Concurrency: Sorting Networks

Jeffery von Ronne

Department of Computer Science
Carnegie Mellon University

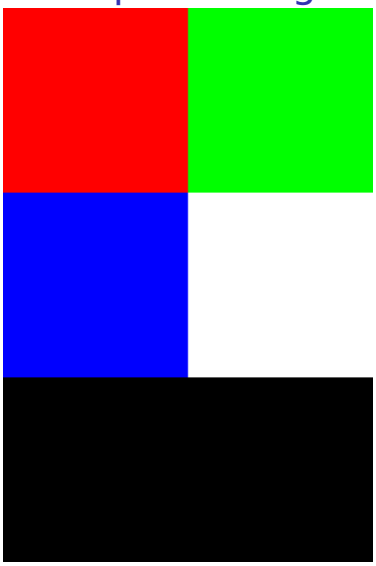
April 2, 2012

Outline

Review: Bitmap Images

- screen consists of individual pixels
  - pixel = picture elements
- arranged into rows and columns
  - projector 1024x768
  - 720p = 1280x720
  - 1080p = 1920x1080;

2 × 3 pixel image

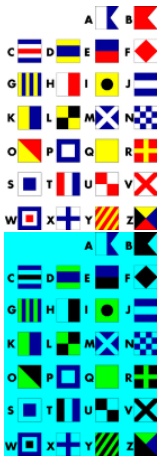


Bitmap as a 3-D Ruby Arrays

```
bitmap = [[[255,0,0], [0,255,0]],
          [[0,0,255], [255,255,255]],
          [[0,0,0],   [0,0,0]]]
```


Concurrency Example: Remove Red Operation

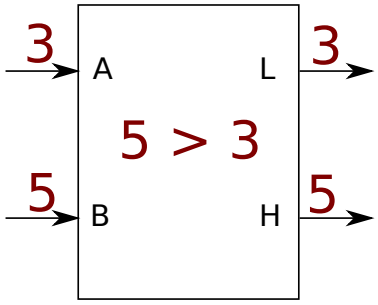
```
def remove_red(image)
  num_rows = image.length
  num_columns = image[0].length
  for row in 0..num_rows-1 do
    for column in 0..num_columns-1 do
      green = image[row][column][1]
      blue = image[row][column][2]
      image[row][column] = [0, green, blue]
    end
  end
  return nil
end
```



- What order are the pixels processed? Does this matter?
- do multiple pixels concurrently
- Graphical Processing Units (GPU)

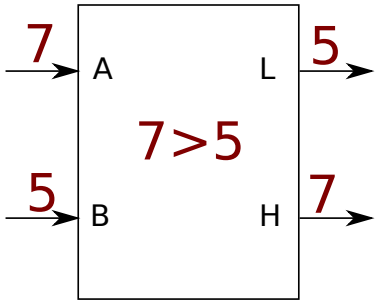
Single Element Comparison

- two inputs at left: "A" and "B"
- two outputs at right: "L" (low) and "H" (high)

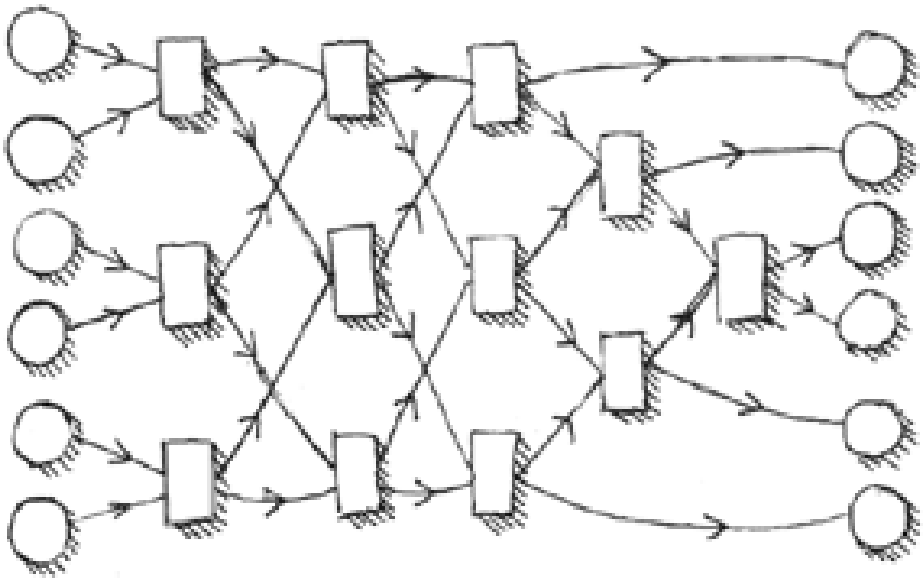


Single Element Comparison

- two inputs at left: "A" and "B"
- two outputs at right: "L" (low) and "H" (high)

