COMP 322: Fundamentals of Parallel Programming

Lecture 28: Bitonic Sort

John Mellor-Crummey
Department of Computer Science, Rice University
johnmc@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Introduction

- **Why study sorting?**
  - **—one of the most common operations performed on computers**
- **Sorting algorithm attributes**
  - **—internal vs. external**
    - **internal: data fits in memory**
    - **external: uses tape or disk**
  - **—comparison-based or not**
    - **comparison sort**
      - **basic operation: compare elements and exchange as necessary**
      - $\Theta(n \log n)$ **comparisons to sort n numbers**
    - **non-comparison-based sort**
      - ***e.g.*** **radix sort based on the binary representation of data**
      - $\Theta(n)$ **operations to sort n numbers**
  - **—parallel vs. sequential**

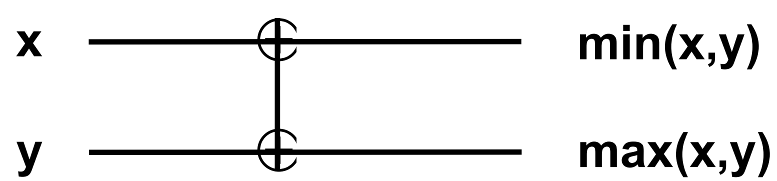
<u>Today's focus</u>
Bitonic sort: internal, comparison-based, parallel sort

Sorting Network

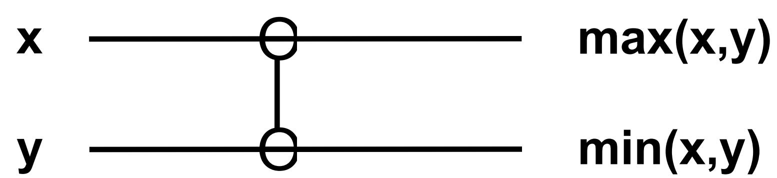
- **Network of comparators designed for sorting**
- **Comparator : two inputs x and y; two outputs x' and y'**

  —types

  - **increasing (denoted ⊕): x' = min(x,y) and y' = max(x,y)**



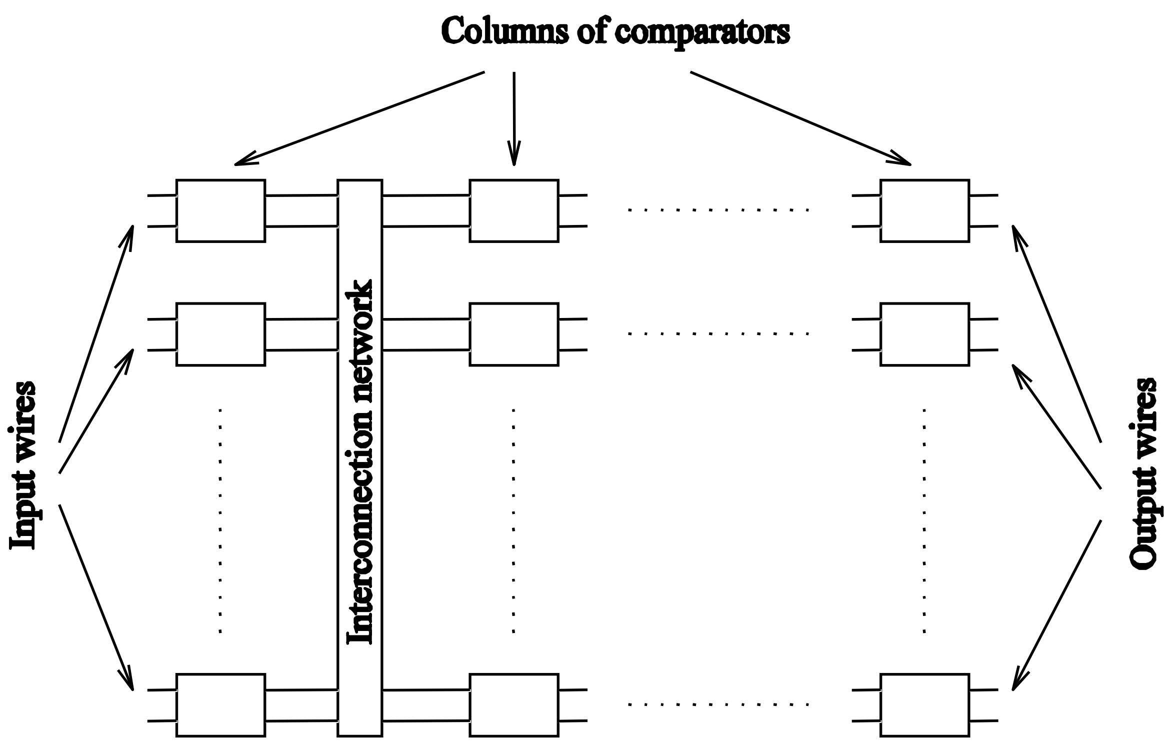
  - **decreasing (denoted ⊖) : x' = max(x,y) and y' = min(x,y)**



- **Sorting network speed is proportional to its depth**

Sorting Networks

- **Network structure: a series of columns**
- **Each column consists of a vector of comparators (in parallel)**
- **Sorting network organization:**

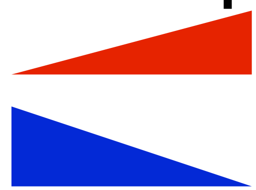


Example: Bitonic Sorting Network

- **Bitonic sequence**
  - **—two parts: increasing and decreasing**
    - **⟨1,2,4,7,6,0⟩: first increases and then decreases (or vice versa)**
  - **—cyclic rotation of a bitonic sequence is also considered bitonic**
    - **⟨8,9,2,1,0,4⟩: cyclic rotation of ⟨0,4,8,9,2,1⟩**
- **Bitonic sorting network**
  - **—sorts n elements in** $\Theta(\log^2 n)$ **time**
  - **—network kernel: rearranges a bitonic sequence into a sorted one**

Bitonic Split (Batcher, 1968)

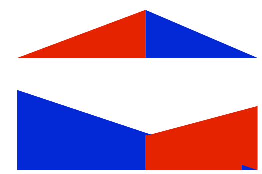
- Let $s = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ be a bitonic sequence such that
  - $a_0 \leq a_1 \leq \cdots \leq a_{n/2-1}$, and
  - $a_{n/2} \geq a_{n/2+1} \geq \cdots \geq a_{n-1}$



