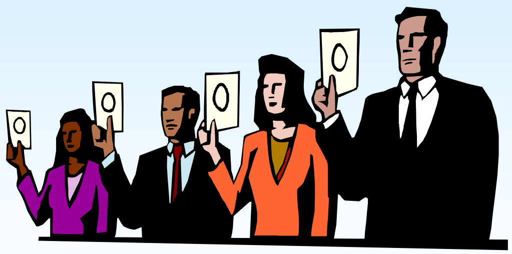
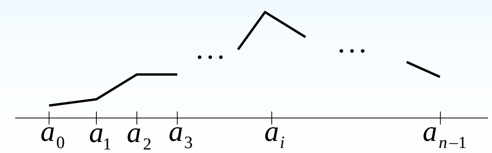
Bitonic Sort



Bitonic Sequences

- A *bitonic sequence* is a list of keys a_0, a_1, …, a_{n-1} such that
  1. For some i, the keys have the ordering $a_0 \le a_1 \ldots \le a_i \ge \ldots \ge a_{n-1}$, or
  2. The sequence can be shifted cyclically so that #1 holds

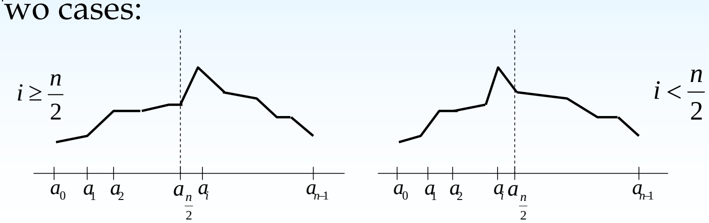


A Property of Bitonic Sequences

- Assume that a_0, a_1, …, a_{n-1} is bitonic and that n is even.
  - Let $B_1 = \min(a_0 , a_{n/2}), \min(a_1 , a_{(n/2)+1}), \ldots, \min(a_{(n/2)-1} , a_{n-1})$
  - Let $B_2 = \max(a_0 , a_{n/2}), \max(a_1 , a_{(n/2)+1}), \ldots, \max(a_{(n/2)-1} , a_{n-1})$
- Then B_1 and B_2 are bitonic sequences, and for all $x \in B_1$ and $y \in B_2$, $x \le y$.

Picture "Proof" of the Property

- $B_1 = \min(a_0 , a_{n/2}), \min(a_1 , a_{(n/2)+1}), \ldots, \min(a_{(n/2)-1} , a_{n-1})$
- $B_2 = \max(a_0 , a_{n/2}), \max(a_1 , a_{(n/2)+1}), \ldots, \max(a_{(n/2)-1} , a_{n-1})$
- Two cases:

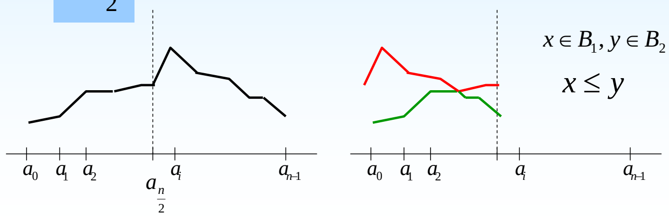


Picture "Proof" of the Property

$$B_1 = \min\{a_0, a_{\frac{n}{2}}\}, \min\{a_1, a_{\frac{n}{2}+1}\}, \cdots, \min\{a_{\frac{n}{2}-1}, a_{n-1}\}$$

$$B_2 = \max\{a_0, a_{\frac{n}{2}}\}, \max\{a_1, a_{\frac{n}{2}+1}\}, \cdots, \max\{a_{\frac{n}{2}-1}, a_{n-1}\}$$

