HeapSort: Part 1

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

$\triangleright$ Assume subtree rooted at i is not a heap;
$\triangleright$ 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*)

$heap\text{-}size[A] \leftarrow length[A]$
for $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1
    do MAX-HEAPIFY(A, i)

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

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

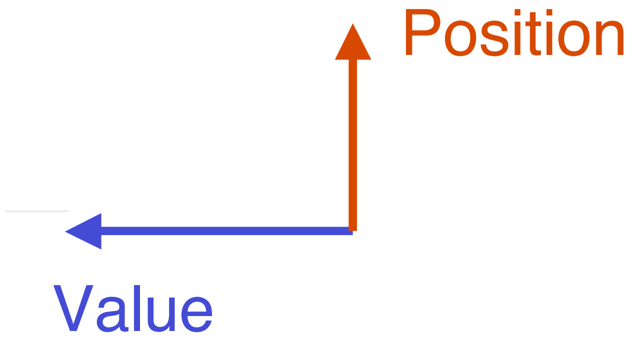
Multiplying both sides by x we get

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

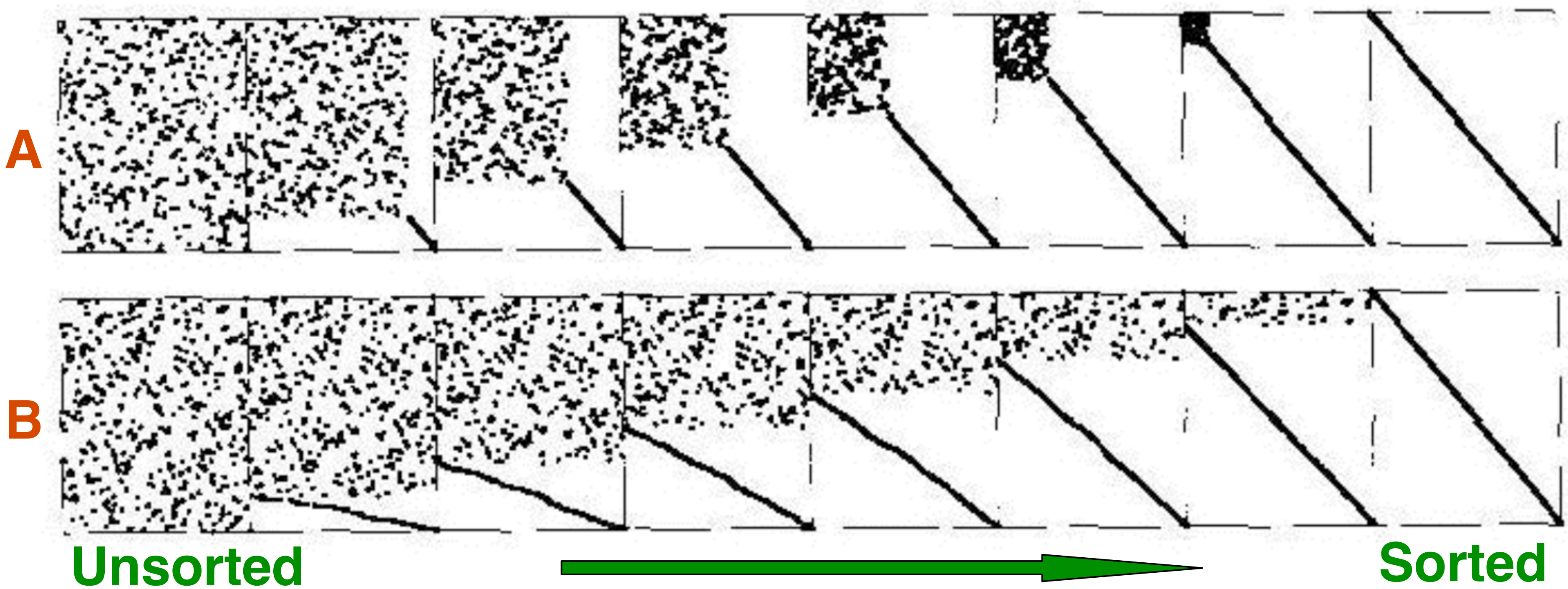
Now replace $x = 1/2$ to show that

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

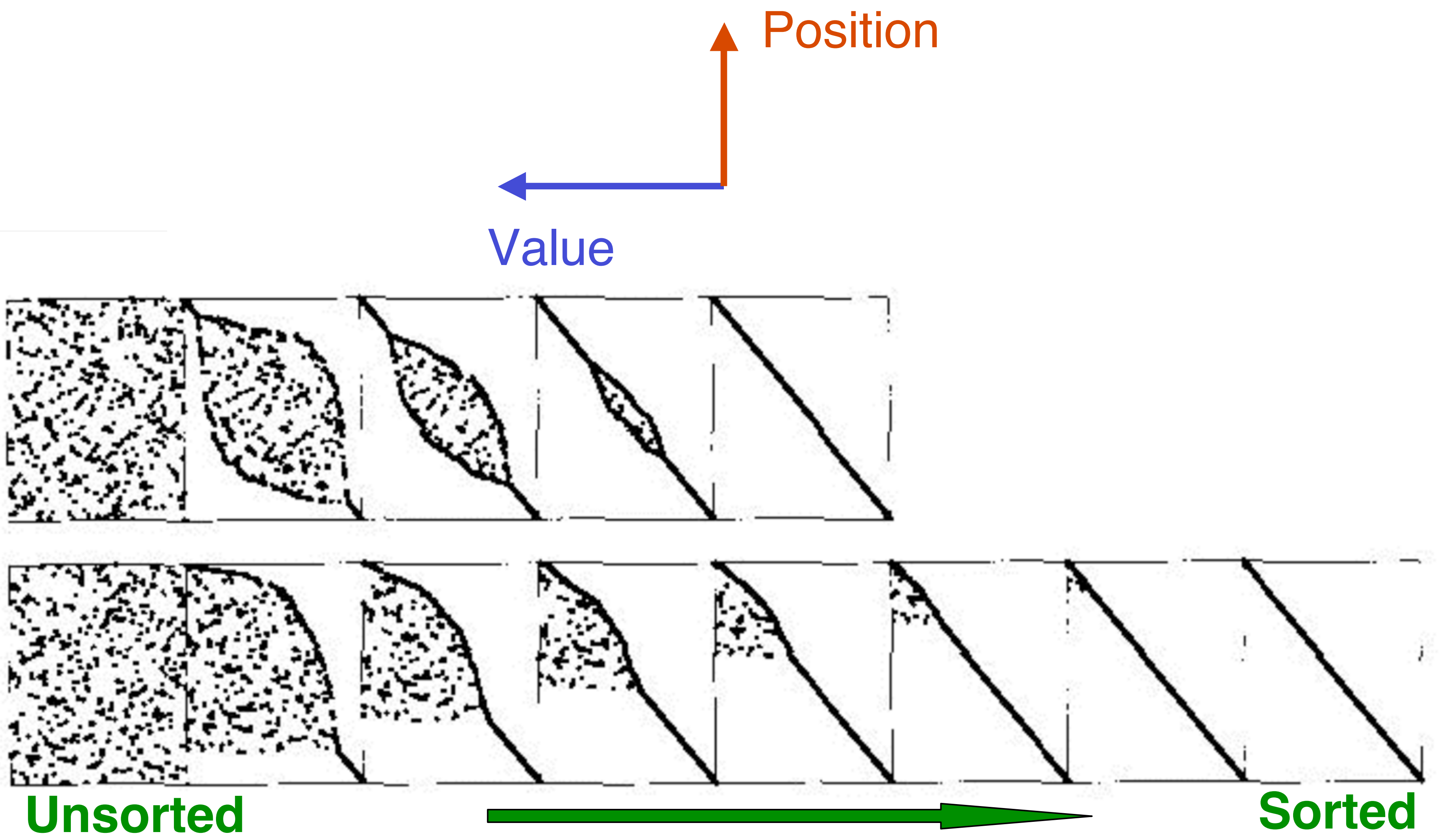
Visualizing Algorithms 1



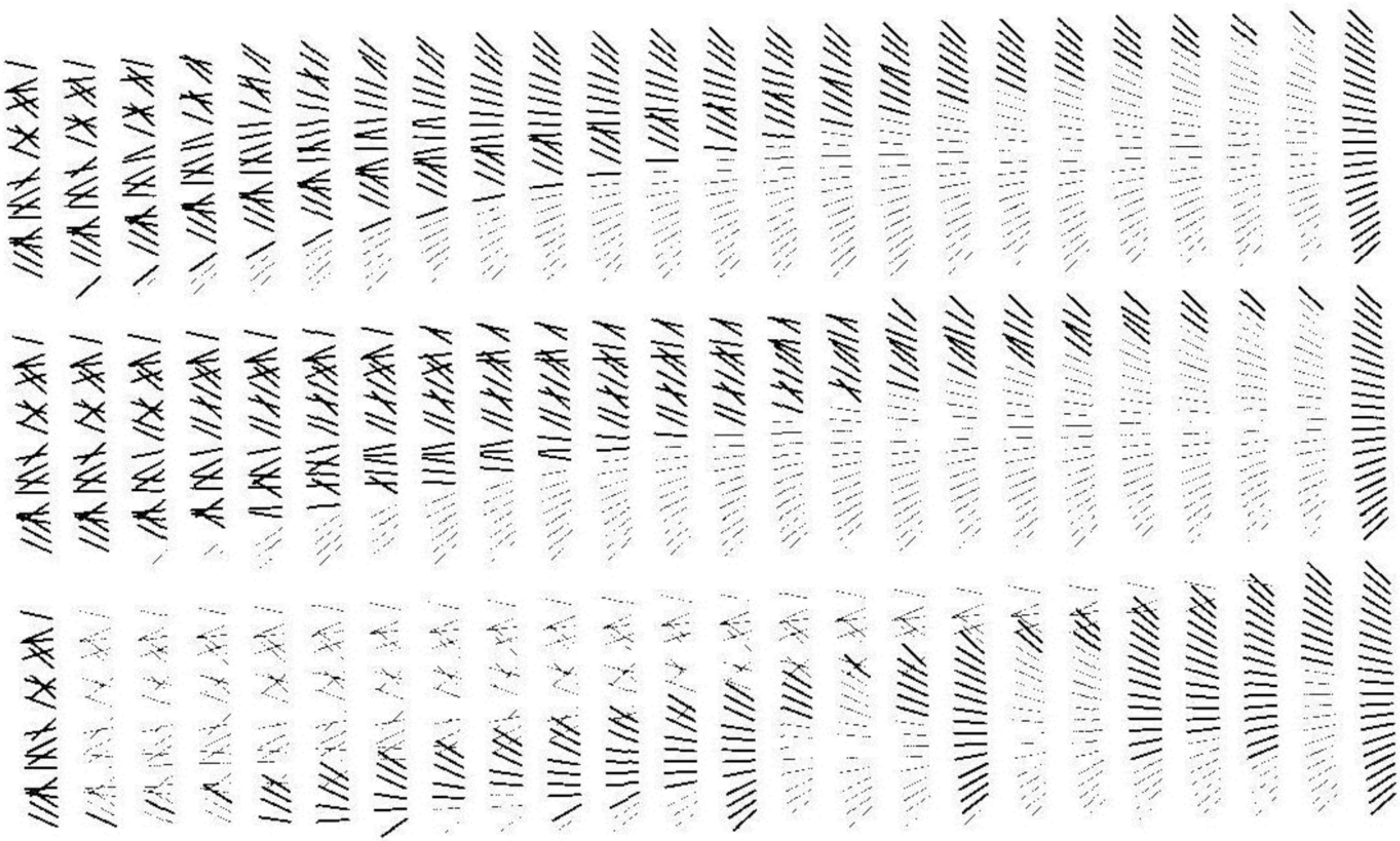
What algorithms are **A** and **B**?



