EXAM RECAP

• Overall, you did well
  • Average in the low 70s
EXAM RECAP

• Overall, you did well
  • Average in the low 70s
  • Q3 was the tricky one
EXAM RECAP

• Overall, you did well
  • Average in the low 70s
  • Q3 was the tricky one
  • AVL, Hashtables, Heaps, B-Trees
EXAM RECAP

• Overall, you did well
  • Average in the low 70s
  • Q3 was the tricky one
  • AVL, Hash tables, Heaps, B-Trees
  • Analysis/short answer
EXAM RECAP

• If you did poorly,
  • Email me about a meeting
  • Quarter isn’t over yet
  • Don’t wait until finals week
EXAM RECAP

• Regrades
  • No office hours today
  • I will be in my office before class Wednesday and Friday from 12:00-2:00 to handle regrades
  • Come prepared with the exam and why you think the grade is incorrect
  • TAs can help you with solutions or problems, but I will make all grade changes
EXAM RECAP

• Final Exam
  • Cumulative
EXAM RECAP

• Final Exam
  • Cumulative
  • Less time pressure
EXAM RECAP

• Final Exam
  • Cumulative
  • Less time pressure
  • More critical thought (but there will still be a few procedural questions)
EXAM RECAP

• Final Exam
  • Cumulative
  • Less time pressure
  • More critical thought (but there will still be a few procedural questions)
  • Algorithm analysis
ASSORTED MINUTIAE

- P2 grades to those who submitted
  - Grades by Wednesday for partners
  - Email me if you’re missing your feedback file
ASSORTED MINUTIAE

• P2 grades to those who submitted
  • Grades by Wednesday for partners
  • Email me if you’re missing your feedback file
• P3 out tonight
ASSORTED MINUTIAE

• P2 grades to those who submitted
  • Grades by Wednesday for partners
  • Email me if you’re missing your feedback file
• P3 out tonight
  • Part 1 due next Wednesday
  • Try to get ahead on the assignment
ASSORTED MINUTIAE

• P2 grades to those who submitted
  • Grades by Wednesday for partners
  • Email me if you’re missing your feedback file

• P3 out tonight
  • Part 1 due next Wednesday
  • Try to get ahead on the assignment
  • 3 parts – only one written assignment left
ASSORTED MINUTIAE

- Written assignments make up 10% of total grade
- Coding assignments make up 30% of total grade (weighted per part)
- Exam
  - Higher exam grade worth 35%
  - Lower exam grade worth 25%
SORTING

INEFFECTIVE Sorts

DEFINE FASTBOGOSORT(list):
   // AN OPTIMIZED BOGOSORT
   // RUNS IN O(N LOG N)
   FOR N FROM 1 TO LOG(LENGTH(list)):
      SHUFFLE(list):
      IF isSorted(list):
         RETURN list
   RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"

DEFINE PAINCSORT(list):
   IF isSorted(list):
      RETURN list
   FOR N FROM 1 TO 10000:
      PIVOT = RANDOM(0, LENGTH(list))
      LIST = list[PIVOT]+list[:PIVOT]
      IF isSorted(list):
         RETURN list
   IF isSorted(list):
      RETURN list
   IF isSorted(list):
      // THIS CAN'T BE HAPPENING
      RETURN list
   IF isSorted(list):
      // COME ON COME ON
      RETURN list
   // OH JEEZ
   // I'M GONNA BE IN SO MUCH TROUBLE
   LIST = []
   SYSTEM("SHUTDOWN -H +5")
   SYSTEM("RM -RF ")
   SYSTEM("RM -RF ~")
   SYSTEM("RM -RF ")
   SYSTEM("RD /S/Q C:") // PORTABILITY
   RETURN [1, 2, 3, 4, 5]