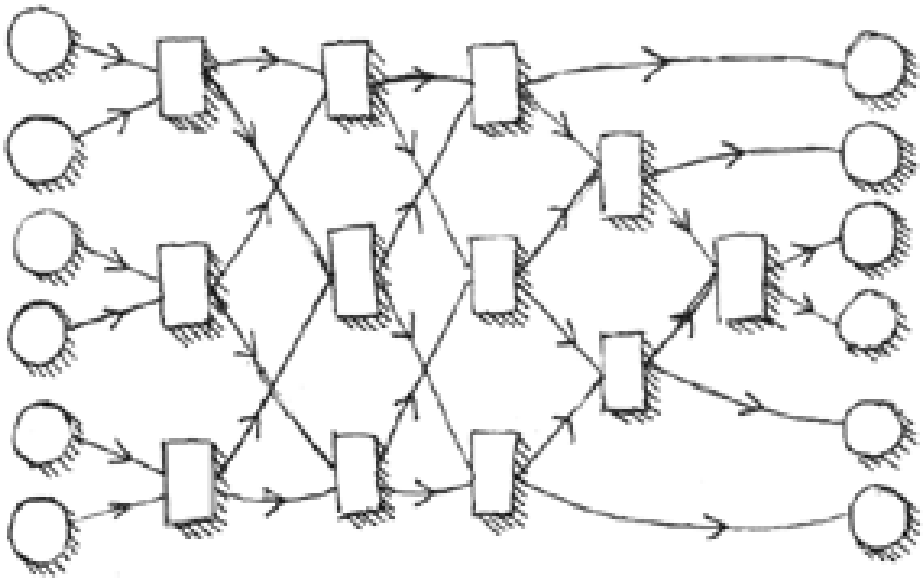


Activity: Sorting Network Simulation



Input: `[5, 1, 6, 3, 4, 2]`

Activity: Sorting Network Simulation



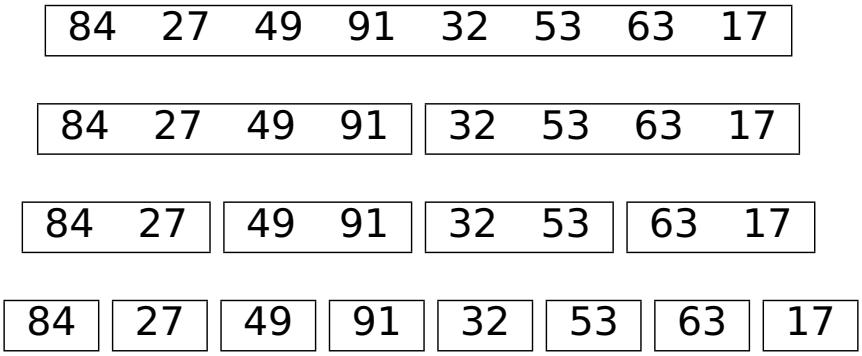
How many steps does this take . . . sequentially? concurrently?

Review: Merge Sort

Recursive Procedure

1. Recursively, sort the left half
2. Recursively, sort the right half
3. merge the two sorted half-list into sorted list

Example



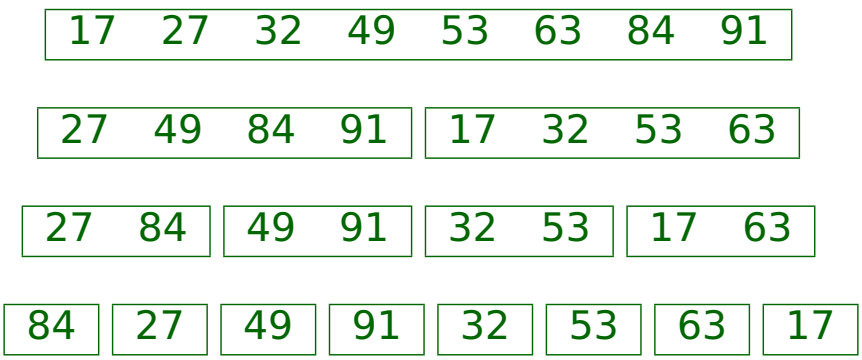
Sorting Network for Odd-Even Merge | Review: Merge Sort

