

Sorting networks

Bitonic sort

Bitonic sort [Bat 68] is one of the fastest sorting networks. A [sorting network](#) [Knu 73][CLRS 01] is a special kind of sorting algorithm, where the sequence of comparisons is not data-dependent. This makes sorting networks suitable for implementation in hardware or in [parallel processor arrays](#).

The sorting network bitonic sort consists of $\Theta(n \cdot \log(n)^2)$ comparators. It has the same [asymptotic complexity](#) as [odd-even mergesort](#) and [shellsort](#). Although a sorting network with only $O(n \cdot \log(n))$ comparators is known [AKS 83], due to its large constant it is slower than bitonic sort for all practical problem sizes.

In the following, bitonic sort is developed on the basis of the [0-1-principle](#) [Knu 73][CLRS 01]. The 0-1-principle states that a comparator network that sorts every sequence of 0's and 1's is a sorting network, i.e. it sorts every sequence of arbitrary values.

Basic concepts

Definition: A sequence $a = a_0, \dots, a_{n-1}$ with $a_i \in \{0, 1\}$, $i = 0, \dots, n-1$ is called [0-1-sequence](#).

A 0-1-sequence is called [bitonic](#),¹⁾ if it contains at most two changes between 0 and 1, i.e. if there exist subsequence lengths $k, m \in \{1, \dots, n\}$ such that

$$a_0, \dots, a_{k-1} = 0, \quad a_k, \dots, a_{m-1} = 1, \quad a_m, \dots, a_{n-1} = 0 \quad \text{or}$$

$$a_0, \dots, a_{k-1} = 1, \quad a_k, \dots, a_{m-1} = 0, \quad a_m, \dots, a_{n-1} = 1$$

In the following figure, different examples of bitonic 0-1-sequences are outlined, where 0's are drawn white and 1's gray.

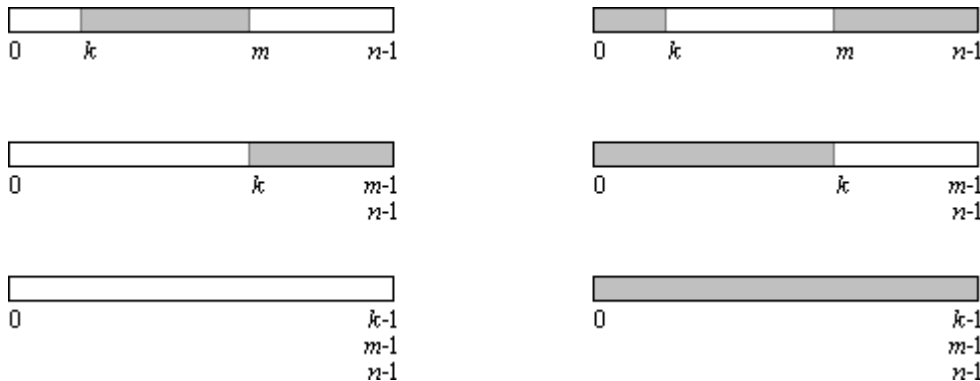


Figure 1: Some examples of bitonic 0-1-sequences

Definition: Let $n \in \mathbb{N}$, n even. The comparator network B_n is defined as follows:

$$B_n = [0 : n/2] [1 : n/2+1] \dots [n/2-1 : n-1] \quad (\text{see example of Figure 2})$$

Example:

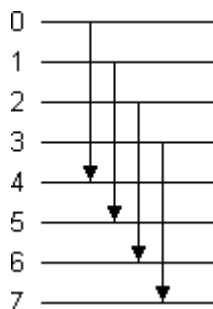


Figure 2: Comparator network B_8

Theorem: Let $a = a_0, \dots, a_{n-1}$ be a bitonic 0-1-sequence, where $n \in \mathbb{N}$, n even. Application of comparator network B_n to a yields

$$B_n(a) = b_0, \dots, b_{n/2-1} \quad c_0, \dots, c_{n/2-1}$$

where all b_i are less than or equal to all c_j , i.e.

$$b_i \leq c_j \quad \text{for all } i, j \in \{0, \dots, n/2-1\}$$

and furthermore

$b_0, \dots, b_{n/2-1}$ is bitonic and

$c_0, \dots, c_{n/2-1}$ is bitonic.