$i \geq \frac{n}{2}$

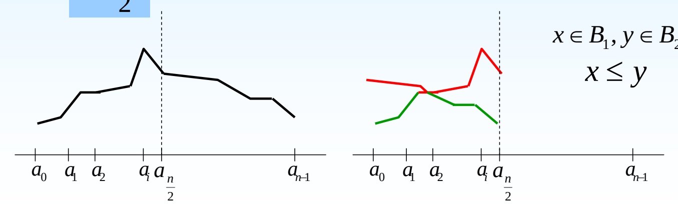


Picture "Proof" of the Property

$$B_1 = \min\{a_0, a_{\frac{n}{2}}\}, \min\{a_1, a_{\frac{n}{2}+1}\}, \cdots, \min\{a_{\frac{n}{2}-1}, a_{n-1}\}$$

$$B_2 = \max\{a_0, a_{\frac{n}{2}}\}, \max\{a_1, a_{\frac{n}{2}+1}\}, \cdots, \max\{a_{\frac{n}{2}-1}, a_{n-1}\}$$

$i < \frac{n}{2}$

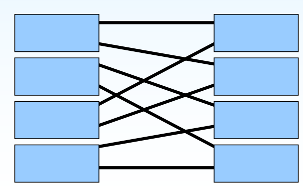


Bitonic Sort Algorithm

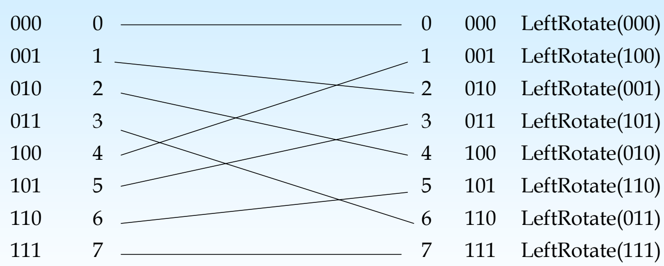
- The bitonic sort algorithm recursively calls two procedures:
  - BSORT(i, j, X) takes bitonic sequence $a_i, a_{i+1}, \ldots, a_j$ and produces a non-decreasing (X=+) or a non-increasing sorted sequence (X=-)
  - BITONIC(i, j) takes an unsorted sequence $a_i, a_{i+1}, \ldots, a_j$ and produces a bitonic sequence
- The main (sort) algorithm is then
  - BITONIC(0,n-1)
  - BSORT(0,n-1,+)

Bitonic Sort

- Bitonic sort is an example of a synchronous algorithm
  - Computation proceeds in stages where each stage is a (smaller or larger) shuffle-exchange network
  - Barrier synchronization at each stage

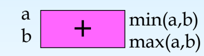


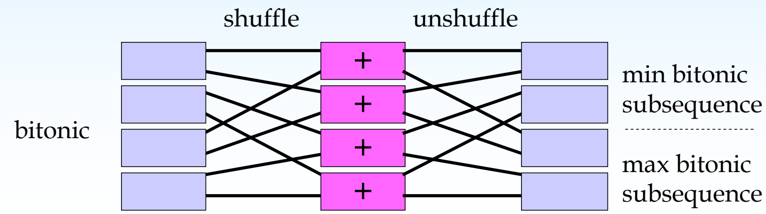
Perfect Shuffle



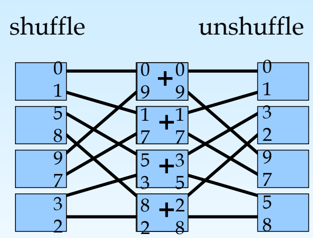
The Shuffle-Exchange

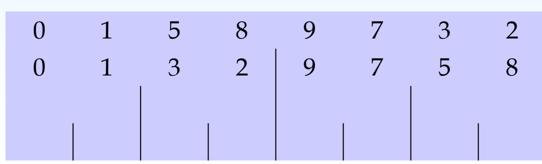
- Shuffle-exchange network routes the data correctly for comparison
- At each shuffle stage, we use a + switch to separate B_1 and B_2





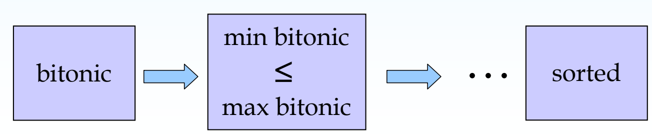
The Shuffle-Exchange





Back to Bitonic Sort

- Remember
  - BSORT(i, j, X) takes bitonic sequence a_i, a_{i+1}, …, a_j and produces a non-decreasing (X=+) or a non-increasing sorted sequence (X=-)
  - BITONIC(i, j) takes an unsorted sequence a_i, a_{i+1}, …, a_j and produces a bitonic sequence
- Let's look at BSORT first …



