Searching & Sorting

- Searching
  - Linear or sequential
  - Binary
- Sorting
  - quadratic sorts
  - divide & conquer sorts
  - data structures

Searching

- looking for a particular item in a database
- Success (matching) is not always possible
- algorithms
  - linear or sequential search
  - binary search
- data structures
  - binary search trees
  - hash tables

Linear or Sequential Search

- start with the first element
- if found stop, else go to the next element
- iterate until found or search is exhausted

Linear Search Analysis

- How many comparisons?
  - 1 element 1
  - 2 elements 2
  - 3 elements 3
  - n elements n

$$1 + 2 + 3 + \ldots + n = (n + 1)\, n / 2 = (n+1)/2$$

Binary Search

- find the middle element in database
- if found stop, else decide which half to search
- find the middle element in that half and repeat process
- stop when item is found or search is exhausted

Binary Search Analysis

- data must be sorted
- How many comparisons?
  - 1 time n/2
  - 2 time n/4
  - 3 time n/8
  - n times $n/2^p$

  - p = divisions = $\log_2 n$

Binary Search analysis

- How many iterations are needed before we end up with no elements left to examine?

Array length	iterations
15	4
31	5
63	6

As the size of the data doubles, the number of comparisons increments by one!

Sorting

- Ordering elements in a list
  - increasing order
  - alphabetically
  - decreasing order

Why sorting?

- In general, list are sorted!
- Makes add, delete, and modify more efficient
- In general, ordered lists are easier to read and display

Sorting techniques

- internal sorting
  - data is sorted using computer’s main memory
- external sorting
  - using secondary storage (files, disks, tapes, etc.)

Quadratic Sorts

- They have a worst case running time of $O(n^2)$
- Selection sort performs poorly even is the values are already sorted!
- Insertion sort performs reasonably well if the values are almost sorted

Quadratic sorts

- insertion sort
- selection sort
- exchange (bubble) sort

Selection Sort

- select the smallest (or largest) value
- swap its position with value in position 1
- repeat process until list is sorted
- Analysis
  - only two elements change at the time
  - minimizes swaps
  - independent of original ordering
  - worst, best, and expected runtime is $O(n^2)$

Exchange Sort (Bubble)

- start at one end of the list
- compare the first two elements
- sort them in increasing order (bubble up)
- repeat the process
- Analysis
  - several elements change at one time
  - dependent on initial ordering
  - worst, best, and expected runtime is $O(n^2)$

Insertion Sort

- select one element and insert it in its correct position by moving it to the “sorted” region of the list
- repeat process
- Analysis
  - several elements change at one time
  - dependent on initial ordering
  - average case is $O(n^2)$, best case is $O(n)$ when the data is already sorted

Divide and Conquer sorts

- Divide the elements to be sorted into two groups of approximately equal size
- Sort each group
- Combine the sorted groups into one sorted list