Proof: Let $a = a_0, \dots, a_{n-1}$ be a bitonic 0-1-sequence. Written in two rows, the sequence looks like shown in the following figure. The sequence starts with 0's, continues with 1's, and ends with 0's (Figure 3a). Or it starts with 1's, continues with 0's, and ends with 1's (Figure 3b). The regions of 1's may overlap or not.

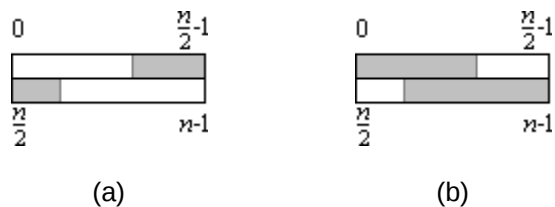
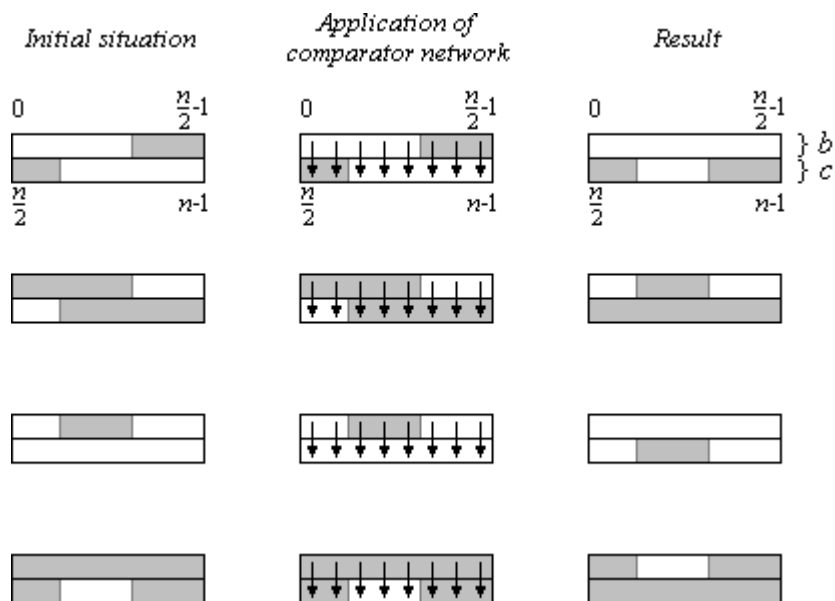



Figure 3: Bitonic 0-1-sequences (arranged in two rows)

Several other variations are possible (see Figure 4 below). Application of comparator network B_n corresponds to a comparison between upper and lower row. In each case, the result stated in the theorem is achieved: all b_i are less than or equal to all c_j and b is bitonic and c is bitonic (Figure 4):

Figure 4: Application of comparator network B_n to bitonic 0-1-sequences

Bitonic sorting network

The building blocks of the sorting network *BitonicSort* are comparator networks B_k with different k , where k is a power of 2. By using the divide-and-conquer strategy , networks *BitonicMerge* and *BitonicSort* are formed.

First, a comparator network *BitonicMerge* is built that sorts a bitonic sequence. Due to the theorem, B_n produces two bitonic subsequences, where all elements of the first are smaller or equal than those of the second. Therefore, *BitonicMerge* can be built recursively as shown in Figure 5.

The bitonic sequence necessary as input for *BitonicMerge* is composed of two sorted subsequences, where the first is in ascending and the other in descending order. The subsequences themselves are sorted by recursive application of *BitonicSort* (Figure 6).

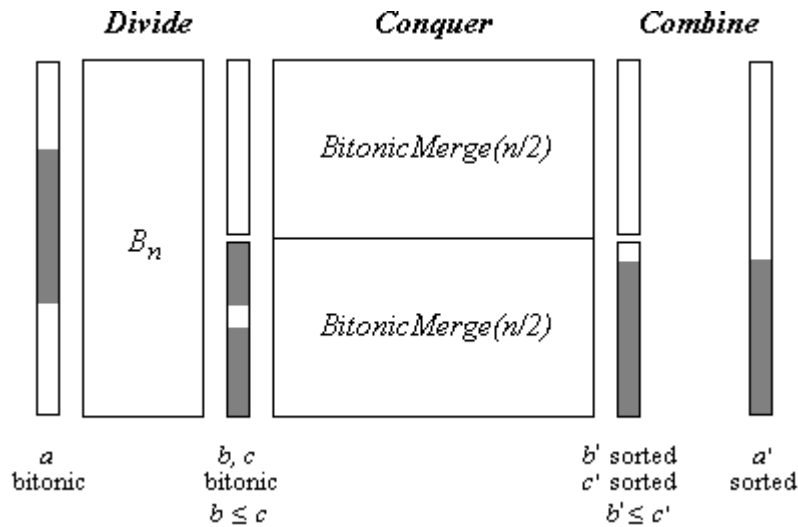


Figure 5: BitonicMerge(n)