BSORT

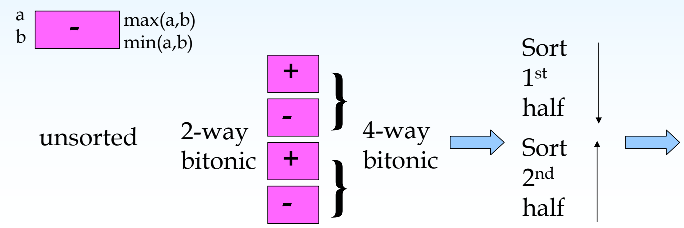
```
BSORT(i,j,X)
        If |j-i|<2 then
                return [min/max(i,i+1), max/min(i,i+1)]   ← Base case
        Else                                              ← Min/Max Bitonic
                Shuffle(i,j,X)
                Unshuffle(i,j)
                Pardo
                        BSORT (i,i+(j-i+1)/2 - 1,X)
                        BSORT (i+(j-i+1)/2,j,X)
```

0	1	5	8	9	7	3	2
0	1	3	2	9	7	5	8
0	1	3	2	5	7	9	8
0	1	2	3	5	7	8	9

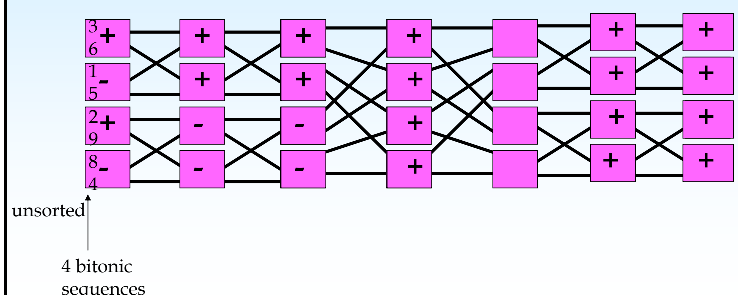
← Base case

BITONIC

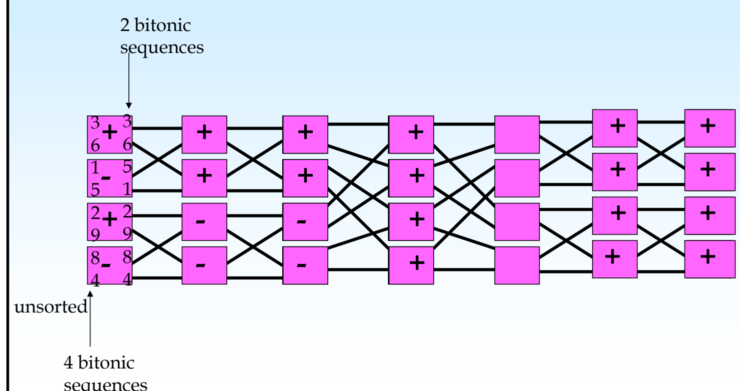
```
BITONIC(i,j)
  If |j-i|<2 then return [i, i+1]
  Else
    Pardo
      BITONIC(i, i+(j-i+1)/2 – 1); BSORT (i, i+(j-i+1)/2 - 1, +)
      BITONIC(i+(j-i+1)/2, j); BSORT (i+(j-i+1)/2, j, -)
```



