text{ then } f(T_j, S_{n-2}) \text{ else } f(S_{n-2}, T_j) \rightarrow T_{j+1};$$

$$\ldots$$

$$\text{if } |T_{n-1}| \geq |S_1| \text{ then } f(T_{n-1}, S_1) \text{ else } f(S_1, T_{n-1}) \rightarrow T_n \quad [7]$$

In addition, if the last application of f in the algorithm [7] follows the rule of g [4], where $|S_1| = |T_{n-1}| + 1$ or $|S_1|$ is smaller than $|T_{n-1}| + 1$, then all elements of S_1 are paired by g to elements of T_{n-1}.

Consequently, T_{n-1} elements from S_2 that were not paired with elements of T_{n-2} in the previous iteration of the algorithm [7] will be paired *first* with elements of S_1 thanks to the input order of g [2] and the application of the reverse order by function h [1].

In conclusion:

It is possible to build a sequence where no two consecutive elements have the same property *c if and only if:*

The *largest input sub-sequence* S_1 of the partition is equal or smaller, in the number of elements, than the sum of all the number of elements from all the other sub-sequences combined, plus one:

$$|S_1| \leq |T_{n-1}| + 1 \xrightarrow{[4]} |S_1| \leq |S_2| + |S_3| + \ldots + |S_n| + 1 \quad [8]$$

† The *elements* of the sequence are *tuples*. One *member* of each tuple is defined as the property *c*. The tuples with the *same value* in member *c* are elements of the same sub-sequence.

Universal pseudo random list sorting algorithm

Feasibility test:

(1) Order sequence **S** by *grouping* elements by property *c*, *counting* every group and *sorting* them by cardinality [6].

(2) Check feasibility of **S** by using the test [8].

(Only for the first time) Check initial sequence with steps (1) and (2). If they are accepted, build an output sequence in this order (until **S*** = ∅)‡:

(3) Pick a *random* element of **S**, $s*$.

(a) Check if output sequence **O** is empty (it is in the first iteration) *or* $s*$ has not the same property c as the last element of **O**.

(b) Check a new shorter sequence **S*** = **S** - {$s*$} with steps (1) and (2).

(4) If (a) *and* (b) are accepted then concatenate $s*$ to sequence **O** and go to step (3) with the new **S***.

(5) If (a) *or* (b) are rejected go to step (3) with the original **S**.

Correctness: The output sequence **O** has no two consecutive elements sharing the same property c, as follows:

Let b: O $\rightarrow$ {true, false}, a predicate function that checks **O** = $(o_1, o_2, o_3 \ldots o_{|S|})$, such as no element of **O** has the same property c as the immediate neighbors. It starts at $b(o_1)$. It follows from (4) that all elements of **O** accepted (a) *and* (b). Therefore, from (a) o_1 has no neighbors and from (b) **S** - $\{o_1\}$ is a *feasible* sequence, i.e., it is possible to build a sequence where no two consecutive elements have the same property c. Therefore, $b(o_1)$ is *true*. Iteratively, the function b tests the next elements $b(o_{n,\, n=2 \text{ to } |S|})$; from (a) it follows o_n and o_{n-1} are not sharing the property c, and from (b) it follows **S** - $\{o_1, \ldots, o_n\}$ is a *feasible* sequence, i.e., it is possible to build a sequence where no two consecutive elements have the same property c. Therefore, $b(o_n)$ is *true* for every element in the sequence **O**.

‡ The feasibility test guarantees the recursive loop will finish.

Variation of the algorithm: **O** accepts *k consecutive elements* with property *c*

Let the function g [2] to pair up to k elements of E with one element of F, where $|E| \geq |F|$, and concatenate the remaining unpaired elements of E to the end, such as:

$$g((e_1, e_2, \ldots, e_{|E|}), (f_1, f_2, \ldots, f_{|F|})) \rightarrow$$

$$\rightarrow ((e_1, e_2 \ldots e_k, f_1), (e_{k+1}, e_{k+2} \ldots e_{2k}, f_2), (e_{2k+1}, e_{2k+2} \ldots e_{3k}, f_3) \ldots$$

$$\ldots (e_{(|F|-1)k+1} \ldots e_{|F|k}, f_{|F|}), (e_{|F|k+1} \ldots e_{|E|}))$$ [9]

It follows that the pairing function g [9] can build a sequence where *up to k* consecutive elements share the property c *if and only if*:

$$|E| \leq k \cdot |F| + k.$$ [10]

Where the remaining *unpaired elements* of E at the end can be *at most k*.

Using this modified function g [9] as part of f [5], the same function h [1] as defined before, and the same recursive algorithm [7] over all the sub-sequences of the input sequence **S** [6], if the last step with f, S_1 and T_{n-1} follow the rule of g [10], we can conclude:

It is possible to build a sequence where up to *k consecutive elements* shared the property *c* *if and only if* the *largest input sub-sequence* S_1 is equal or smaller, in the number of elements, than *k times* the sum of all the number of elements from all the other sub-sequences combined, plus k [10]:

$$|S_1| \leq k \cdot (|S_2| + |S_3| + \ldots + |S_n|) + k \quad [11]$$

The changes in the new version of the sorting algorithm are:

> In step (2), check feasibility of **S** by using the new test [11].
>
> In step (a), check if output sequence **O** is empty (it is in the first iteration) or $|\mathbf{O}| < k$ or s^* has not the same property c as *all* last *k elements* of **O**.
>
> As before, if the step (a) is accepted (*true*) then step (b) will be evaluated.

It is trivial to show the proof [9] [10] [11] and the sorting algorithm step (a) are the same as the original if $k = 1$.

Correctness: The same predicate function b will check and accept the sequence **O**.

> $b(o_1)$ to $b(o_{k-1})$, iteratively, are accepted by (a): $|\mathbf{O}| < k$; and by (b): $\mathbf{S}^* = \mathbf{S} - \{o_1, \ldots, o_{k-1}\}$ is a *feasible* sequence where it is possible to build a sequence with up to *k consecutive elements* sharing the property *c*. For all the other elements of **O**, $b(o_k)$ to $b(o_{|O|})$, iteratively, are accepted by (a): o_{n-k} to o_{n-1} have not, *all of them*, the same property c as o_n; and (b): $\mathbf{S}^* = \mathbf{S} - \{o_1, \ldots, o_n\}$ is a *feasible* sequence where it is possible to build a sequence with up to *k consecutive elements* sharing the property *c*. Therefore, $b(o_n)$ is *true* for every element in the sequence **O**.

Notes: The feasibility test [8] [11] provides an upper and lower *bound* to the cardinality of each sub-sequence. If all sub-sequences are of the same cardinality [6] then the case is trivial and there exist many functions to pair the elements. However, to avoid any *pattern* in the output sequence, e.g. similar *distances* between elements with the same property *c*, or any pattern with any *grouping* of elements, the proposed algorithm produces a true random output sequence with only the desired constraint.

The *bounds* in the size relationship among the sub-sequences are *universal bounds for any other algorithm or pairing function* that builds an output sequence with only *k consecutive elements* of the same property. The proposed algorithm in this document is the simplest: pick a *random element* and check if the *remainder sequence* is still within the boundaries. Carry on until all *input elements* are sorted in the output.