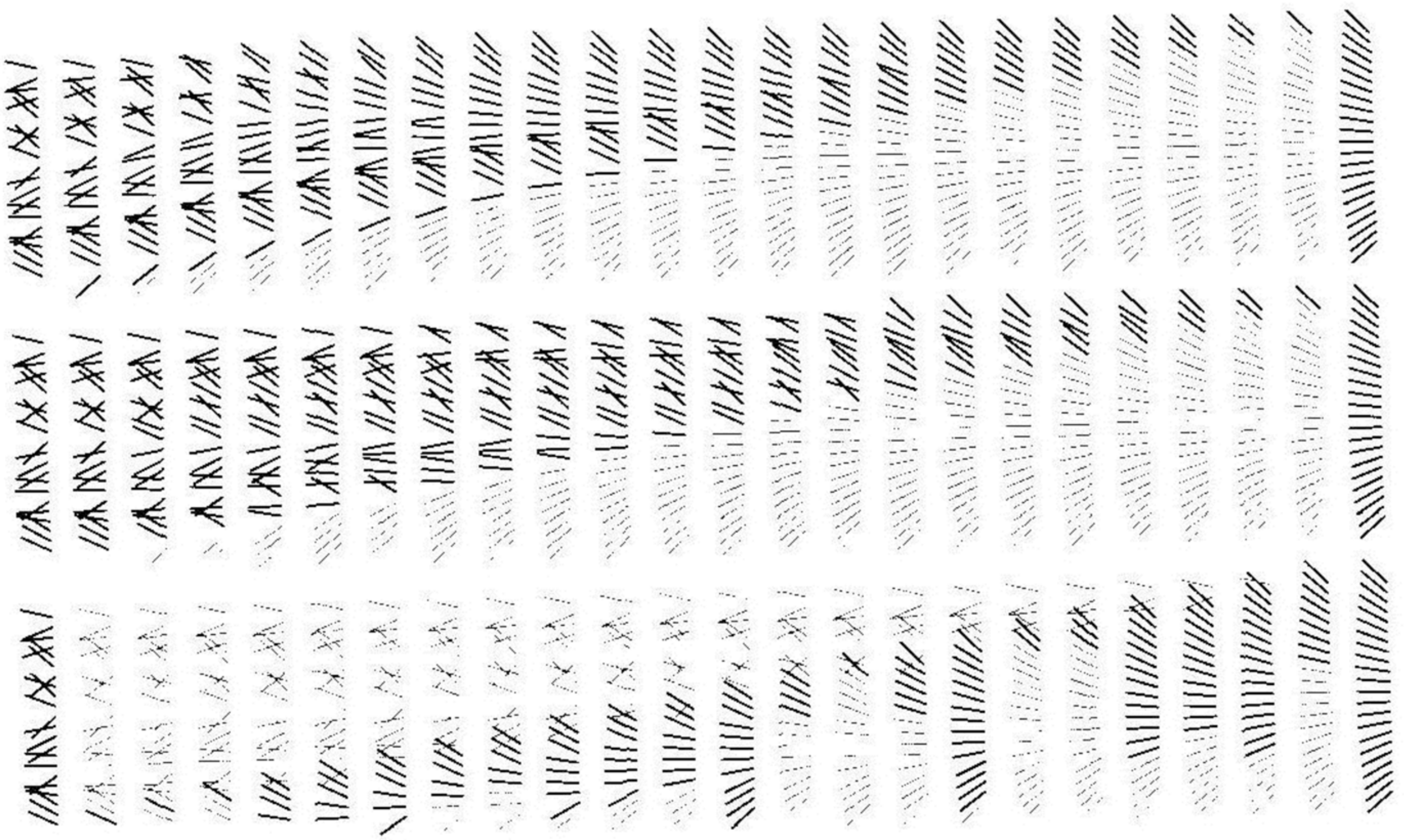
Exam Dates (Tentative)

- Midterm October 9
- Final Exam December 11 (??)
- Homework Assignments
  - Sep 11, Sep 23, Oct 2, Oct 14, Oct 23, Nov 4, Nov 18
- Quizzes
  - Sep 23, Oct 2, Oct 14, Oct 23, Nov 4, Nov 18,
- Semester Project October 1

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

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 ∃ 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 ∀ 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

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?

Order Statistics

- Maximum, Minimum   n-1 comparisons

7	3	1	9	4	8	2	5	0	6

- MinMax
  - 2(n-1) comparisons
  - 3n/2 comparisons
- Max and 2ndMax
  - (n-1) + (n-2) comparisons
  - ???

k-Selection; Median

- Select the k-th smallest item in list
- Naïve Solution
  - Sort;
  - pick the k-th smallest item in sorted list.