- Consider the following subsequences of s

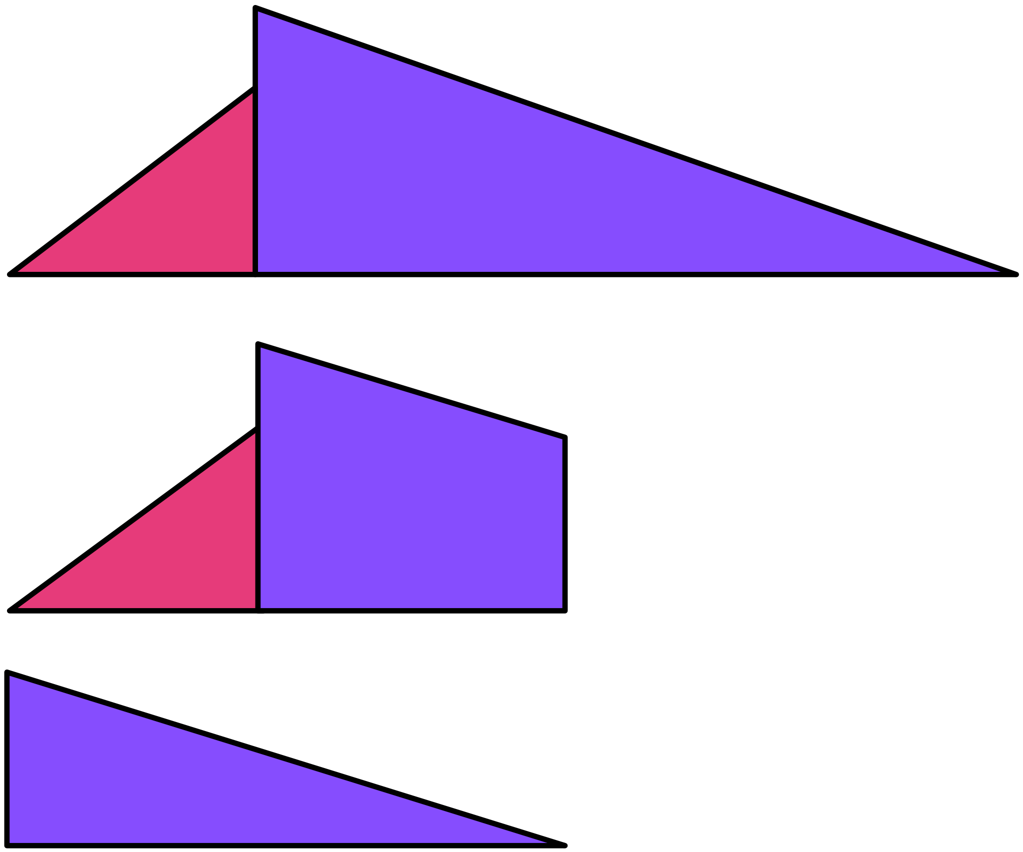
$$s_1 = \langle \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots, \min(a_{n/2-1}, a_{n-1}) \rangle$$

$$s_2 = \langle \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots, \max(a_{n/2-1}, a_{n-1}) \rangle$$



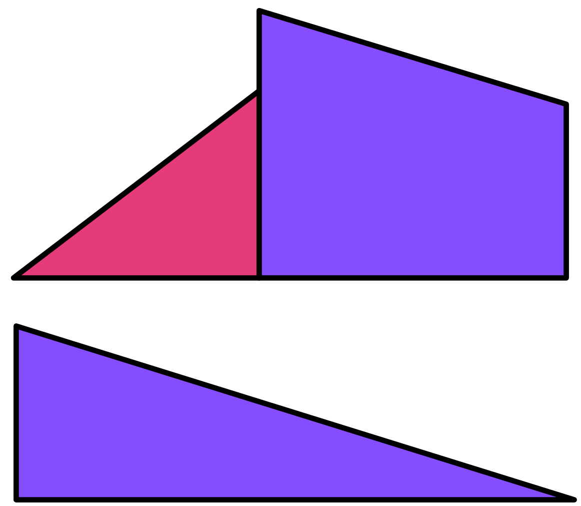
- Sequence properties
  - s_1 and s_2 are both bitonic
  - $\forall_x \forall_y \; x \in s_1, y \in s_2 , x < y$
- Apply recursively on s_1 and s_2 to produce a sorted sequence
- Works for any bitonic sequence, even if $|s_1| \neq |s_2|$

