# THE UGSORT ALGORITHM

Tree, Ian J.

unaffiliated researcher

## Author Note

Ian J. Tree, unaffiliated researcher

Eindhoven, the Netherlands

Email: ian.tree@acm.org

Abstract

This paper describes the novel UGSort merge sorting algorithm. The algorithm is simple, elegant, flexible and easy to implement.

*Keywords*: sort, algorithm, merge

# The UGSort Algorithm

The UGSort algorithm (Unexpectedly Good Sort) is a simple, elegant and flexible algorithm. It is easy to implement in different settings and is efficient.

## The Algorithm

The algorithm is a merge sorting algorithm based on partitioning the input keys into an array of double ended queues (dequeues) and then merging the double ended queues into a single queue from which the results of the sort can be output.

Sorting a set of keys KS = $\{k_1,k_{2...}k_n\}$ with n keys begins with initialising the first element of an array (S) of double ended queues $S_0$ with the first key available from the input set $k_1$, both the head and the tail of $S_0$ will be set to $k_1$.

Each subsequent key in the input set $k_i = k_{2...}k_n$ will be added to the partitions by testing against the head and tail keys in each of the available double ended queues in S. If the key $k_i$ is less than or equal to the head value of the queue, then the key is added to the head of the queue. If the key was not consumed by the head of the queue, then if it is greater than or equal to the tail value of the queue then, it is added to the tail of the queue. If the key was not consumed by any of the queues in S then a new queue is added to the end of the array initialised with the unused key $k_i$.

If a new queue was added to the array S then the number of elements in the array is checked against a computed maximum $S_{max}$ if it exceeds the maximum then a pre-emptive merge is performed. The value of $S_{max}$ is the greater of 100 or $(\frac{sqrt(\frac{i}{2})}{2})$, where $i$ is the number of keys processed so far. The pre-emptive merge will reduce the number of queues in the array S by merging alternate elements in the array with their predecessor. Thus, $S_1$ is merged with $S_0$ and removed from the array and so on for each alternate element in the array.

After the last key $k_n$ has been added to the queues in the array S a final merge is performed where all queues are merged into the first queue in the array $S_0$.

The input set KS is now sorted into ascending sequence in the queue $S_0$.

**Pseudocode**

```
Sort Input:          KS = {k1,k2...kn}
Partition Array:     S = {}
Max Partitions:      m = 100
S0 = double_ended_queue(k1)
ForEach ki in k2...kn
    ForEach Sj in S
        If ki <= Sj.head
            Sj.head = ki
            Key_consumed = true
            Exit ForEach
        End If
        If ki >= Sj.tail
            Sj.tail = ki
            Key_consumed = true
            Exit ForEach
        End If
        End ForEach
        If (!Key-consumed)
        Sj+1 = double_ended_queue(k1)
            If (j+1 > m)
                // Perform pre-emptive merge
                ForEach Sj in S
                    Merge Sj+1 into Sj
```

```
                                        Delete S_{j+1}
                                    End ForEach
                                    NewMax = (sqrt(i/2)/2)
                                    If (NewMax > m) m = NewMax
                                End If
                            End If
        End ForEach
        //  Perform final merge
        ForEach S_j in S_1...S_m
            Merge S_j into S_0
        End ForEach
```

**Key Stability**

Key stability is the sorting property of preserving the input sequence for keys of equal value. The UGSort algorithm as presented is not stable. However, it can be made stable by making minor changes to the algorithm. The tests for addition to the head or tail of a queue must be changed from less than or equal and greater than or equal to less than and greater than. The merge processing must always give precedence to the leftmost (lowest index) queue. These changes will result in a slightly less efficient algorithm.

**Best and Worst Cases**

The “Best-Case” input for the algorithm is an input set SK that is pre-sorted into descending sequence, this results in a single queue being used with all keys being added to the head of the queue.

The "Worst-Case” input for the algorithm is an input set SK that is pre-sorted into ascending sequence and then reordered by taking alternating keys from the head and the tail of the sorted queue. For KS = $\{k_1,k_{2...}k_n\}$ the worst-case input would be $\{k_1,k_n,k_2,k_{n-1...}\}$.

**Sort Properties**

As noted in the previous section the UGSort algorithm is not stable but, may be transformed into a stable sort by small modifications. The algorithm is not capable of in-place sorting and is therefore an out-of-place algorithm.

It is possible to introduce parallel processing for the pre-emptive and merge phases as individual merge operations can be done in parallel, there is no additional co-ordination needed for parallel merges.