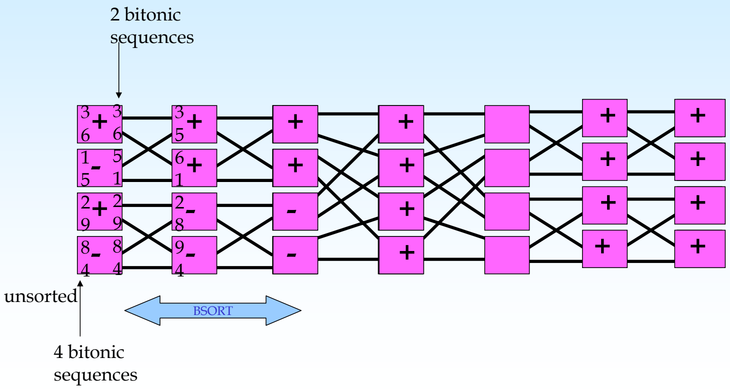
An Example



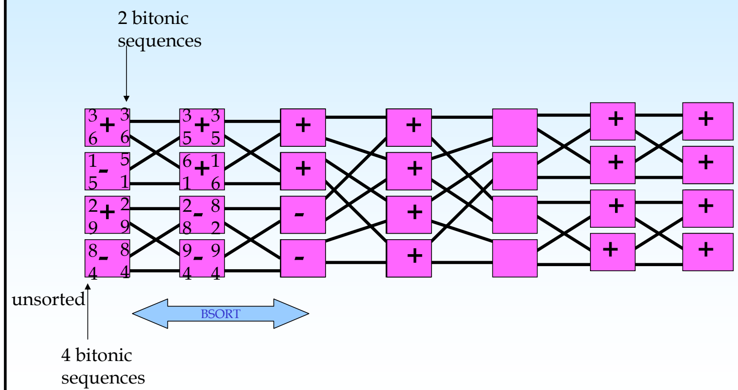
An Example



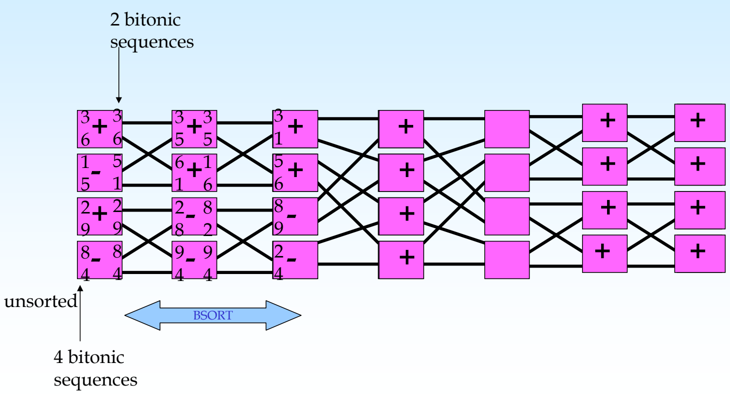
An Example



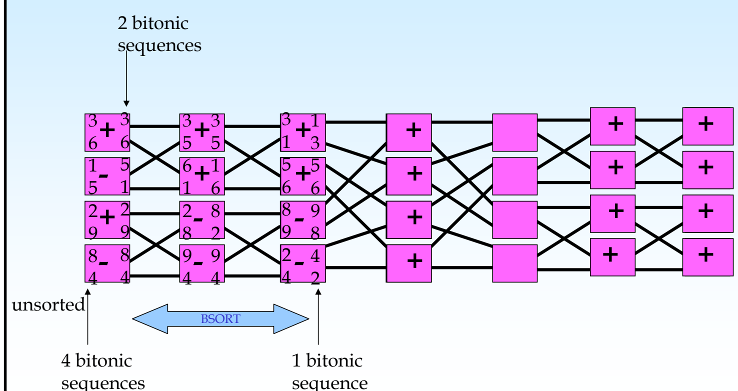
An Example



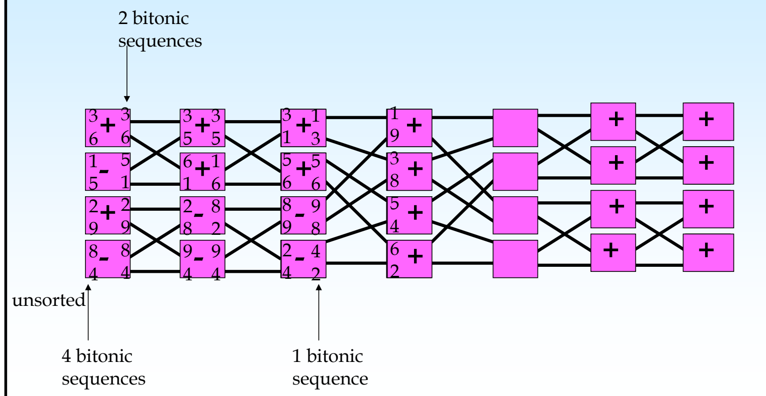
An Example



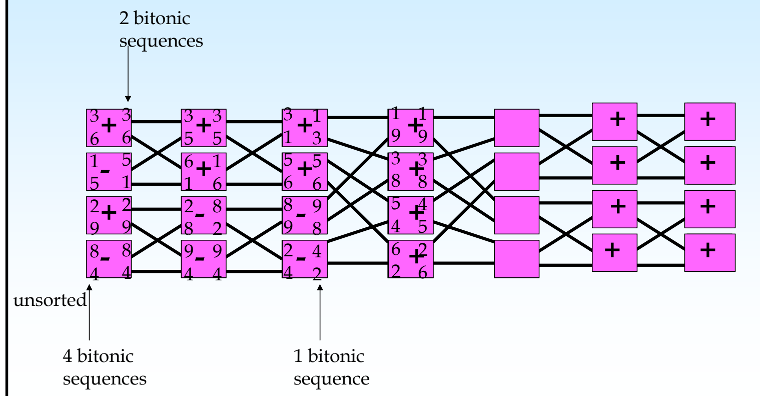