Splitting Bitonic Sequences - I



Sequence properties

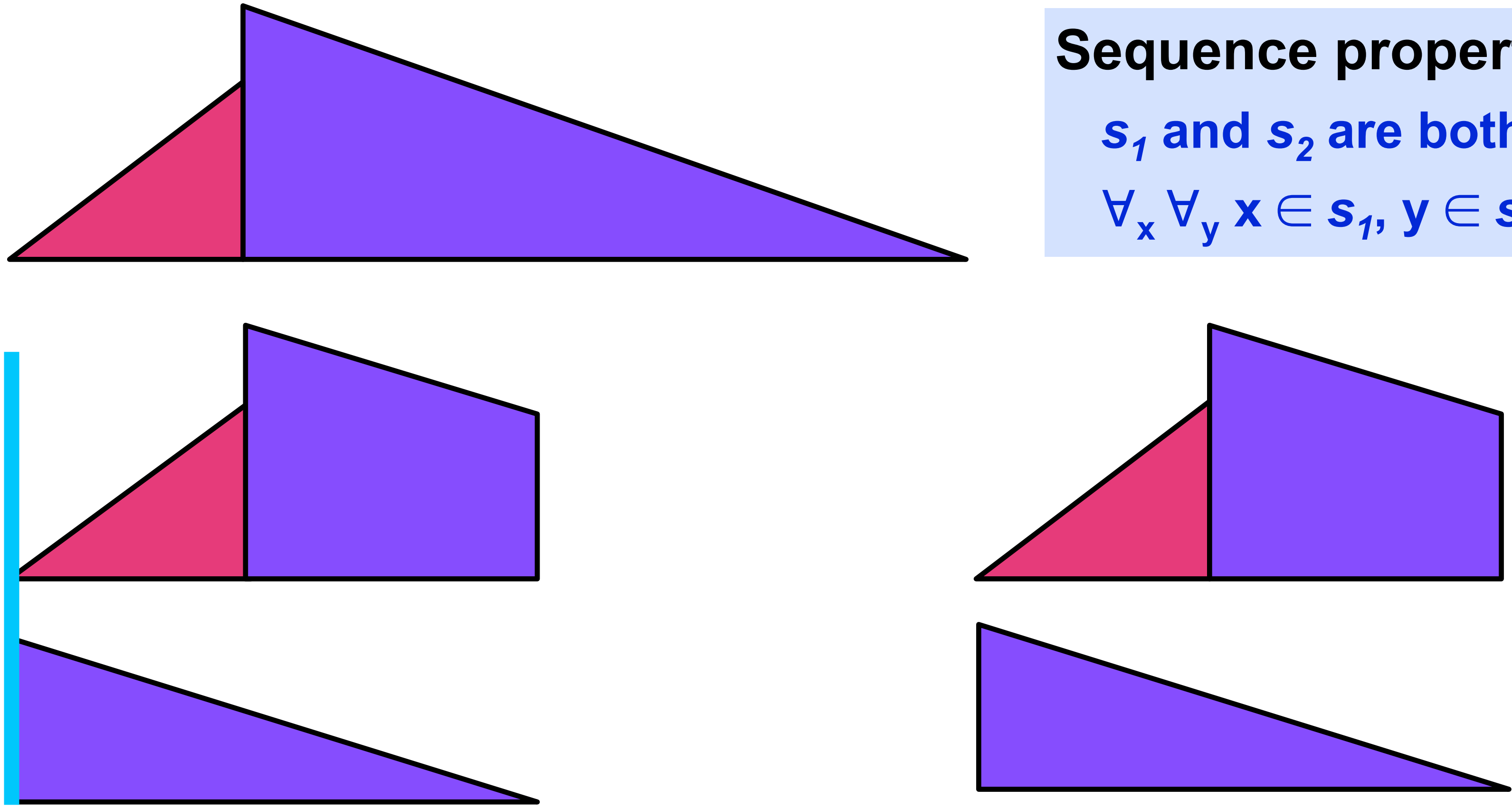
- s_1 and s_2 are both bitonic
- $\forall_x \forall_y\ x \in s_1, y \in s_2, x < y$



min

max

Splitting Bitonic Sequences - I



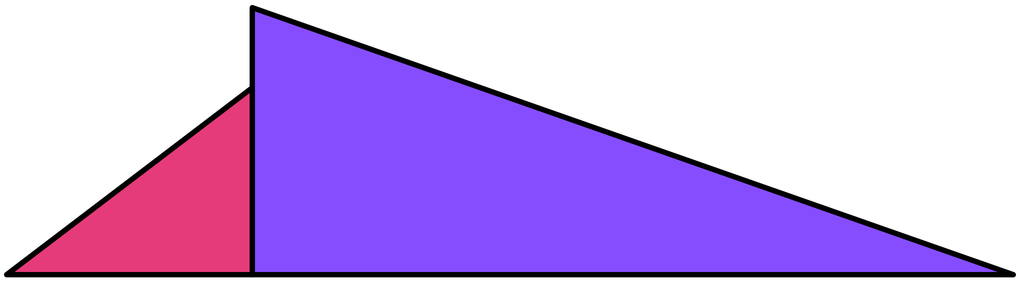
Sequence properties

s_1 and s_2 are both bitonic

$\forall_x \forall_y \; x \in s_1, y \in s_2, x < y$

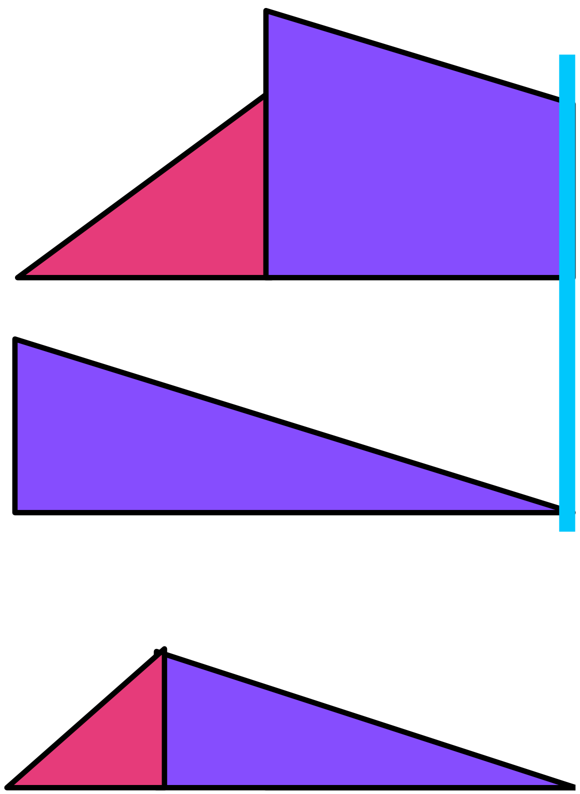
min max

Splitting Bitonic Sequences - I

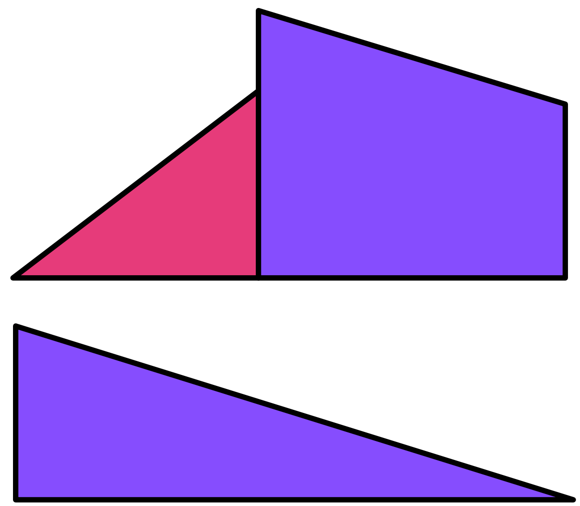


Sequence properties

- s_1 and s_2 are both bitonic
- $\forall_x \forall_y\ x \in s_1, y \in s_2, x < y$

