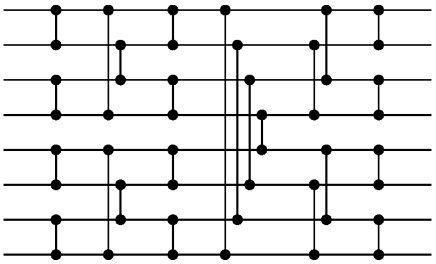
# Bitonic sorter

**Bitonic mergesort** is a parallel algorithm for sorting. It is also used as a construction method for building a sorting network. The algorithm was devised by Ken Batcher. The resulting sorting networks consist of $O(n \log^2(n))$ comparators and have a delay of $O(\log^2(n))$, where $n$ is the number items to be sorted.[1]



Visualization of the bitonic mergesort network with eight inputs

A sorted sequence is a monotonically non-decreasing (or non-increasing) sequence. A *bitonic* sequence is a sequence with $x_0 \leqq \ldots \leqq x_k \geqq \ldots \geqq x_{n-1}$ for some $k, 0 \leqq k < n$ , or a circular shift of such a sequence.

## Example code

The following is an implementation of the bitonic mergesort sorting algorithm in Python. The input is a boolean value *up*, and a list *x* of length a power of 2. The output is a sorted list that is ascending if *up* is true, and decreasing otherwise.

```
def bitonic_sort(up,x):
    if len(x)<=1:
        return x
    else:
        first = bitonic_sort(up,x[:len(x)/2])
        second = bitonic_sort(not up,x[len(x)/2:])
        return bitonic_merge(up,first+second)

def bitonic_merge(up,x):
    # assume input x is bitonic, and sorted list is returned
    if len(x) == 1:
        return x
    else:
        bitonic_compare(up,x)
        first = bitonic_merge(up,x[:len(x)/2])
        second = bitonic_merge(up,x[len(x)/2:])
        return first + second

def bitonic_compare(up,x):
    dist = len(x)/2
    for i in range(dist):
        if up and x[i] > x[i+dist] or \
            not up and x[i] < x[i+dist]:
            x[i], x[i+dist] = x[i+dist], x[i] #swap

>>> bitonic_sort(True,[10,30,11,20,4,330,21,110])
```

```
[4, 10, 11, 20, 21, 30, 110, 330]
>>> bitonic_sort(False,[10,30,11,20,4,330,21,110])
[330, 110, 30, 21, 20, 11, 10, 4]
```

The implementation in Java:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class BitonicMergeSort{

    static <T extends Comparable<? super T>> List<T> bitonic_sort(boolean up, List<T> list) {
        if (list.size() <= 1)
            return list;
        List<T> first = new ArrayList<T>(list.subList(0, list.size() / 2));
        first = bitonic_sort(up, first);
        List<T> second = new ArrayList<T>(list.subList(list.size() / 2,
list.size()));
        second = bitonic_sort(!up, second);
        first.addAll(second);
        return bitonic_merge(up, first);
    }

    static <T extends Comparable<? super T>> List<T> bitonic_merge(boolean up, List<T> list) {
        if (list.size() == 1)
            return list;
        bitonic_compare(up, list);
        List<T> first = new ArrayList<T>(list.subList(0, list.size() / 2));
        first = bitonic_merge(up, first);
        List<T> second = new ArrayList<T>(list.subList(list.size() / 2,
list.size()));
        second = bitonic_merge(up, second);
        first.addAll(second);
        return first;
    }

    static <T extends Comparable<? super T>> List<T> bitonic_compare(boolean up, List<T> list) {
        int dist = list.size() / 2;
        for (int i = 0; i < dist; i++) {
            int result = list.get(i).compareTo(list.get(i + dist));
            if (up && result > 0 || !up && result < 0) {
                T temp = list.get(i);
                list.set(i, list.get(i + dist));
                list.set(i + dist, temp);
            }
        }
        return list;
```

```
    }

    public static void main(String[] args) {
        List<Integer> original = Arrays.asList(10, 30, 11, 20, 4, 330, 21, 110);
        System.out.printf("Original List:\n %s \n", original);

        List<Integer> ascSorted = bitonic_sort(true, original);
        System.out.printf("\nSorted List (Ascending order):\n %s \n",
ascSorted);

        List<Integer> descSorted = bitonic_sort(false, original);
        System.out.printf("\nSorted List (Descending order):\n %s \n",
descSorted);
    }
}
```

## References

[1] http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm

## External links

- A discussion of this algorithm (http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm)
- Reference code (http://www.nist.gov/dads/HTML/bitonicSort.html) at NIST
- Tutorial with animated pictures and working code (http://www.tools-of-computing.com/tc/CS/Sorts/bitonic_sort.htm)
- Experimental Analysis of Parallel Sorting Algorithms (http://citeseer.ist.psu.edu/blelloch98experimental.html)

# Article Sources and Contributors

**Bitonic sorter** *Source*: http://en.wikipedia.org/w/index.php?oldid=371733494 *Contributors*: Antaeus Feldspar, Babicka baba, Bobo192, Booyabazooka, CJLL Wright, Cybercobra, Fred.Annexstein, GTBacchus, GregorB, NiklasL, Octotron, RainbowCrane, Xaosflux, 17 anonymous edits

# Image Sources, Licenses and Contributors

**File:Batcher Bitonic Mergesort for eight inputs.svg** *Source*: http://en.wikipedia.org/w/index.php?title=File:Batcher_Bitonic_Mergesort_for_eight_inputs.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Octotron

# License