QuickSort

$\textsc{QuickSort}(array\ A, int\ p, int\ r)$

if $(p < r)$
  then $q \leftarrow \textsc{Partition}(A, p, r)$
    $\textsc{QuickSort}(A, p, q-1)$
    $\textsc{QuickSort}(A, q+1, r)$

To sort array call $\textsc{QuickSort}(A, 1, length[A])$.

$\textsc{Partition}(array\ A, int\ p, int\ r)$

$x \leftarrow A[r]$ ▷ Choose **pivot**
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** $r - 1$
  do if $(A[j] \leq x)$
    then $i \leftarrow i + 1$
      exchange $A[i] \leftrightarrow A[j]$
exchange $A[i+1] \leftrightarrow A[r]$
return $i + 1$

Page 146, CLRS

Analysis of QuickSort

- Average case
  - $T(n) <= 2T(n/2) + O(n)$
  - $T(n) = O(n \log n)$
- Worst case
  - $T(n) = T(n-1) + O(n)$
  - $T(n) = O(n^2)$
- **"IN PLACE"** sorting algorithm
  - Which sorting algorithm is not an "IN PLACE" sorting algorithm?

Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

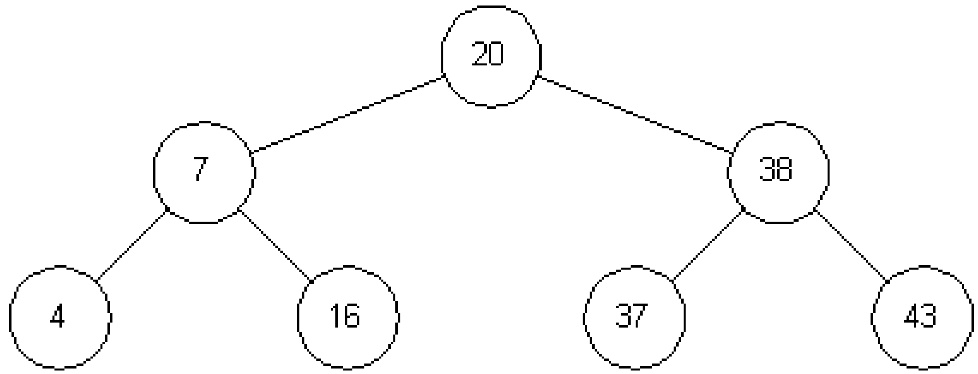
Sorting Algorithms

- SelectionSort
- InsertionSort
- BubbleSort
- ShakerSort
- QuickSort
- MergeSort
- HeapSort
- Bucket & Radix Sort
- Counting Sort

HeapSort

- First convert array into a heap (BUILD-MAX-HEAP, p133)
- Then convert heap into sorted array (HEAPSORT, p136)

Storing binary trees as arrays



20	7	38	4	16	37	43

Heaps (Max-Heap)

43	16	38	4	7	37	20

43	16	38	4	7	37	20	2	3	6	1	30

HEAP represents a binary tree stored as an array such that:

- Tree is filled on all levels except last
- Last level is filled from left to right
- Left & right child of i are in locations $2i$ and $2i+1$
- HEAP PROPERTY:

  Parent value is at least as large as child's value

HeapSort: Part 1

MAX-HEAPIFY(*array A*, *int i*)

▷ Assume subtree rooted at i is not a heap;
▷ but subtrees rooted at children of i are heaps

$l \leftarrow$ LEFT$[i]$
$r \leftarrow$ RIGHT$[i]$
if $((l \leq \textit{heap-size}[A])$ *and* $(A[l] > A[i]))$
    then $\textit{largest} \leftarrow l$
    else $\textit{largest} \leftarrow i$
if $((r \leq \textit{heap-size}[A])$ *and* $(A[r] > A[\textit{largest}]))$
    then $\textit{largest} \leftarrow r$
if $(\textit{largest} \neq i)$
    then exchange $A[i] \leftrightarrow A[\textit{largest}]$
        MAX-HEAPIFY$(A, \textit{largest})$

O(height of node in location i) = O(log(size of subtree))

p130

HeapSort: Part 2

BUILD-MAX-HEAP(*array A*)

```
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1
        do MAX-HEAPIFY(A, i)
```