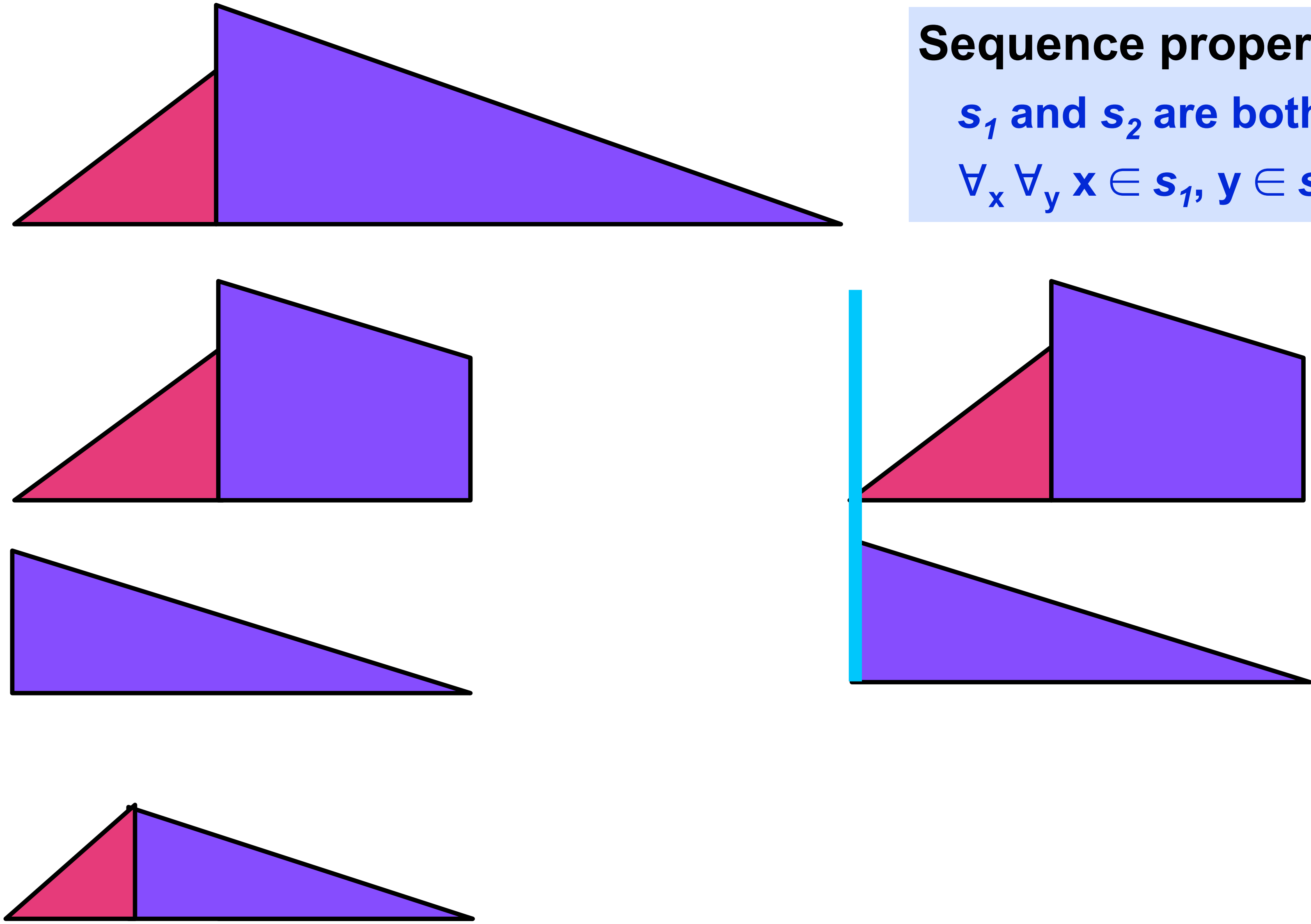


min



max

Splitting Bitonic Sequences - I



Sequence properties

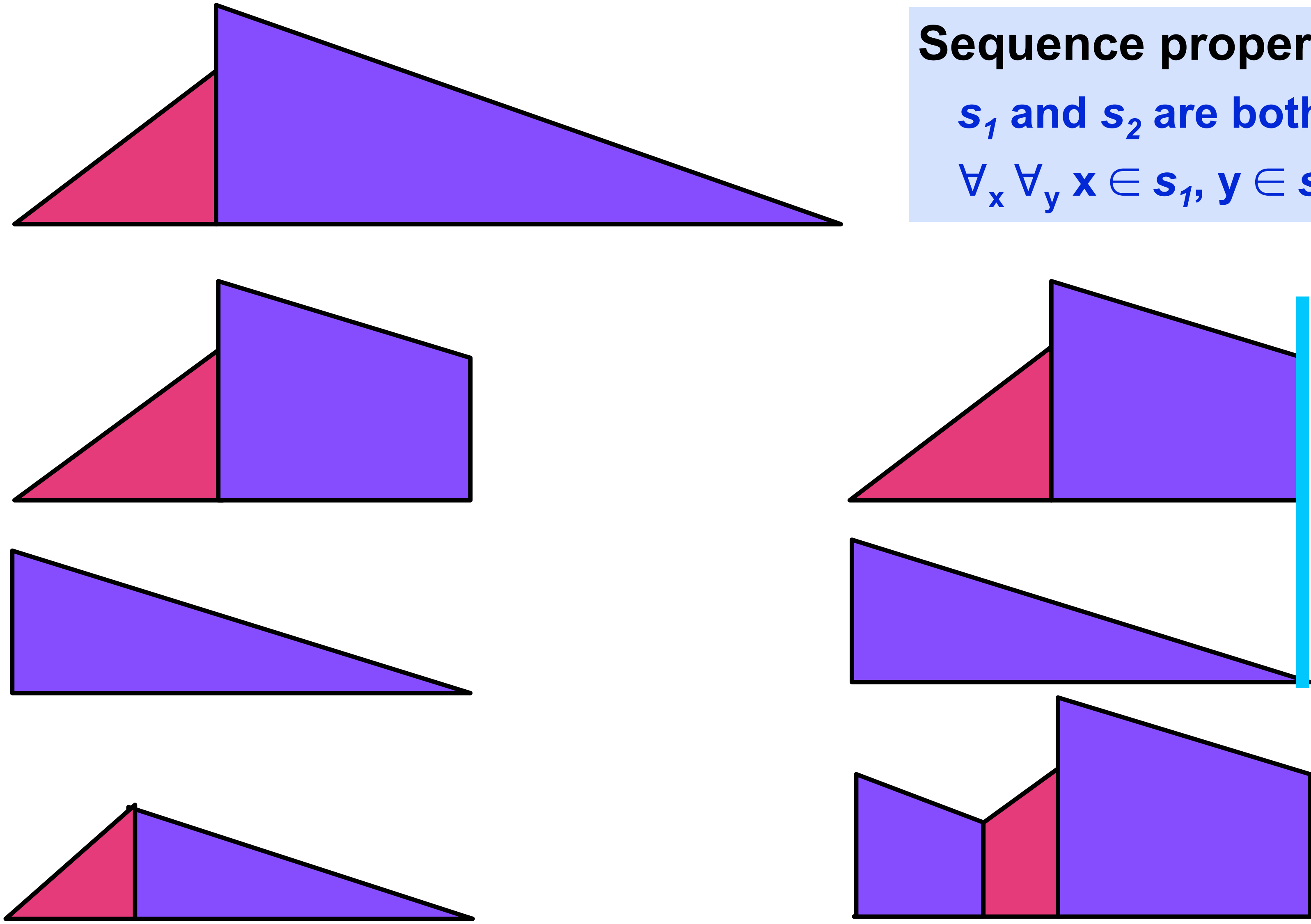
- s_1 and s_2 are both bitonic
- $\forall_x \forall_y\ x \in s_1, y \in s_2, x < y$