Review: Merge Sort

Recursive Procedure

1. Recursively, sort the left half
2. Recursively, sort the right half
3. merge the two sorted half-list into sorted list

Example



Jeffery von Ronne (CMU) | 15-110 Unit 10B | April 2, 2012 | 10 / 18

Review: Merge Operation for Merge Sort

Merge Operation

- Takes two sorted lists (a and b)
- Returns one sorted list
- Containing elements of a and b

Can we do this concurrently? How?

Sequential Implementation

```
def merge(a,b)
  i, j = 0, 0
  c = []
  while i < a.length and j < b.length
    if a[i] <= b[j] then
      c << a[i]
      i = i + 1
    else
      c << b[j]
      j = j + 1
    end
  end
  return c + a[i..-1] + b[j..-1]
end
```

```
>> merge([27,49,84,91], [17,32,53,63])
=> [17, 27, 32, 49, 53, 63, 84, 91]
```


An Observation

Merge with Odd and Even Elements Marked

```
merge([27, 49, 84, 91, 92, 93],
      [17, 32, 53, 63, 95, 98])

=> [17, 27, 32, 49, 53, 63, 84, 91, 92, 93, 95, 98]
```

- elements initially at even indices
- elements initially at odd indices

- Do you see a pattern?
- How many even/odd elements are in `result[0..i]`?
- In `result[0..i]`:
  - always, at least as many even as odd
  - always, at most two more even than odd
  - when `i` is even, there is exactly one more even than odd

A Strategy for Merging

Procedure for Merging a and b

Parameters: two sorted lists a and b Result: one sorted list c

- Split a into even_a and odd_a
- Split b into even_b and odd_b
- Recursively, merge even_a and even_b into even_c
- Recursively, merge odd_a and odd_b into odd_c
- interleave even_c and odd_c to get an almost-sorted c
- swap neighbors, as necessary, to completely sort c

a = 27 49 84 91 b = 17 32 53 63

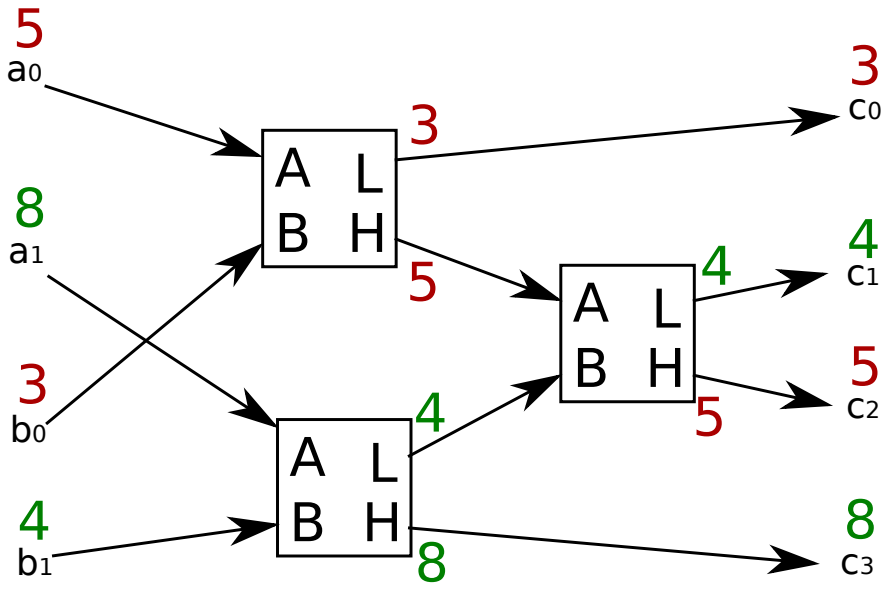
even_a = 27 84 odd_a = 49 91 even_b = 17 53 odd_b = 32 63