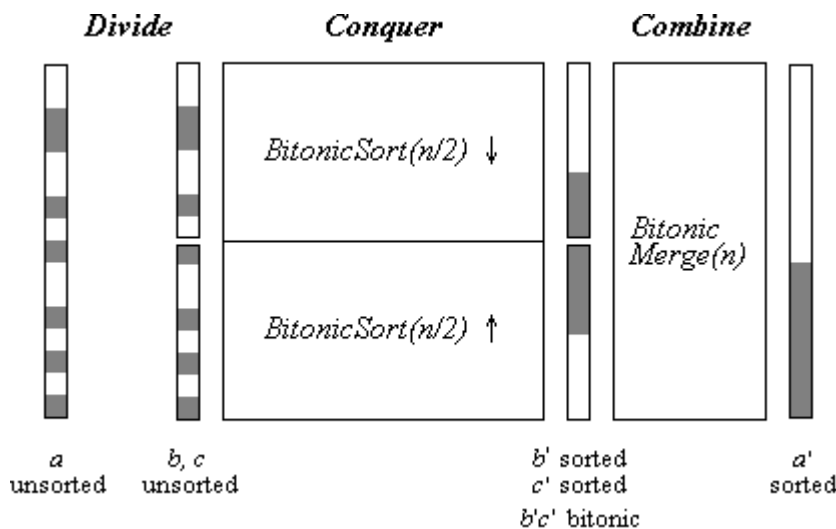
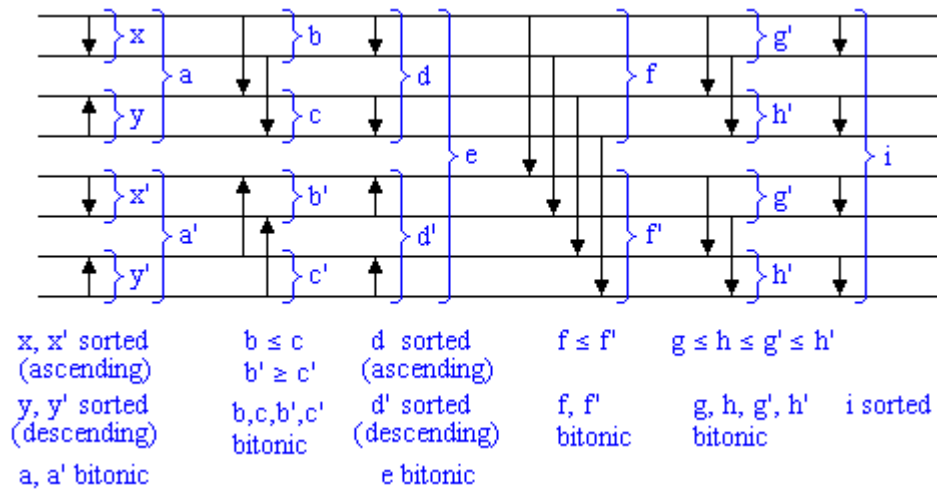


Figure 6: BitonicSort(n)

In the following Figure 7, as an example the sorting network *BitonicSort*(8) is given. The bitonic sequence e in the middle will be sorted by recursive application of B_k . The sequence e is bitonic, since it is composed of two halves d and d' which are sorted in opposite directions. These again are formed from bitonic sequences a and a' etc.

Given an arbitrary 0-1-sequence as input to the comparator network, the assertions stated in the figure will hold.

Figure 7: Sorting network BitonicSort for $n = 8$

At the output of the comparator network, the 0-1-sequence is sorted. Now the 0-1-principle can be applied to the comparator network as a whole: since the network sorts every arbitrary 0-1-sequence, it will also sort every sequence of arbitrary values, hence it is a sorting network.