min max

Splitting Bitonic Sequences - I

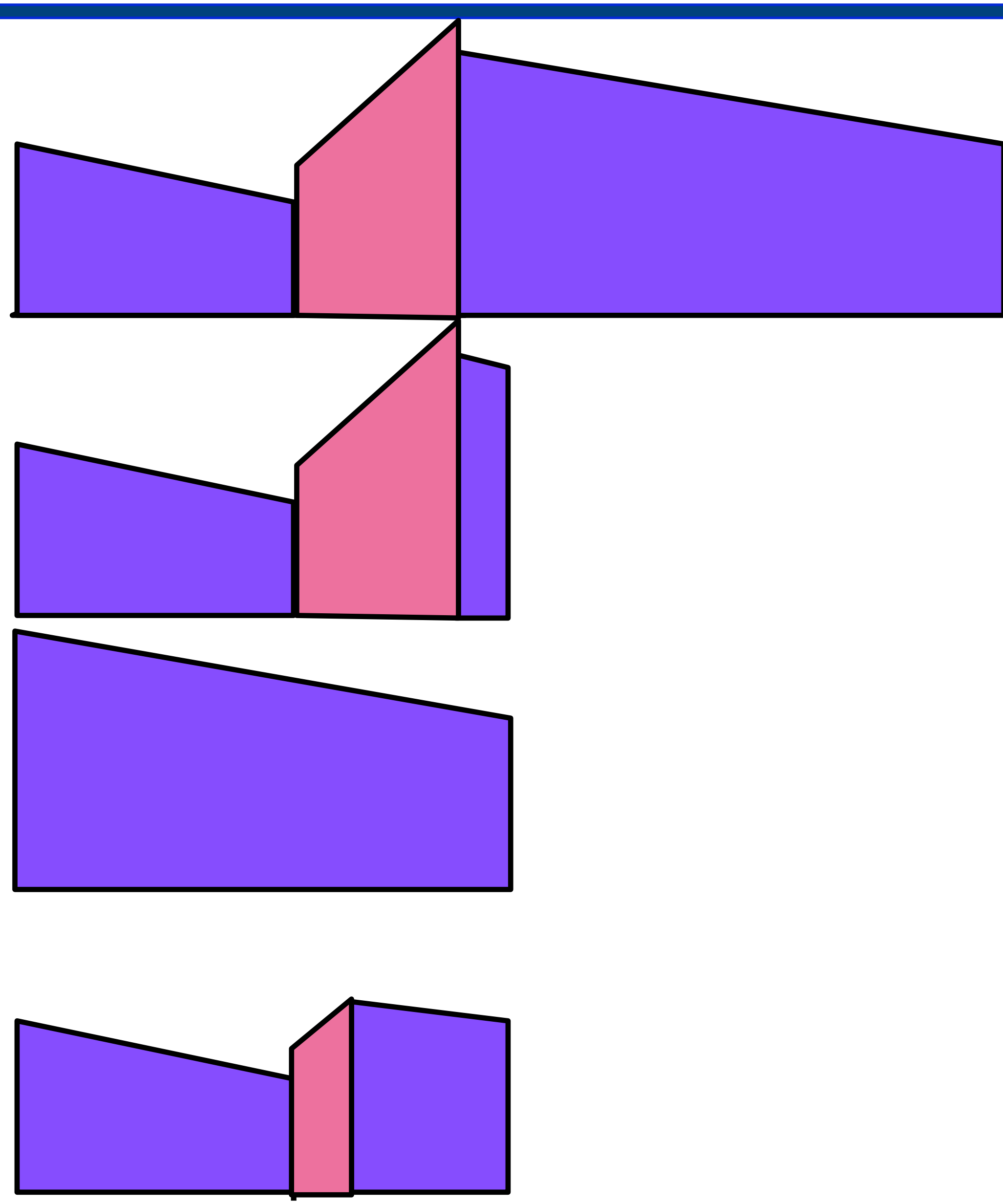
Sequence properties

- s_1 and s_2 are both bitonic
- $\forall_x \forall_y \; x \in s_1, y \in s_2 \, , \; x < y$