HeapSort: Part 2

```
BUILD-MAX-HEAP(array A)
  heap-size[A] ← length[A]
  for i ← ⌊length[A]/2⌋ downto 1
       do MAX-HEAPIFY(A, i)
```

```
HEAPSORT(array A)
  BUILD-MAX-HEAP(A)
  for i ← length[A] downto 2
       do exchange A[1] ↔ A[i]
          heap-size[A] ← heap-size[A] − 1
          MAX-HEAPIFY(A, 1)
```

O(log n)

Total: O(nlog n)

Build-Max-Heap Analysis

For the HeapSort analysis, we need to compute:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}$$

We know from the formula for geometric series that

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Differentiating both sides, we get

$$\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}$$

Multiplying both sides by x we get

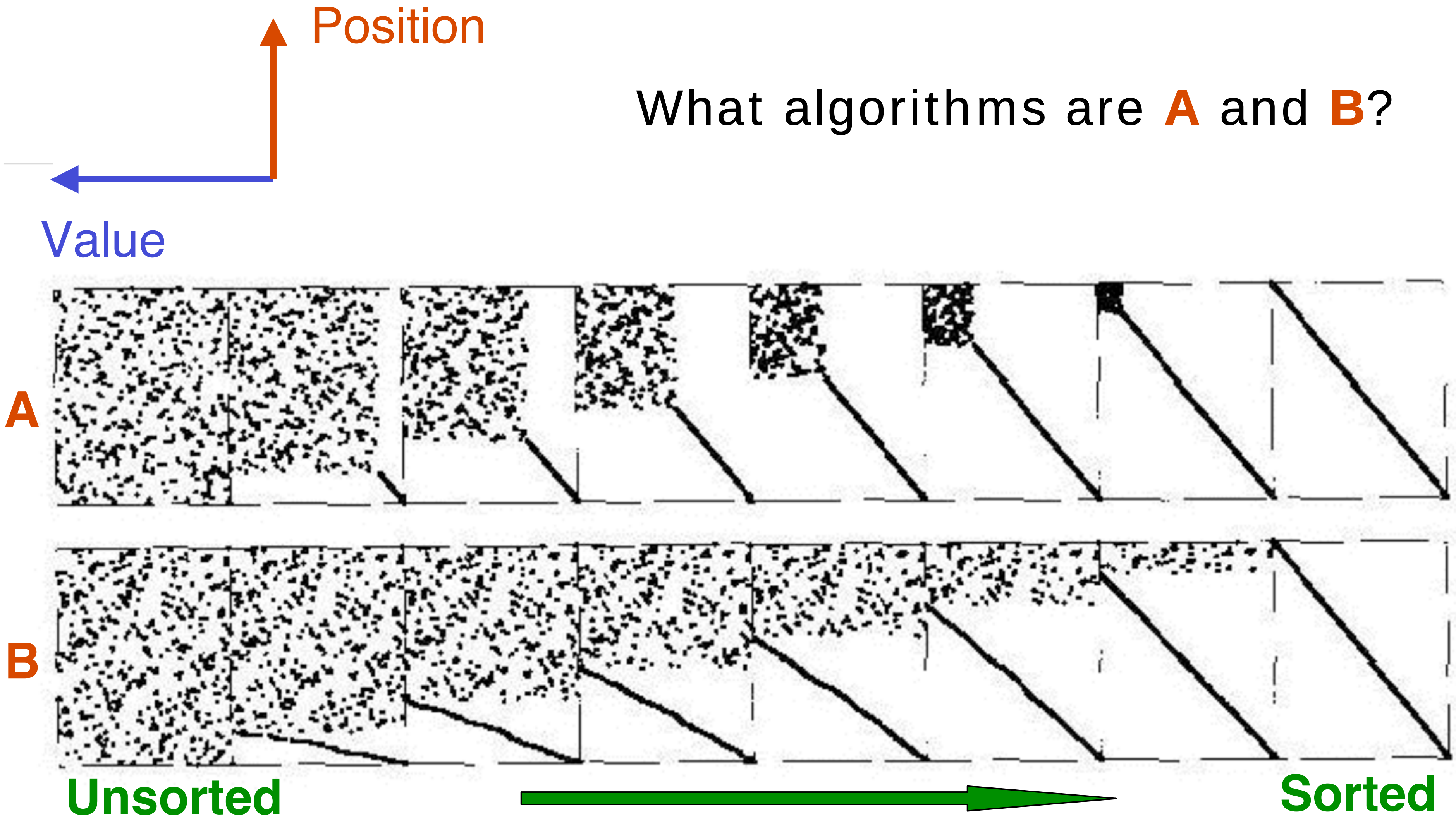
$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Now replace $x = 1/2$ to show that