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?

Sorting Algorithms

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

Upper and Lower Bounds

- Define an upper bound on the time complexity of a problem. The upper bound on the time complexity of a problem is $T(n)$ if $\exists$ an algorithm that solves the problem with time complexity $O(T(n))$.
- Clearly upper bound on the time complexity for sorting is $O(n \log n)$.
- Define a lower bound on the time complexity of a problem. The lower bound on the time complexity of a problem is $T(n)$ if $\forall$ algorithms that solve the problem, their time complexity is $\Omega(T(n))$.
- It can be proved that the upper bound is tight! In other words, it can be mathematically proved that the lower bound for sorting is $\Omega(n \log n)$.

Bucket Sort

- N values in the range [a..a+m-1]
- For e.g., sort a list of 50 scores in the range [0..9].
- Algorithm
  - Make m buckets [a..a+m-1]
  - As you read elements throw into appropriate bucket
  - Output contents of buckets [0..m] in that order
- Time O(N+m)

Stable Sort

- A sort is stable if equal elements appear in the same order in both the input and the output.
- Which sorts are stable? Homework!

Radix Sort

3 5 9		3 5 9		3 3 6		3 3 6
3 5 7		3 5 7		3 5 9		3 5 1
3 5 1		3 5 1		3 5 7		3 5 5
7 3 9	⇒	3 3 6	⇒	3 5 1	⇒	3 5 7
3 3 6		3 5 5		3 5 5		3 5 9
7 2 0		7 3 9		7 2 0		7 2 0
3 5 5		7 2 0		7 3 9		8 3 9

Algorithm

for i = 1 **to** d **do**

    sort array A on digit i using any sorting algorithm

Time Complexity: $O((N+m) + (N+m^2) + \ldots + (N+m^d))$

Space Complexity: $O(m^d)$

Radix Sort

3 2 9	7 2 0	7 2 0	3 2 9
4 5 7	3 5 5	3 2 9	3 5 5
6 5 7	4 3 6	4 3 6	4 3 6
8 3 9	4 5 7	8 3 9	4 5 7
4 3 6	6 5 7	3 5 5	6 5 7
7 2 0	3 2 9	4 5 7	7 2 0
3 5 5	8 3 9	6 5 7	8 3 9

Algorithm

for i = 1 **to** d **do**

  sort array A on digit i using a stable sort algorithm

Time Complexity: O((n+m)d)

Counting Sort

Initial Array

1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	3

Counts

0	1	2	3	4	5
2	0	2	3	0	1

Cumulative Counts

0	1	2	3	4	5
2	2	4	7	7	8

External Sorting Methods

- Assumptions:
  - data is too large to be held in main memory;
  - data is read or written in blocks;
  - 1 or more external devices available for sorting
- Sorting in main memory is cheap or free
- Read/write costs are the dominant cost
- Wide variety of storage types and costs
- No single strategy works for all cases

External Merge Sort

- Initial distribution pass
- Several multi-way merging passes

ASORTINGANDMERGINGEXAMPLEWITHFORTYFIVERECORDS.$

AOS.DMN.AEX.FHT.ERV.$

IRT.EGR.LMP.ORT.CEO.$

AGN.GIN.EIW.FIY.DRS.$

AAGINORST.FFHIORTTY.$

DEGGIMNNR.CDEEORRSV.$

AEEILMPWX.$

AAADEEEGGGIIILMMNNNOPRRSTWX.$

CDEEFFHIOORRRSTTVY.$

AAACDDEEEEEEFFGGGHIIIILMMNNNOOOPRRRRRSSTTTWXY.$

With 2P external devices
Space for M records in main memory
Sorting N records needs
$1 + \log_P(N/M)$ passes