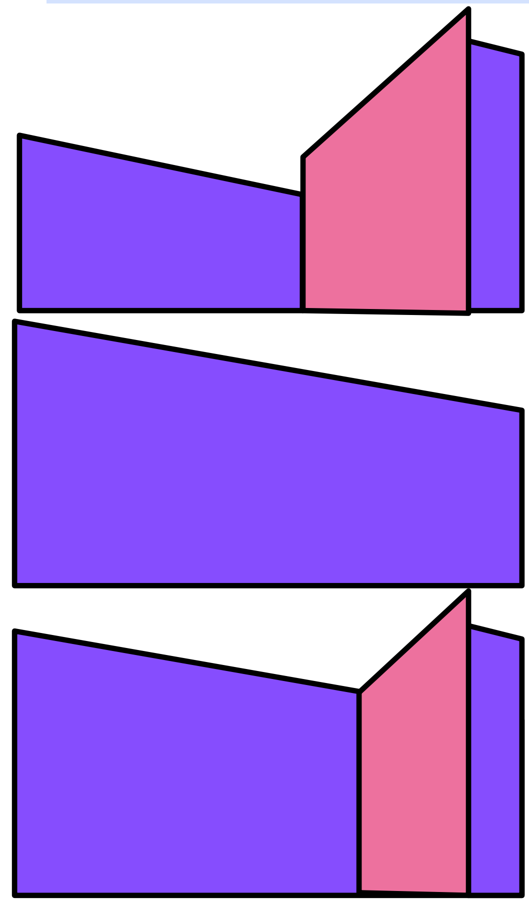
min max

Splitting Bitonic Sequences - II



Sequence properties

- s_1 and s_2 are both bitonic
- $\forall_x \forall_y\ x \in s_1,\ y \in s_2\ ,\ x < y$



min

max

Bitonic Merge

Sort a bitonic sequence through a series of bitonic splits

Example: use bitonic merge to sort 16-element bitonic sequence

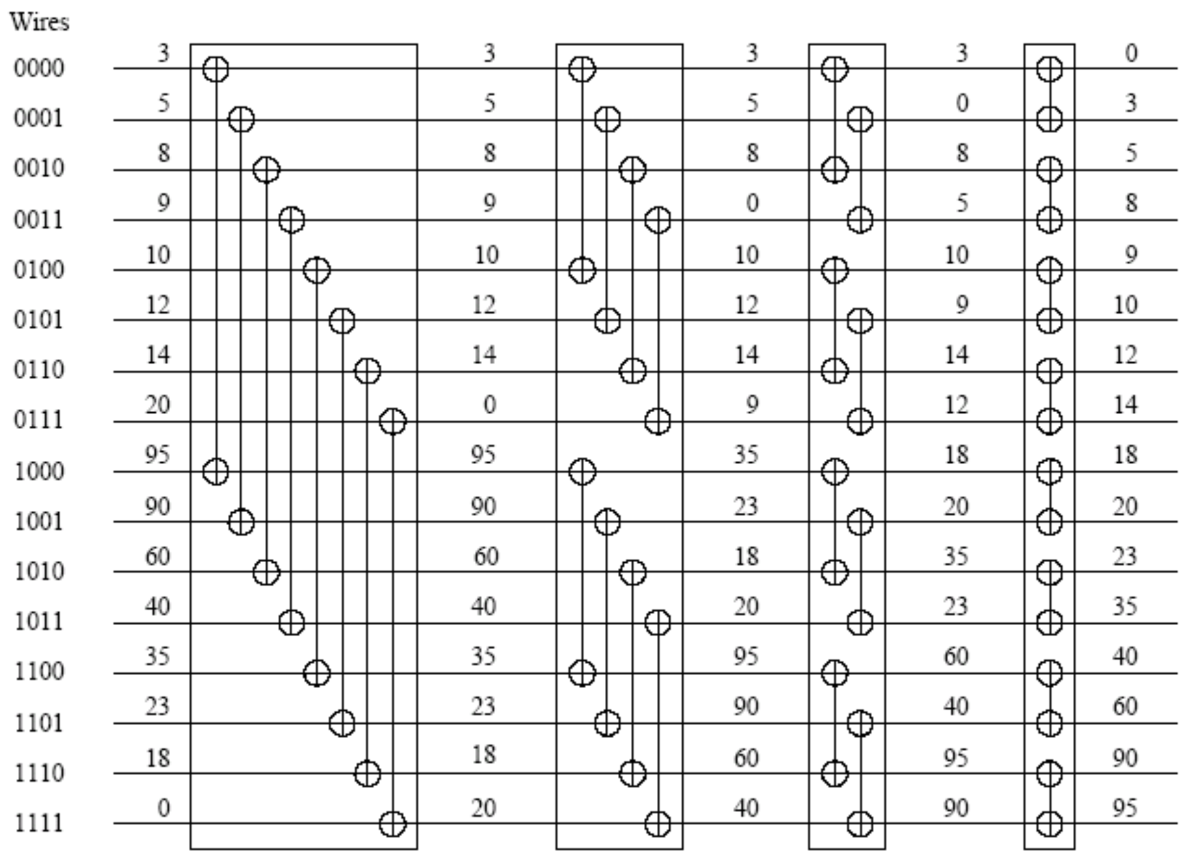
How: perform a series of $\log_2 16 = 4$ bitonic splits

Original sequence	3	5	8	9	10	12	14	20	95	90	60	40	35	23	18	0
1st Split	3	5	8	9	10	12	14	0	95	90	60	40	35	23	18	20
2nd Split	3	5	8	0	10	12	14	9	35	23	18	20	95	90	60	40
3rd Split	3	0	8	5	10	9	14	12	18	20	35	23	60	40	95	90
4th Split	0	3	5	8	9	10	12	14	18	20	23	35	40	60	90	95

Sorting via Bitonic Merging Network

- **Sorting network can implement bitonic merge algorithm**
  - —*bitonic merging network*
- **Network structure**
  - —$\log_2 n$ columns
  - —each column
    - $n/2$ comparators
    - performs one step of the bitonic merge
- **Bitonic merging network with n inputs: ⊕BM[n]**
  - —produces an increasing sequence
- **Replacing ⊕ comparators by ⊖ comparators: ⊖BM[n]**
  - —produces a decreasing sequence

Bitonic Merging Network, ⊕ BM[16]



- **Input: bitonic sequence**
  - — input wires are numbered 0,1,…, n - 1 (shown in binary)
- **Output: sequence in sorted order**
- **Each column of comparators is drawn separately**