even_c = 17 27 53 84 odd_c = 32 49 63 91

c = 17 $\underline{32\ 27}$ $\underline{49\ 53}$ $\underline{63\ 84}$ 91

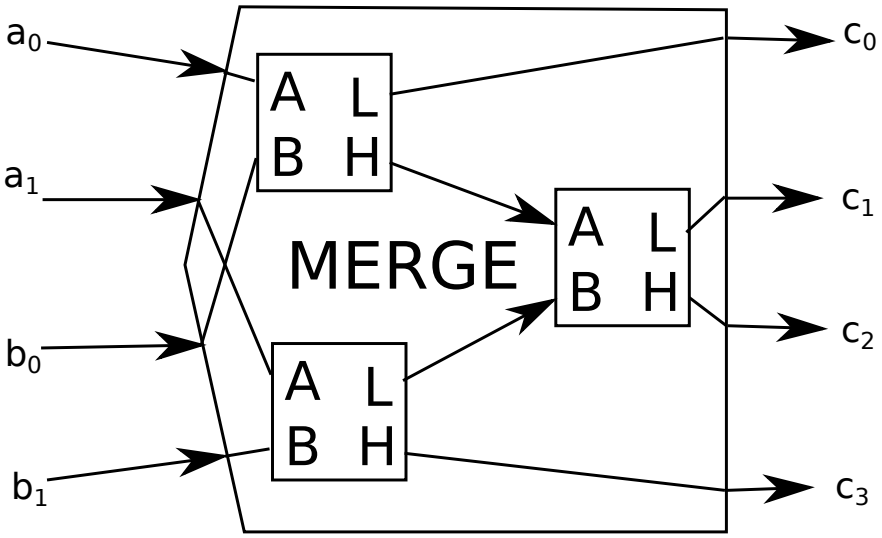
c = 17 27 32 49 53 63 84 91

2 × 2 Merge

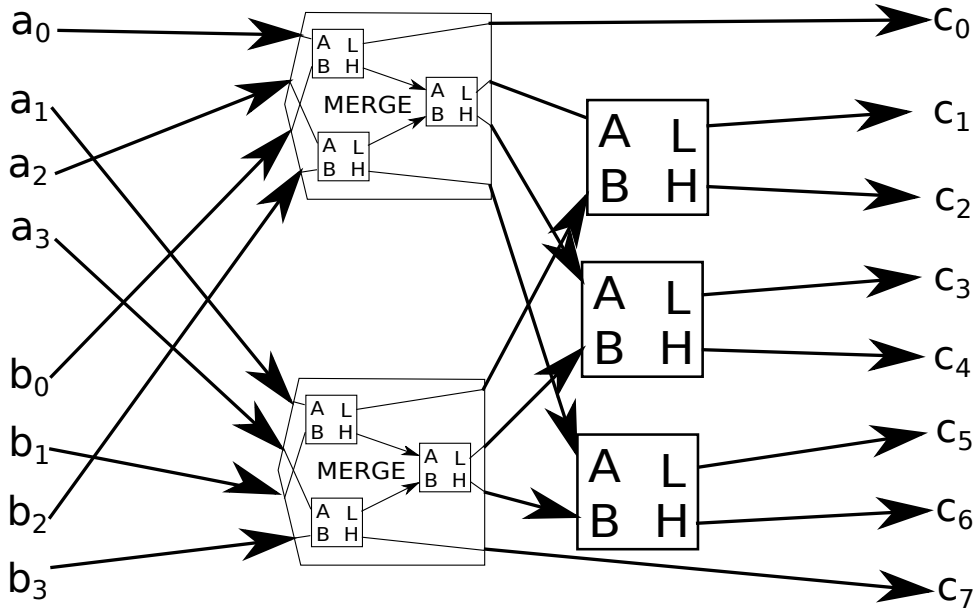


merge([5,8],[3,4])