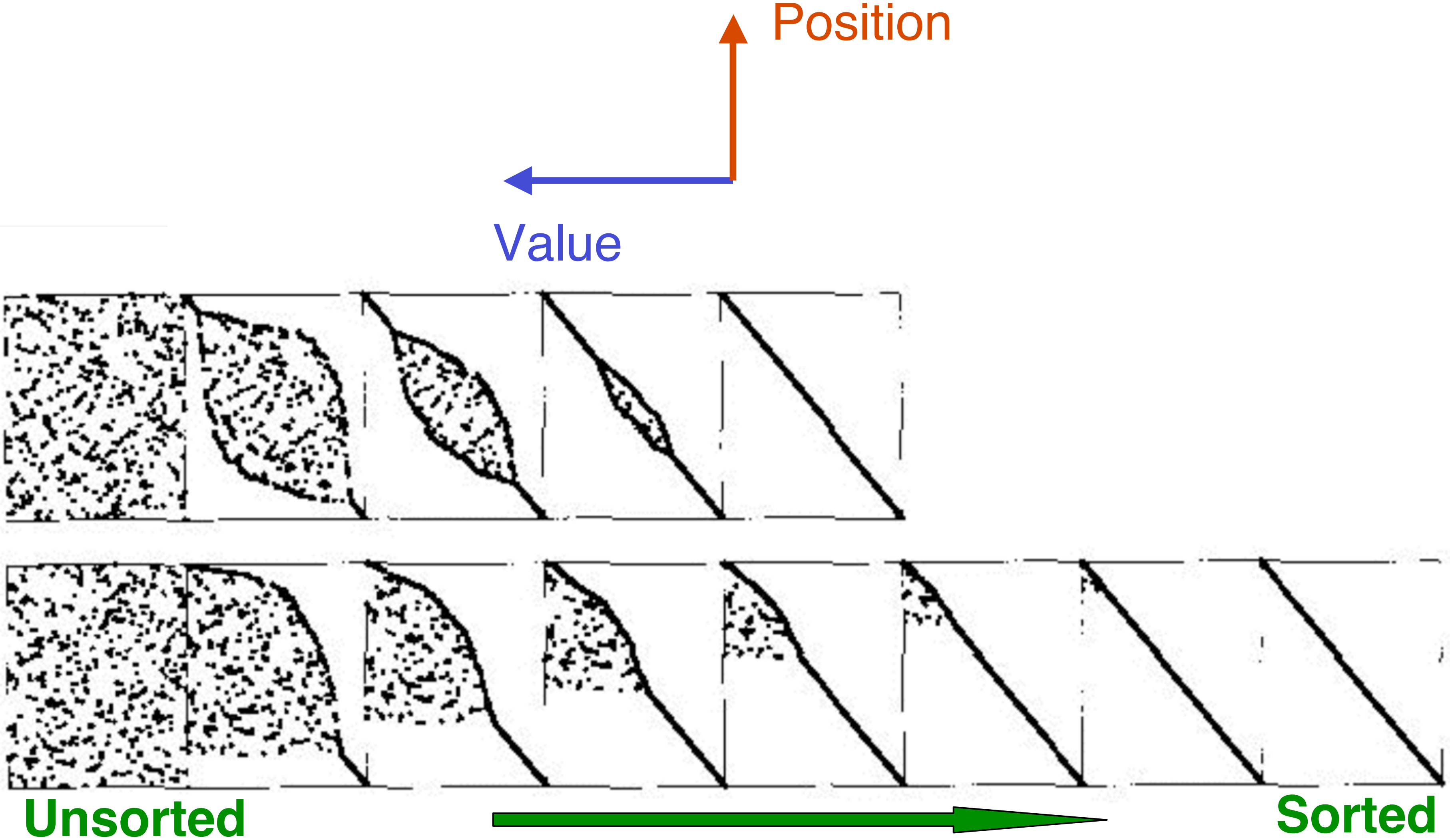
$$\sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \leq \frac{1}{2}$$