Bitonic Sort

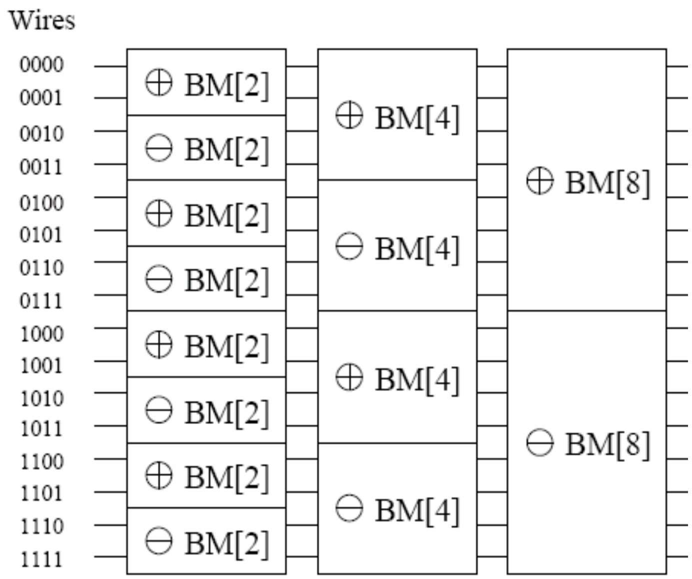
How do we sort an unsorted sequence using a bitonic merge?

Two steps

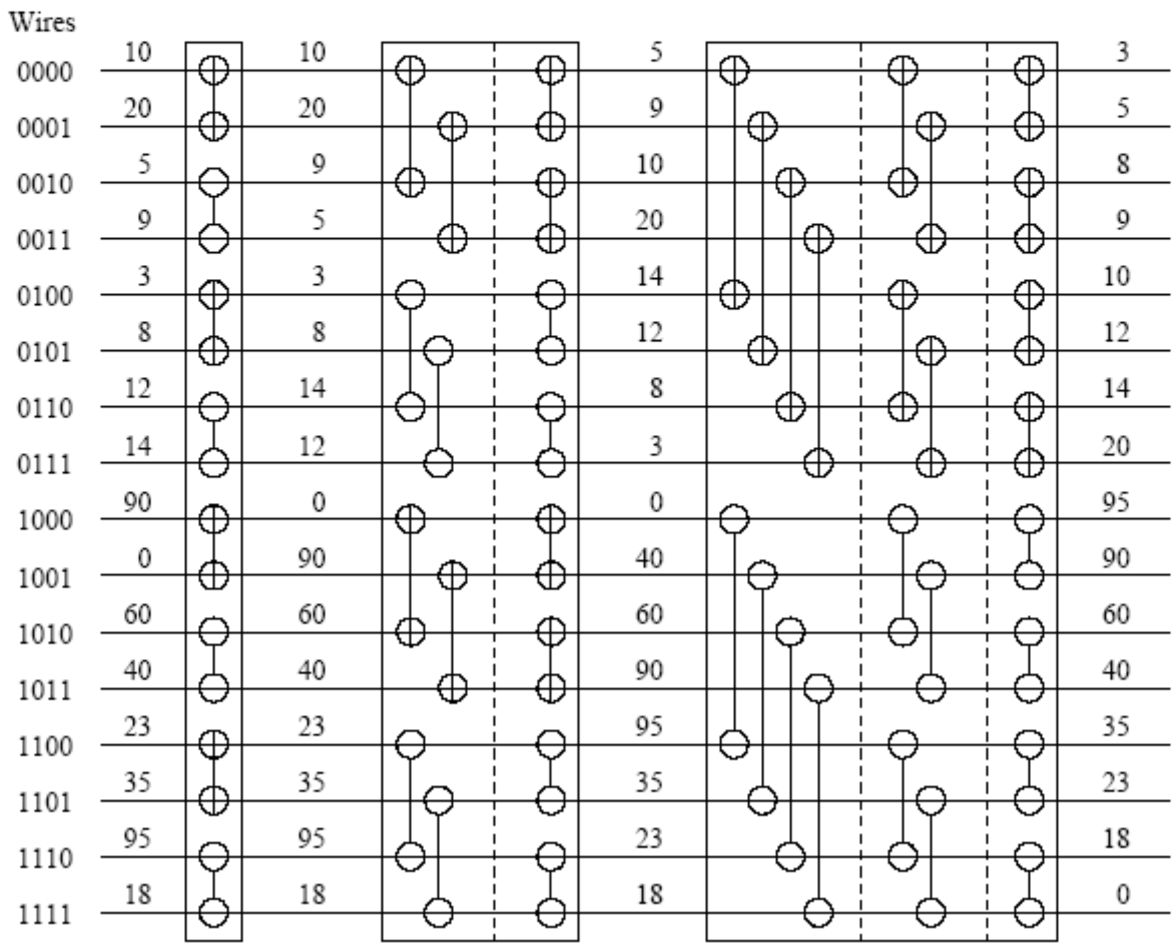
- **Build a bitonic sequence**
- **Sort it using a bitonic merging network**

Building a Bitonic Sequence

- **Build a single bitonic sequence from the given sequence**
  - **—any sequence of length 2 is a bitonic sequence.**
  - **—build bitonic sequence of length 4**
    - **sort first two elements using ⊕BM[2]**
    - **sort next two using ⊖BM[2]**
- **Repeatedly merge to generate larger bitonic sequences**
  - **—⊕BM[*k*] & ⊖BM[*k*]: bitonic merging networks of size *k***