An Example



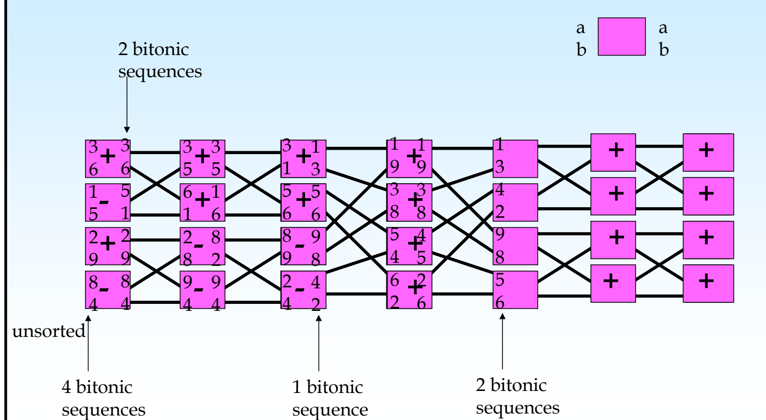
An Example



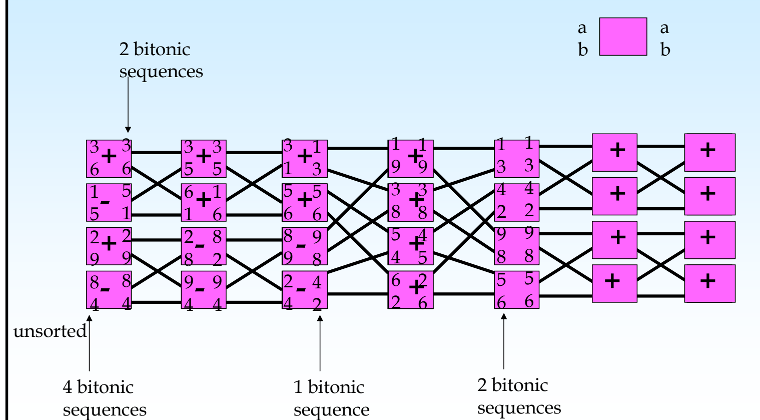
An Example



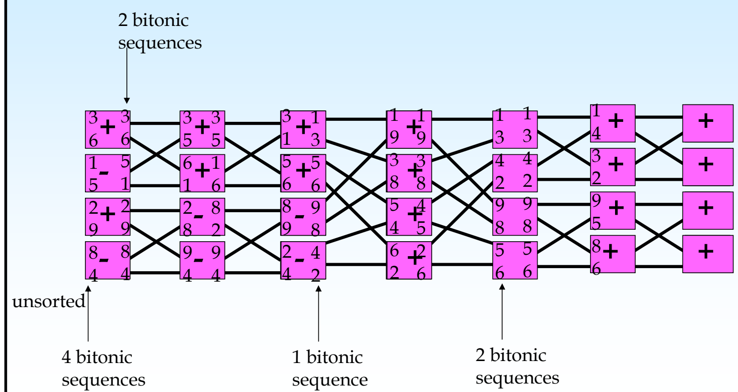
An Example



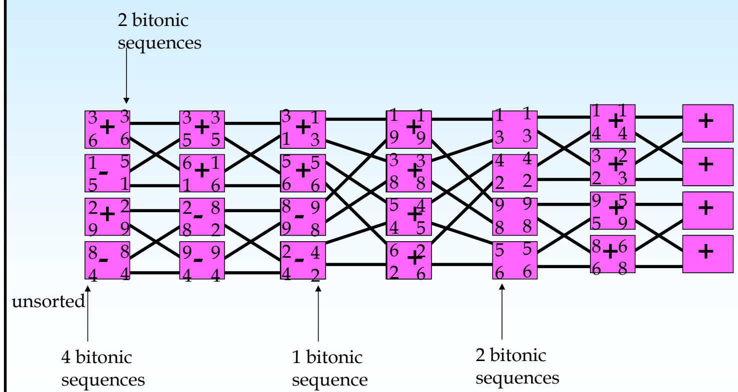
An Example



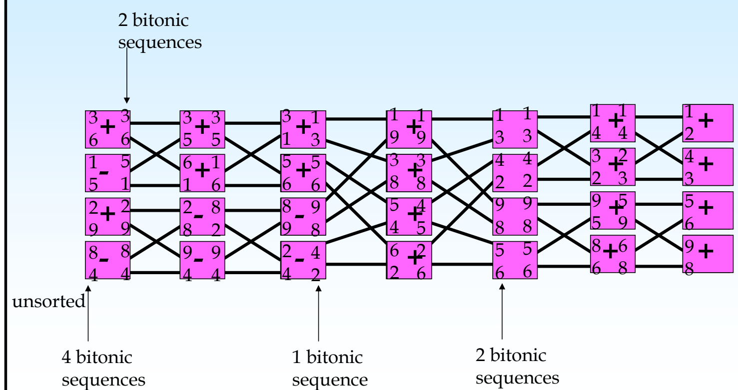
An Example



