NOTES ON BATCHER SORT

ERIC MARTIN

Given a natural number N, an algorithm meant to sort a list L of N integers is *nonadaptive* if it is equivalent to a sequence of instructions of the form:

```
order(L, i_0, j_0)
order(L, i_1, j_1)
...
order(L, i_k, j_k)
```

for some natural number k and some natural numbers i_0, j_0, i_1, j_1, ..., i_k, j_k smaller than N, where order() is defined as:

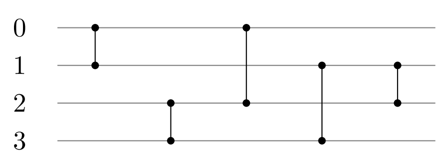
```
def order(L, i, j):
    if L[i] > L[j]:
        L[i], L[j] = L[j], L[i]
```

In other words, the same sequence of comparisons is performed to sort any list of length N, each comparison of two values resulting or not in a swap of those values.

For instance, if N is equal to 4, then such a sequence of instructions could be:

```
order(L[0], L[1])
order(L[2], L[3])
order(L[0], L[2])
order(L[1], L[3])
order(L[1], L[2])
```

(The smallest element is the least of the smallest of the first two and the smallest of the last two. The largest element is the greatest of the largest of the first two and the largest of the last two. The middle elements might have to be swapped.) It can be represented by the following *sorting network*:



The first two comparisons could be performed in parallel, and then the next two comparisons could also be performed in parallel.

Assume that the number N of data to be sorted is even. Recall that Merge sort splits an array in two halves, sorts both halves recursively, yielding two sorted halves H_1 and H_2, and then merges H_1 and H_2. When $N > 2$, merging can be done by first *unshuffling* the data, transforming (for $N = 8$)

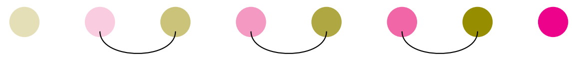
into

then ordering both new halves C_1 and C_2 separately

Date: Session 2, 2015.

then *shuffling* the resulting halves D_1 and D_2

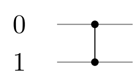
and finally ordering the second and third elements, the fourth and fifth elements...



Indeed:

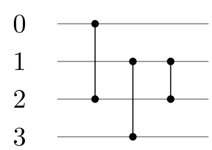
- After unshuffling, C_1 contains the smallest element from H_1, , and the smallest element from H_2, , with the smallest of the two, , being put into first position after shuffling.
- After unshuffling, C_2 contains the largest element from H_1, , and the largest element from H_2, , with the largest of the two, , being put into last position after shuffling.
- Assume that $N > 2$, and suppose that the first $2p + 1$ data, $0 \leq p < \frac{N}{2} - 1$, have been correctly put into place. What remains to put into place is in the sorted sequence $x_1, \ldots, x_i$ from D_1 and in the sorted sequence $y_1, \ldots, y_j$ from D_2, $i, j \geq 1$. Then x_1 and y_1 are ordered, clearly putting the smallest of the remaining data into place. Without loss of generality, assume it is x_1 and assume for a contradiction that y_1 is not the next smallest datum, so $i > 1$ and x_2 is the next smallest datum. If x_1 and x_2 both come from H_1 or both come from H_2, then there is some element in-between which belongs to C_2, hence has to be one of $y_1, \ldots, y_j$, which is impossible. Hence x_1 has to be the penultimate element from H_1, , and x_2 has to be the first element from H_2, , or the other way around, with the last element of H_1, , being one of $y_1, \ldots, y_j$. But then an even number of elements, namely, all elements of H_1 except for the last two and no others, have been put into place, which again is impossible. Hence the first $2p + 3$ data are correctly put into place.

Assume now that N is a power of 2. Note that if $N > 2$ then C_1 and C_2 themselves consist of two sorted halves– and for C_1, and for C_2–, hence the technique just described can be applied to the halves of C_1 and the halves of C_2 if $N > 2$, and to the halves of those halves if $N > 4$..., until we reach a problem of size two which is solved simply by executing a call to order():

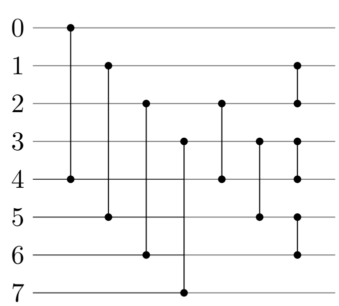


When using the network model, every recursive step after the base case amounts to "copying the pattern of the previous step" to both the network of even lines and to the network of odd lines, and then ordering the elements on lines 1 and 2, and if $N > 4$ the elements on lines 3 and 4 and the elements on lines 5 and 6, and if $N > 8$ the elements on lines 7 and 8...