DEFINE JOB Weiner|Quicksort(list):
   OK SO YOU CHOOSE A PIVOT
   THEN DIVIDE THE LIST IN HALF
   FOR EACH HALF:
      CHECK TO SEE IF IT'S SORTED
      NO WHATEVER IT DOESN'T MATTER
      COMPARE EACH ELEMENT TO THE PIVOT
      THE BIGGER ONES GO IN A NEW LIST
      THE EQUAL ONES GO IN THE OTHER LIST
      THE SECOND LIST FROM BEFORE
      HANG ON, LET ME NAME THE LISTS
      THIS IS LIST A
      THE NEW ONE IS LIST B
      PUT THE BIG ONES INTO LIST B
      NOW TAKE THE SECOND LIST
      CALL IT LIST UH, A2
      WHICH ONE WAS THE PIVOT IN?
      SCRATCH THAT ALL THAT
      IT JUST RECURSIVELY CALLS ITSELF
      UNTIL BOTH LISTS ARE EMPTY
      RIGHT?
      NOT EMPTY, BUT YOU KNOW WHAT I MEAN
      AM I ALLOWED TO USE THE STANDARD LIBRARIES?
SORTING

- Problem statement:
SORTING

• Problem statement:
  • Given some collection of comparable data, arrange them into an organized order
SORTING

• Problem statement:
  • Given some collection of comparable data, arrange them into an organized order
  • Important to note that you may be able to “organize” the same data different ways
SORTING

• Why sort at all?
SORTING

• Why sort at all?
  • Data pre-processing
SORTING

• Why sort at all?
  • Data pre-processing
  • If we do the work now, future operations may be faster
SORTING

• Why sort at all?
  • Data pre-processing
  • If we do the work now, future operations may be faster
  • Unsorted v. Sorted Array, e.g.
SORTING

• Why sort at all?
  • Data pre-processing
  • If we do the work now, future operations may be faster
  • Unsorted v. Sorted Array, e.g.

• Why not just maintain sortedness as we add?
SORTING

• Why sort at all?
  • Data pre-processing
  • If we do the work now, future operations may be faster
  • Unsorted v. Sorted Array, e.g.

• Why not just maintain sortedness as we add?
  • Most times, if we can, we should
SORTING

• Why sort at all?
  • Data pre-processing
  • If we do the work now, future operations may be faster
  • Unsorted v. Sorted Array, e.g.

• Why not just maintain sortedness as we add?
  • Most times, if we can, we should
  • Why would we not be able to?
SORTING

• Maintaining Sortedness v. Sorting
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
    • Data comes in batches
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
    • Data comes in batches
    • Multiple “sorted” orders
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
    • Data comes in batches
    • Multiple “sorted” orders
    • Costly to maintain!
SORTING

• Maintaining Sortedness v. Sorting
  • Why *don’t* we maintain sortedness?
    • Data comes in batches
    • Multiple “sorted” orders
    • Costly to maintain!

• We need to be sure that the effort is worth the work
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
    • Data comes in batches
    • Multiple “sorted” orders
    • Costly to maintain!
• We need to be sure that the effort is worth the work
  • No free lunch!
SORTING

• Maintaining Sortedness v. Sorting
  • Why don’t we maintain sortedness?
    • Data comes in batches
    • Multiple “sorted” orders
    • Costly to maintain!

• We need to be sure that the effort is worth the work
  • No free lunch!

• What does that even mean?
BOGO SORT

- Consider the following sorting algorithm
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
BOGO SORT

- Consider the following sorting algorithm
  - Shuffle the list into a random order
  - Check if the list is sorted,
  - if so return the list
  - if not, try again
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
  • if not, try again
• What is the problem here?
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
  • if not, try again
• What is the problem here?
  • Runtime!
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
  • if not, try again
• What is the problem here?
  • Runtime! Average $O(n!)$!
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
  • if not, try again
• What is the problem here?
  • Runtime! Average $O(n!)$!
  • Why is this so bad?
BOGO SORT

• Consider the following sorting algorithm
  • Shuffle the list into a random order
  • Check if the list is sorted,
  • if so return the list
  • if not, try again

• What is the problem here?
  • Runtime! Average $O(n!)$!
  • Why is this so bad?