    $O(n \log n)$ time complexity
- Randomized solution: Average case $O(n)$
- Improved Solution: worst case $O(n)$

QuickSort

```
QuickSort(array A, int p, int r)
if (p < r)
    then q ← Partition(A, p, r)
         QuickSort(A, p, q − 1)
         QuickSort(A, q + 1, r)
```

To sort array call QuickSort(A, 1, *length*[A]).

```
Partition(array A, int p, int r)
x ← A[r]                          ▷ Choose pivot
i ← p − 1
for j ← p to r − 1
    do if (A[j] ≤ x)
        then i ← i + 1
             exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Page 146, CLRS

RandomizedPartition

- RandomizedPartition picks the pivot uniformly at random from among the elements in the list to be partitioned.

Homework

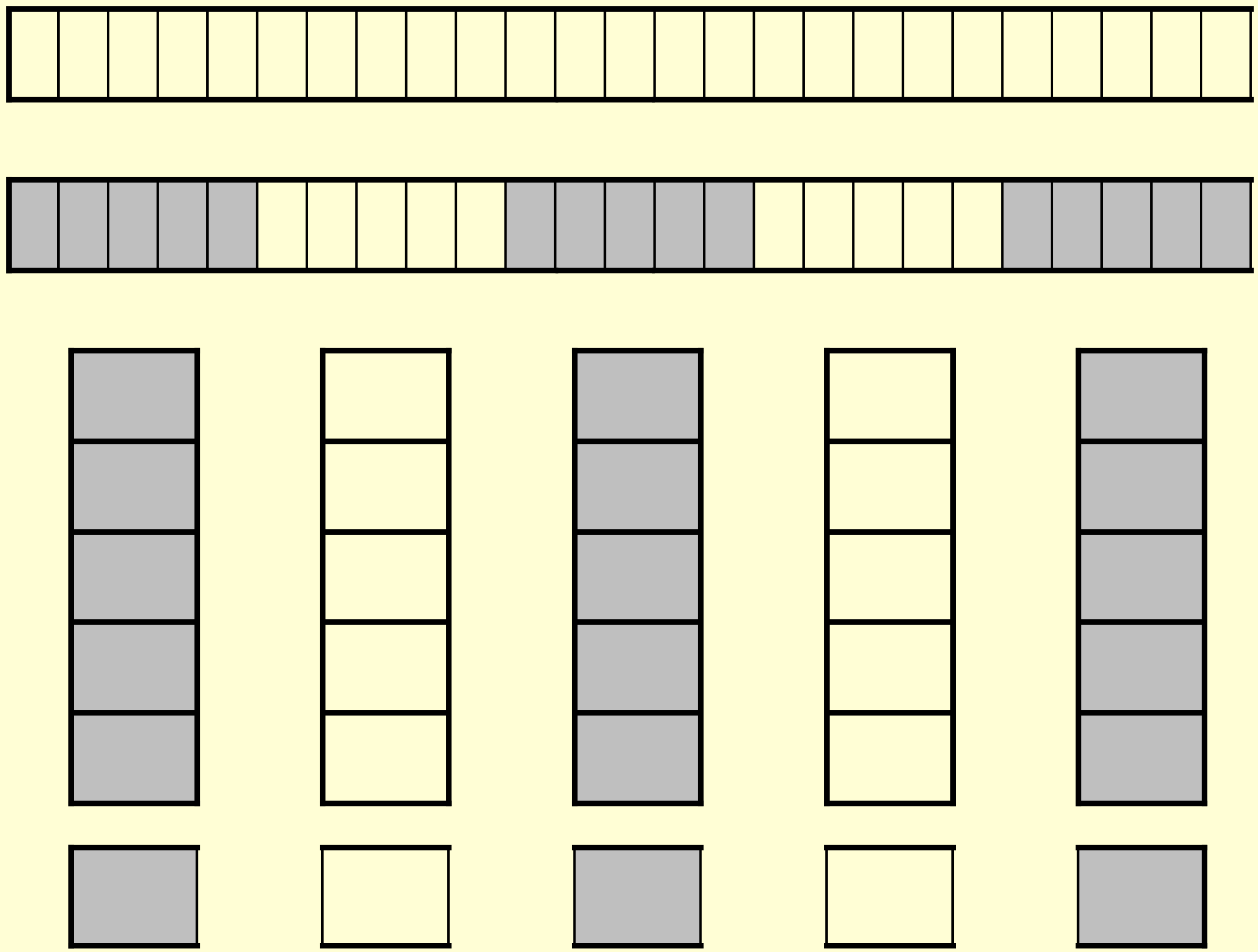
- **Statement of Collaboration**
  - Take it **seriously.**
  - **Reproduce** the statement faithfully and sign it by hand.
  - For each problem, explain **separately** the sources and your collaborations with other people.
  - Your homework **will not be graded** without the signed statement.
- **Extra Credit Problem**
  - You can turn it in any time until second last class day (Dec 4th).
  - You may retry a problem, but don't waste my time.
  - You will not get partial credit on an extra credit problem.
  - Put it on a separate sheet of paper and label it appropriately.

QuickSelect: a variant of QuickSort