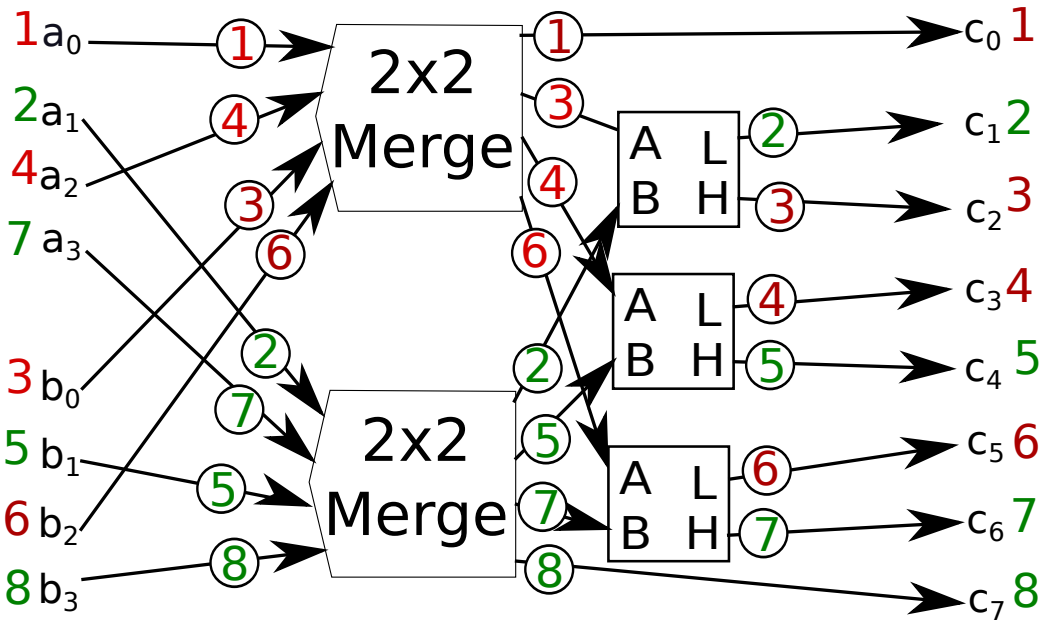
2 × 2 Merge



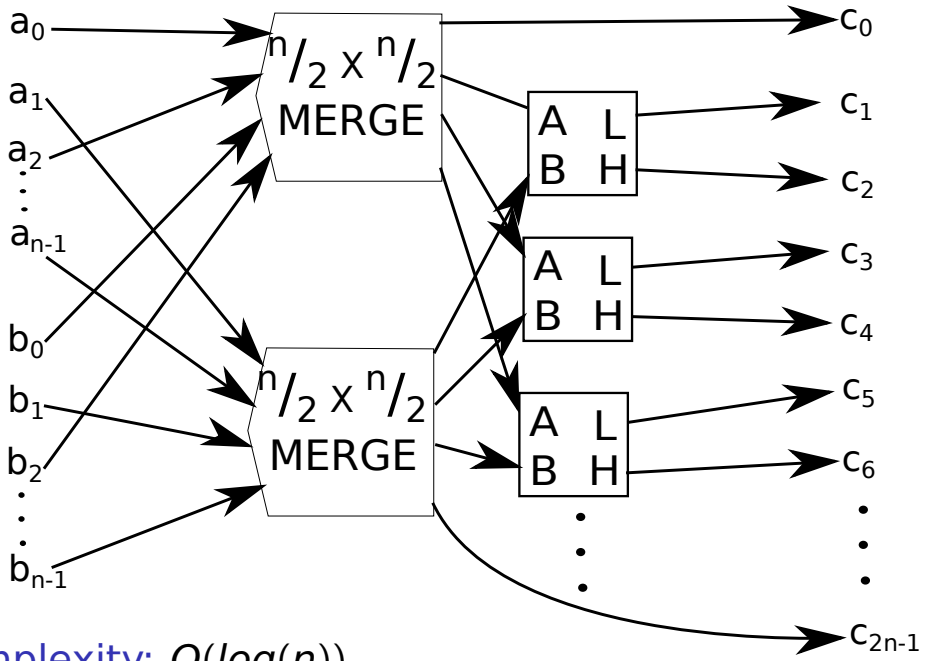
4 × 4 Odd-Even Merge



4 × 4 Odd-Even Merge



$n \times n$ Odd-Even Merge



Time Complexity: $O(log(n))$

Summary

You've seen:

- tasks that can be handled concurrently:
  - image manipulation
  - sorting
- sorting networks: comparison elements wired together
  - sorting operations (input: 1 unsorted list)
  - merging operations (input: 2 sorted lists)
- odd-even merge
  - divide and conquer (odd vs. even)
  - recursive construction
- time measured in comparisons between input and output
  - reduced through concurrency.