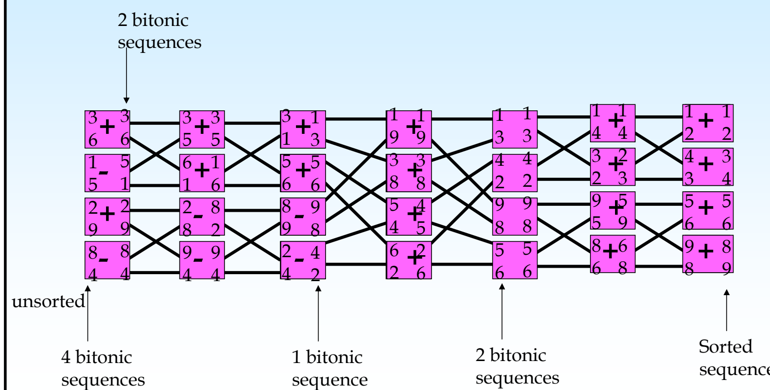
An Example



An Example



An Example



Time Complexities

$$T_{\mathrm{BSORT}}(n = 2^j) = 1 + T(2^{j-1})$$
$$= \ldots = 2(j-1) + T(2)$$
$$= O(j) = O(\log n)$$

$$T_{\mathrm{BITONIC}}(n = 2^j) = T(2^{j-1}) + 2(j-1) + 1$$
$$= O(j^2)$$
$$= O(\log^2 n)$$