```
QUICKSELECT(array A, int k, int p, int r)
   ▷ Select k-th largest in subarray A[p..r]
   if (p = r)
       then return A[p]
   q ← PARTITION(A, p, r)
   i ← q − p + 1          ▷ Compute rank of pivot
   if (i = k)
       then return A[q]
   if (i > k)
       then return QUICKSELECT(A, k, p, q)
       else return QUICKSELECT(A, k − i, q + 1, r)
```

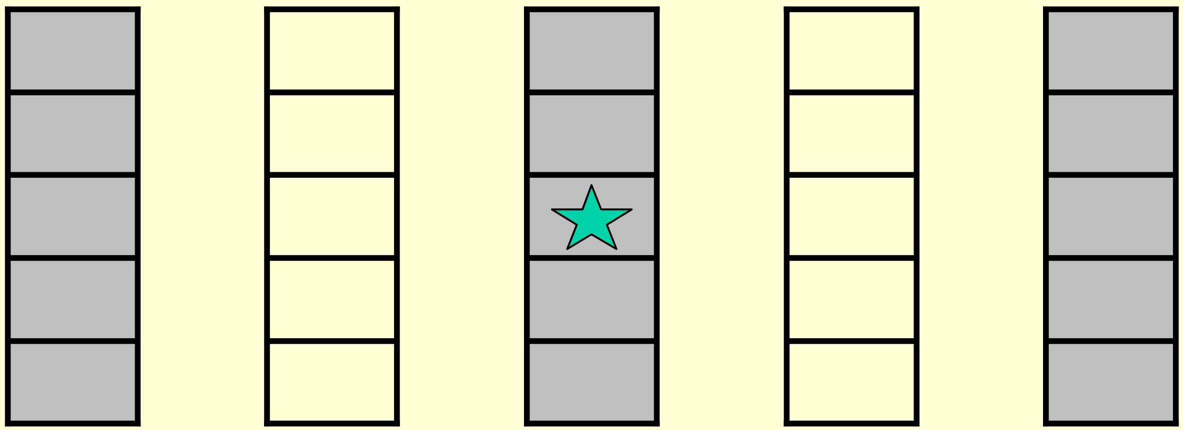
k-Selection & Median: Improved Algorithm

- Start with initial array



k-Selection & Median: Improved Algorithm(Cont'd)

- Use median of medians as pivot



- $T(n) < O(n) + T(n/5) + T(3n/4)$

ImprovedSelect

ImprovedSelect(*array* A, *int* k, *int* p, *int* r)

$\triangleright$ Select k-th largest in subarray $A[p..r]$

if $(p = r)$

  then return $A[p]$

  else $N \leftarrow r - p + 1$

Partition $A[p..r]$ into subsets of 5 elements and collect all medians of subsets in $B[1..\lceil N/5 \rceil]$.

$Pivot \leftarrow$ ImprovedSelect$(B, 1, \lceil N/5 \rceil, \lceil N/10 \rceil)$

$q \leftarrow$ PivotPartition$(A, p, r, Pivot)$

$i \leftarrow q - p + 1$ $\triangleright$ Compute rank of pivot

if $(i = k)$

  then return $A[q]$

if $(i > k)$

  then return ImprovedSelect$(A, k, p, q - 1)$

  else return ImprovedSelect$(A, k - i, q + 1, r)$

PivotPartition

$\textsc{PivotPartition}(array\ A, int\ p, int\ r, item\ Pivot)$

$\triangleright$ Partition using provided $Pivot$

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** r

do if $(A[j] \leq Pivot)$

then $i \leftarrow i + 1$

exchange $A[i] \leftrightarrow A[j]$

return $i + 1$