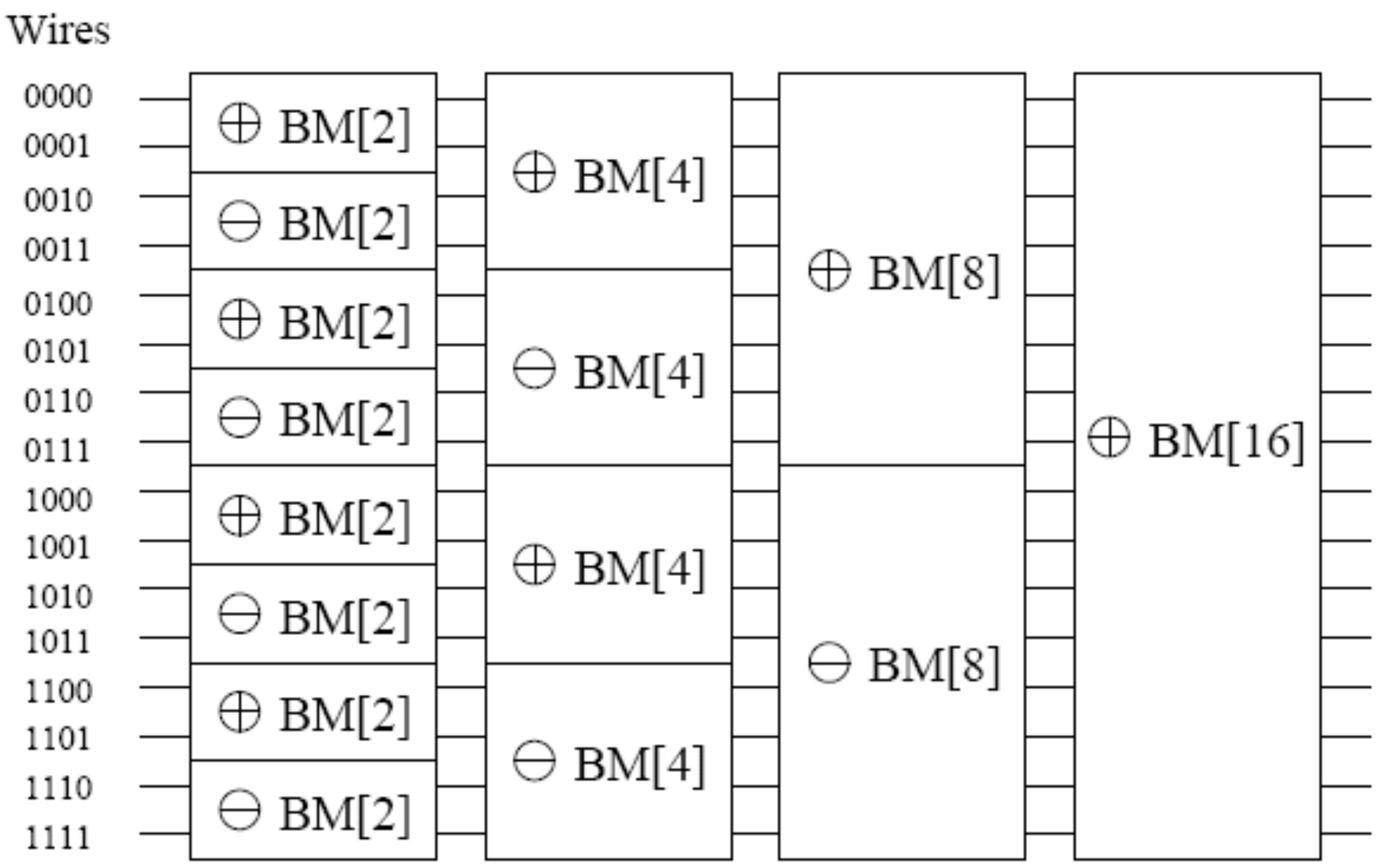
Building a Bitonic Sequence



Input: sequence of 16 unordered numbers

Output: a bitonic sequence of 16 numbers

Bitonic Sort, n = 16



- **First 3 stages create bitonic sequence input to stage 4**
- **Last stage (⊕BM[16]) yields sorted sequence**

Complexity of Bitonic Sorting Networks

- **Depth of the network is** $\Theta(\log^2 n)$
  - **$\log_2 n$ merge stages**
  - **j^{th} merge stage depth is $\log_2 2^j = j$**
  - **depth =** $\sum_{j=1}^{\log_2 n} \log_2 2^j = \sum_{i=1}^{\log_2 n} j = (\log_2 n + 1)(\log_2 n)/2 = \theta(\log^2 n)$
- **Each stage of the network contains *n*/2 comparators**
- **Complexity of serial implementation =** $\Theta(n \log^2 n)$

From Sorting Network to Pseudocode

Batcher's Bitonic Sort

- **bmerge(s): recursively sort a bitonic sequence as follows**
  1. compute s_1 and s_2 as shown earlier for ascending sort of s

     both will be bitonic by Batcher's Lemma

     note: for a descending sort, just swap min & max
  2. recursively call bmerge on s_1 and s_2
  3. return s = concat(bmerge(s_1), bmerge(s_2))
- **bsort(s): create a bitonic sequence then sort it**
  1. convert an arbitrary sequence s into a bitonic sequence
     - sort s[1 … n/2] in place in ascending order (recursive call to sort)
     - sort s[n/2+1 … n] in place in descending order (recursive call to sort)
  2. after step 1, the sequence will be bitonic; sort it using bmerge(s)

Bitonic Sort in HJ

```
void bmerge(final int[] A, final int low, final int high, boolean asc) {
  if ( high-low > 1 ) {
    final int mid = (low + high)/2 ;
    final int size = high – low + 1;
    forall (point[i]:[low:mid]) orderElementPair(A, i, i+size/2, asc);
    finish {
      async bmerge(A, low, mid, asc); async bmerge(A, mid+1, high, asc);
    } // finish
  } // if
} // bmerge

void bsort (final int[] A, final int low, final int high, boolean asc) {
  if ( high-low > 1 ) {
    finish {
      final int mid = (low + high)/2 ;
      async bsort(A, low, mid, asc); async bsort(A, mid+1, high, !asc);
    } // finish
    bmerge(A, low, high, asc); // asc = true is for ascending order
  } // if
} // sort
```

Batcher's Bitonic Sort in NESL

```
function bitonic_merge(a) =
if (#a == 1) then a
else
   let
      halves = bottop(a)
      mins = {min(x, y) : x in halves[0]; y in halves[1]};
      maxs = {max(x, y) : x in halves[0]; y in halves[1]};
   in flatten({bitonic_merge(x) : x in [mins,maxs]});

function bitonic_sort(a) =
if (#a == 1) then a
else
   let b = {bitonic_sort(x) : x in bottop(a)};
   in bitonic_merge(b[0]++reverse(b[1]));
```

Run this code at http://www.cs.rice.edu/~johnmc/nesl.html

References

- Adapted from slides “Sorting” by Ananth Grama
- Based on Chapter 9 of “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003
- “Programming Parallel Algorithms.” Guy Blelloch. Communications of the ACM, volume 39, number 3, March 1996.
- http://www.cs.cmu.edu/~scandal/nesl/algorithms.html#sort
- “Highly Scalable Parallel Sorting.” Edgar Solomonik and Laxmikant V. Kale. Proc. of the IEEE Intl. Parallel and Distributed Processing Symp., April, 2010, Atlanta, GA.