Analysis

In order to form a sorted sequence of length n from two sorted sequences of length $n/2$, there are $\log(n)$ comparator stages required (e.g. the $3 = \log(8)$ comparator stages to form sequence i from d and d'). The number of comparator stages $T(n)$ of the entire sorting network is given by:

$$T(n) = \log(n) + T(n/2)$$

The solution of this recurrence equation is

$$T(n) = \log(n) + \log(n)-1 + \log(n)-2 + \dots + 1 = \log(n) \cdot (\log(n)+1) / 2$$

Each stage of the sorting network consists of $n/2$ comparators. On the whole, these are $\Theta(n \cdot \log(n)^2)$ comparators.

Program

In the following, an implementation of bitonic sort in Java is given. The program is encapsulated in a class *BitonicSorter*.²⁾ With the statements

```
Sorter s=new BitonicSorter();
s.sort(b);
```

an array b of length n is sorted with bitonic sort. The length n must be a power of 2 (cf. [bitonic sort for arbitrary \$n\$](#)).

Function *bitonicSort* first produces a bitonic sequence by recursively sorting its two halves in opposite directions, and then calls *bitonicMerge*.

Function *bitonicMerge* recursively sorts a bitonic sequence in ascending order, if *dir* = *ASCENDING*, and in descending order otherwise. The sequence to be sorted starts at index position *lo*, the number of elements is *n*.

A comparator is modelled by the function *compare*, where the parameter *dir* indicates the sorting direction. If *dir* is *ASCENDING* and $a[i] > a[j]$ is true or *dir* is *DESCENDING* and $a[i] > a[j]$ is false then $a[i]$ and $a[j]$ are exchanged.

```
public class BitonicSorter implements Sorter
{
    private int[] a;
    // sorting direction:
    private final static boolean ASCENDING=true, DESCENDING=false;

    public void sort(int[] a)
    {
        this.a=a;
```

```

        bitonicSort(0, a.length, ASCENDING);
    }

    private void bitonicSort(int lo, int n, boolean dir)
    {
        if (n>1)
        {
            int m=n/2;
            bitonicSort(lo, m, ASCENDING);
            bitonicSort(lo+m, m, DESCENDING);
            bitonicMerge(lo, n, dir);
        }
    }

    private void bitonicMerge(int lo, int n, boolean dir)
    {
        if (n>1)
        {
            int m=n/2;
            for (int i=lo; i<lo+m; i++)
                compare(i, i+m, dir);
            bitonicMerge(lo, m, dir);
            bitonicMerge(lo+m, m, dir);
        }
    }

    private void compare(int i, int j, boolean dir)
    {
        if (dir==(a[i]>a[j]))
            exchange(i, j);
    }

    private void exchange(int i, int j)
    {
        int t=a[i];
        a[i]=a[j];
        a[j]=t;
    }

} // end class BitonicSorter

```

Simulation

(Java applet for the simulation of bitonic sort)

Conclusions

We have presented the bitonic sorting network and proved its correctness. With $\Theta(n \cdot \log(n)^2)$ comparators this sorting algorithm is not optimal – the lower bound for sorting based on comparisons is in $\Omega(n \cdot \log(n))$. There are sorting algorithms like e.g. [heapsort](#) that achieve this lower bound. Nevertheless bitonic sort is interesting for realization in hardware or parallel processor arrays, since its sequence of comparisons is fixed and not dependent on data. The algorithm can be adapted to arbitrary n being not necessarily a power of 2 ([bitonic sort for arbitrary \$n\$](#)).

References

- [AKS 83] M. AJTAI, J. KOMLOS, E. SZEMEREDI: An $O(n \log n)$ Sorting Network. Proceedings of the 25th ACM Symposium on Theory of Computing, 1-9 (1983)
- [Bat 68] K.E. BATCHER: Sorting Networks and their Applications. Proc. AFIPS Spring Joint Comput. Conf., Vol. 32, 307-314 (1968)
- [CLRS 01] T.H. CORMEN, C.E. LEISERSON, R.L. RIVEST, C. STEIN: Introduction to Algorithms. 2nd edition, The MIT Press (2001)
- [Knu 73] D.E. KNUTH: The Art of Computer Programming, Vol. 3 - Sorting and Searching. Addison-Wesley (1973)

- ¹⁾ The term *bitonic* is a technical term meaning that the sequence is twofold *monotonic*.
- ²⁾ It is assumed that the class *BitonicSorter* implements an interface *Sorter* with the method *sort*. Otherwise, "**implements** *Sorter*" may be omitted.



[H.W. Lang](#) [Hochschule Flensburg](#) lang@hs-flensburg.de [Impressum](#) © Created: 05.03.1997 Updated: 29.05.2016