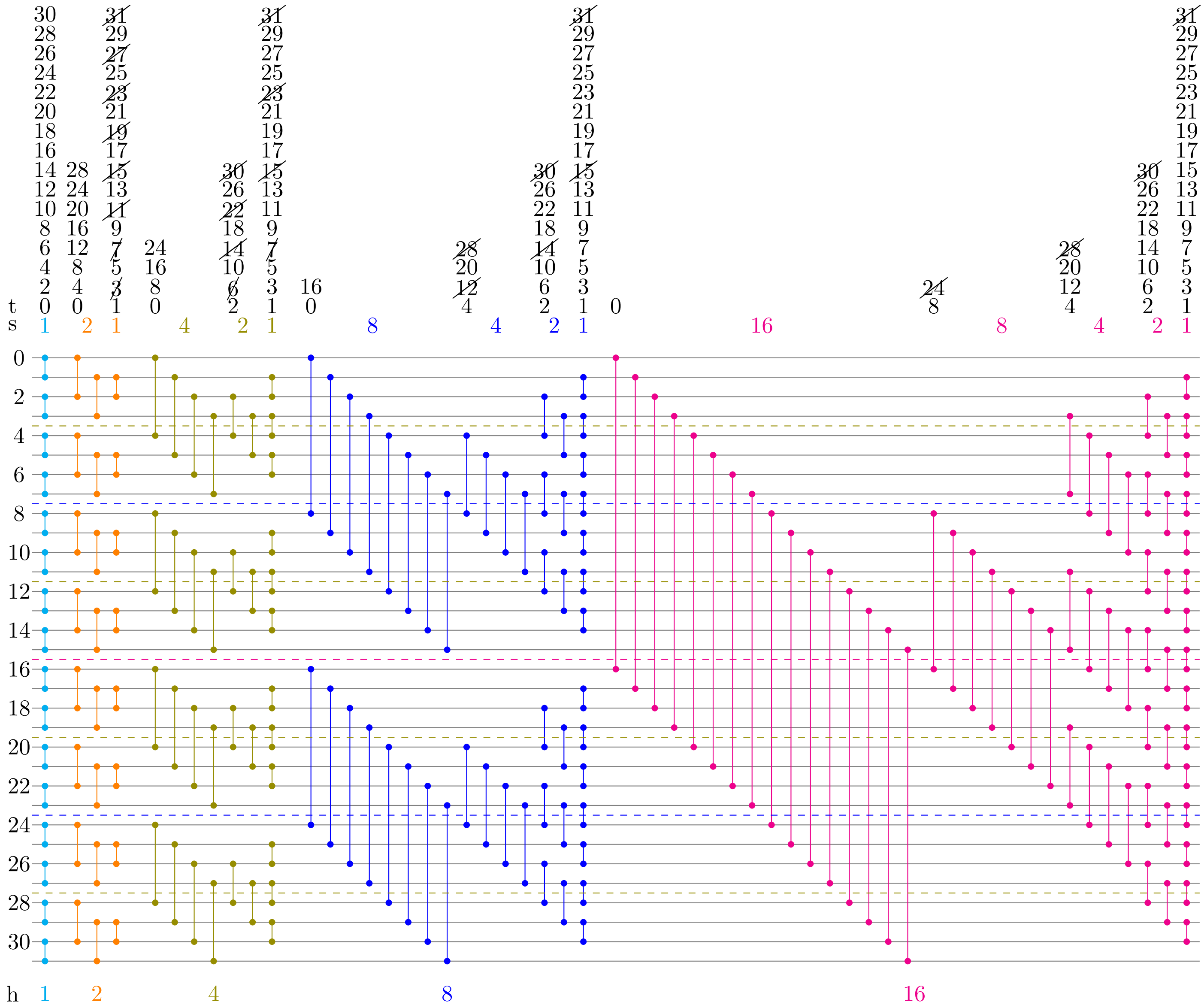
For $N = 4$:



For $N = 8$:



We see a pattern which emerges from merging two sorted arrays of size 2, two sorted arrays of size 4, two sorted arrays of size 8… Merge sort where merging is done as described can therefore be captured by this pattern for sorting two arrays of size $\frac{N}{2}$ if $N > 2$, following this pattern for sorting two arrays of size $\frac{N}{4}$ for both the first and second halves of the data if $N > 4$, following this pattern for sorting two arrays of size $\frac{N}{8}$ for the first, second, third and fourth quarters of the data if $N > 8$…



`h` is for `half_size`, `s` for `span` and `t` for `top`.

`group` is the index of a "group of lines" some of which will be skipped: we skip every `skip`th group.

There are $\log_2(N)$ possible values for `h`. For each possible value of `h`, there are at most $\log_2(N)$ possible values for `s`. For each possible value `h` and each possible value of `s`, there are fewer than N calls to order(), so the algorithm is in $O(N(\log_2(N))^2)$.