Visualizing Algorithms 1

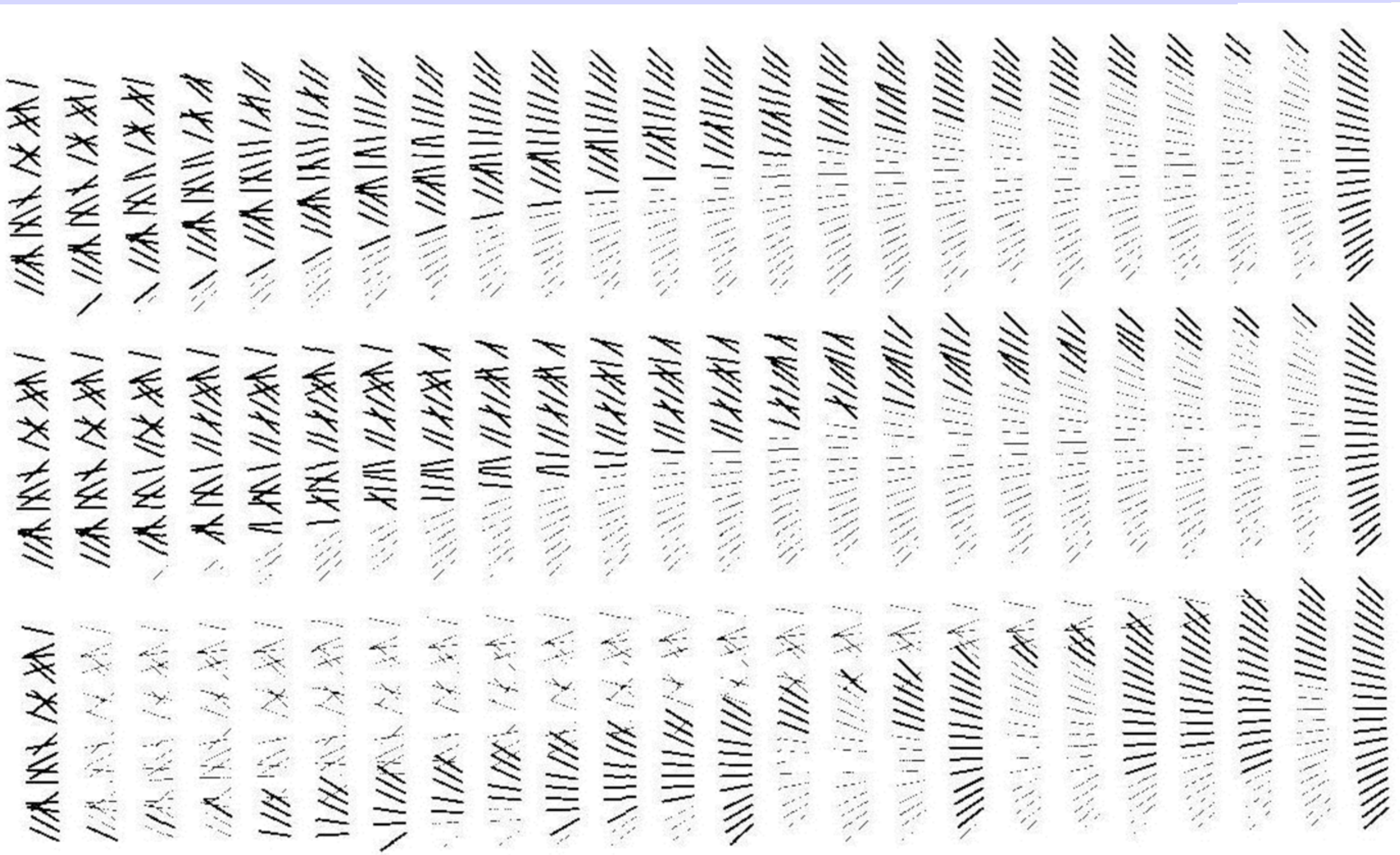
What algorithms are **A** and **B**?



Visualizing Algorithms 2



Visualizing Comparisons 3



Animations

- http://cg.scs.carleton.ca/~morin/misc/sortalg/
- http://home.westman.wave.ca/~rhenry/sort/
  - time complexities on best, worst and average case
- http://vision.bc.edu/~dmartin/teaching/sorting/anim-html/quick3.html
  - runs on almost sorted, reverse, random, and unique inputs; shows code with invariants
- http://www.brian-borowski.com/Sorting/
  - comparisons, movements & stepwise animations with user data
- http://maven.smith.edu/~thiebaut/java/sort/demo.html
  - comparisons & data movements and step by step execution

Problems to think about!

- What is the least number of comparisons you need to sort a list of 3 elements? 4 elements? 5 elements?
- How to arrange a tennis tournament in order to find the tournament champion with the least number of matches? How many tennis matches are needed?