• The computer isn’t thinking, it’s just guess-and-checking
SORTING

- Guess-and-check
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
    • Breaking RSA
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
    • Breaking RSA
    • Greedy-first algorithms
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
    • Breaking RSA
    • Greedy-first algorithms
    • Midterms
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
    • Breaking RSA
    • Greedy-first algorithms
    • Midterms
  • If you don’t have a lot of time, or if the payoff is big, or if the chance of success is high, then it might be a good strategy
SORTING

• Guess-and-check
  • Not a bad strategy when nothing else is obvious
    • Breaking RSA
    • Greedy-first algorithms
    • Midterms
  • If you don’t have a lot of time, or if the payoff is big, or if the chance of success is high, then it might be a good strategy
  • Random/Approximated algs
SORTING

• Why not guess-and-check for sorting?
SORTING

• Why not guess-and-check for sorting?
  • Not taking advantage of the biggest constraint of the problem
SORTING

• Why not guess-and-check for sorting?
  • Not taking advantage of the biggest constraint of the problem
  • Items must be comparable!
SORTING

• Why not guess-and-check for sorting?
  • Not taking advantage of the biggest constraint of the problem
  • Items must be comparable!
  • You should be comparing things!
  • Looking at two items next to each other tells a lot about where they belong in the list, there’s no reason not to use this information.
SORTING

- Types of sorts
SORTING

- Types of sorts
  - Comparison sorts
SORTING

- Types of sorts
  - Comparison sorts
    - Bubble sort
SORTING

• Types of sorts
  • Comparison sorts
    • Bubble sort
    • Insertion sort
SORTING

• Types of sorts
  • Comparison sorts
    • Bubble sort
    • Insertion sort
    • Selection sort
SORTING

• Types of sorts
  • Comparison sorts
    • Bubble sort
    • Insertion sort
    • Selection sort
    • Heap sort
SORTING

• Types of sorts
  • Comparison sorts
    • Bubble sort
    • Insertion sort
    • Selection sort
    • Heap sort
  • “Other” sorts
    • Bucket sort – will talk about later
SORTING

• Types of sorts
  • Comparison sorts
    • Bubble sort
    • Insertion sort
    • Selection sort
    • Heap sort
  • “Other” sorts
    • Bucket sort – will talk about later
    • Bogo sort
DEFINITION: COMPARISON SORT

- A computational problem with the following input and output
  - Input:
    - An array \( A \) of length \( n \) comparable elements
  - Output:
    - The same array \( A \), containing the same elements where:
      - for any \( i \) and \( j \) where \( 0 \leq i < j < n \)
      - \( A[i] \leq A[j] \)
MORE REASONS TO SORT

General technique in computing:

*Preprocess data to make subsequent operations faster*

Example: Sort the data so that you can

- Find the $k^{th}$ largest in constant time for any $k$
- Perform binary search to find elements in logarithmic time

Whether the performance of the preprocessing matters depends on

- How often the data will change (and how much it will change)
- How much data there is
MORE DEFINITIONS

In-Place Sort:
A sorting algorithm is in-place if it requires only $O(1)$ extra space to sort the array.
• Usually modifies input array
• Can be useful: lets us minimize memory

Stable Sort:
A sorting algorithm is stable if any equal items remain in the same relative order before and after the sort.
• Items that ‘compare’ the same might not be exact duplicates
• Might want to sort on some, but not all attributes of an item
• Can be useful to sort on one attribute first, then another one
STABLE SORT EXAMPLE

Input:

[(8, "fox"), (9, "dog"), (4, "wolf"), (8, "cow")]

Compare function: compare pairs by number only

Output (stable sort):

[(4, "wolf"), (8, "fox"), (8, "cow"), (9, "dog")]

Output (unstable sort):

[(4, "wolf"), (8, "cow"), (8, "fox"), (9, "dog")]

SORTING: THE BIG PICTURE

Simple algorithms:
- Insertion sort
- Selection sort
- Shell sort

Fancier algorithms:
- Heap sort
- Merge sort
- Quick sort (avg)

Comparison lower bound:
- $\Omega(n \log n)$

Specialized algorithms:
- Bucket sort
- Radix sort

Handling huge data sets
External sorting
INSERTION SORT

1. Insert where it belongs in sorted section.
2. Shift other elements over and already sorted section is now larger.
3. New current item.

2 4 5 3 8 7 1 6
already sorted  unsorted

2 4 5 3 8 7 1 6
already sorted  unsorted

2 3 4 5 8 7 1 6
already sorted  unsorted

2 3 4 5 8 7 1 6
already sorted  unsorted
INSERTION SORT

Idea: At step $k$, put the $k^{th}$ element in the correct position among the first $k$ elements

```java
for (int i = 0; i < n; i++) {
    // Find index to insert into
    int newIndex = findPlace(i);
    // Insert and shift nodes over
    shift(newIndex, i);
}
```

What can we say about the list at loop $i$? first $i$ elements are sorted (not necessarily lowest in the list)

Runtime?
INSERTION SORT

Idea: At step $k$, put the $k^{th}$ element in the correct position among the first $k$ elements

```java
for (int i = 0; i < n; i++) {
    // Find index to insert into
    int newIndex = findPlace(i);
    // Insert and shift nodes over
    shift(newIndex, i);
}
```

What can we say about the list at loop $i$? first $i$ elements are sorted (not necessarily lowest in the list)

Runtime? Best case: $O(n)$, Worst case: $O(n^2)$ Why?
**INSERTION SORT**

Idea: At step \( k \), put the \( k \)th element in the correct position among the first \( k \) elements.

```java
for (int i = 0; i < n; i++) {
    // Find index to insert into
    int newIndex = findPlace(i);
    // Insert and shift nodes over
    shift(newIndex, i);
}
```

What can we say about the list at loop \( i \)? first \( i \) elements are sorted (not necessarily lowest in the list)

**Runtime?** Best case: \( O(n) \), Worst case: \( O(n^2) \) Why?

**Stable?**

**In-place?**
INSERTION SORT

Idea: At step $k$, put the $k^{th}$ element in the correct position among the first $k$ elements

```java
for (int i = 0; i < n; i++) {
    // Find index to insert into
    int newIndex = findPlace(i);
    // Insert and shift nodes over
    shift(newIndex, i);
}
```

What can we say about the list at loop $i$? first $i$ elements are sorted (not necessarily lowest in the list)

**Runtime?** Best case: $O(n)$, Worst case: $O(n^2)$ Why?

**Stable?** Usually

**In-place?** Yes
SELECTION SORT

1. The current index is 1, and the next smallest element in the unsorted section is 7.

2. The current index is now 2, and the next smallest element is 8.

3. The 'already sorted' section is now one element larger.

4. The current index is 3, and the next smallest element is 4.

5. The next smallest element is swapped with the current index element.
SELECTION SORT

- Can be interrupted (don’t need to sort the whole array to get the first element)
- Doesn’t need to mutate the original array (if the array has some other sorted order)
- Stable sort
INSERTION SORT VS. SELECTION SORT

Have the same worst-case and average-case asymptotic complexity

- Insertion-sort has better best-case complexity; preferable when input is “mostly sorted”

Useful for small arrays or for mostly sorted input
SORTING: THE BIG PICTURE

Simple algorithms: $O(n^2)$

- Insertion sort
- Selection sort
- Shell sort

Fancier algorithms: $O(n \log n)$

- Heap sort
- Merge sort
- Quick sort (avg)

Comparison lower bound: $\Omega(n \log n)$

Specialized algorithms: $O(n)$

- Bucket sort
- Radix sort

Handling huge data sets

External sorting
NEXT CLASS

• Fancier sorts!
NEXT CLASS

• Fancier sorts!
• How fancy can we get?