Divide & Conquer sorts

- merge sort
- quick sort

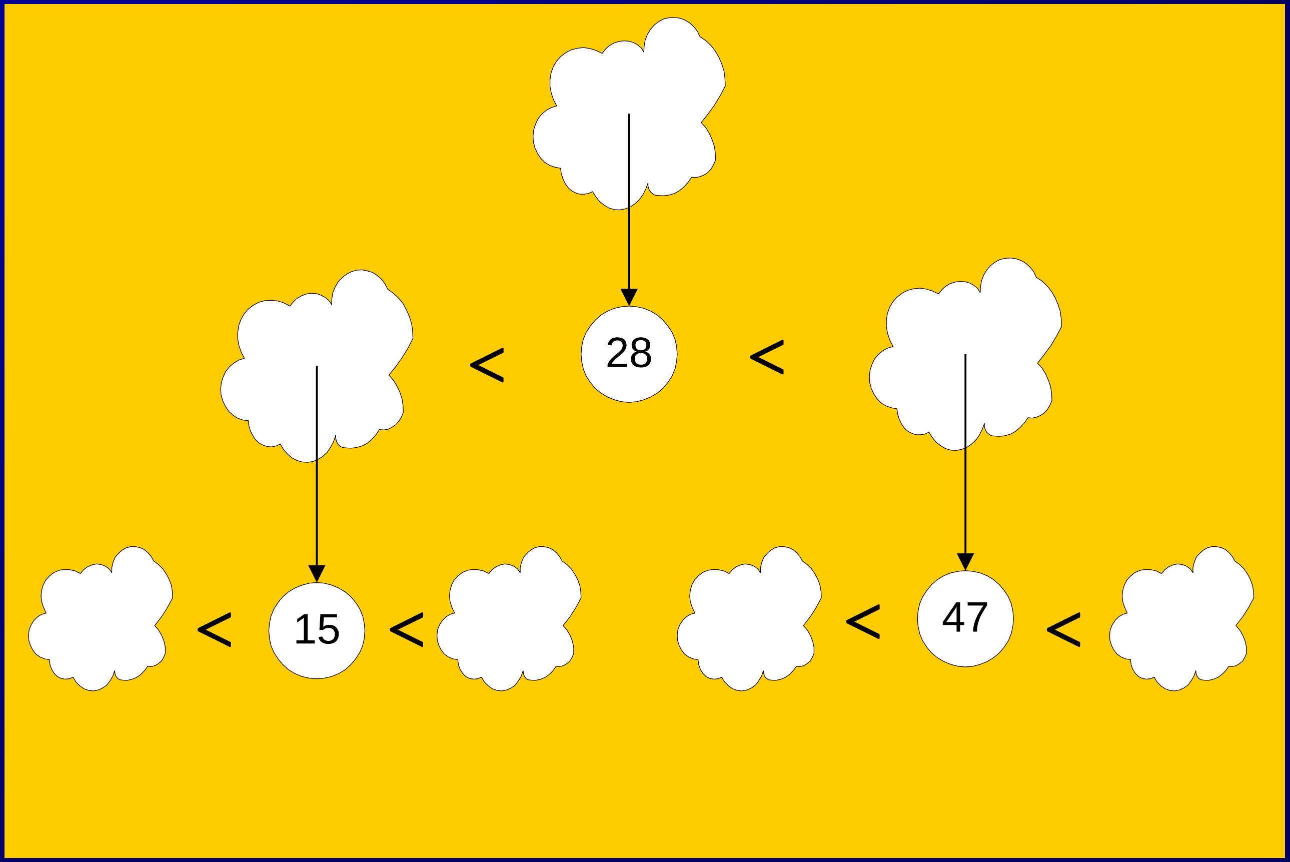
Merge Sort

- divide list in half
- then divide each half in half again
- sort lists when reduced to 1 or 2 elements
- merge sublists recursively
- Analysis
  - $O(n \log_2 n)$
  - uses recursion
  - requires another list to do the merging

Quick Sort

- pick a “pivot” value and its position
- move smaller values to the left of the pivot
- move larger values to the right of the pivot
- apply same technique to each half
- Analysis
  - very fast
  - uses recursion
  - fewer swaps than merge sort
  - average is $O(n \log_2 n)$, worst is $O(n^2)$

QuickSort



Pick a “pivot”. Divide into less-than & greater-than pivot.
Sort each side recursively.

Partitioning the array

- Choose the first element as the pivot
- Starting from the left, search for the first value that is greater than the pivot
- Starting from the right, search for the first value that is less than the pivot
- Swap those two values
- Continue this process until the values searched meet, swap the pivot with the last element in the first half

QuickSort Partition

Pick pivot:

7	2	8	3	5	9	6

Partition with cursors

7	2	8	3	5	9	6

< >

2 goes to less-than

7	2	8	3	5	9	6

< >

Choosing a pivot

- Randomly choose pivot
  - Good in theory
  - call to random number generator can be expensive
- Pick pivot cleverly
  - “Median-of-3” rule
  - take Median (first, middle, last element elements).

Comparison Sorts

- So far, all these sorting techniques use "pair-wise comparisons"
- Can we do better?
  - Better than $O(n \log_2 n)$
  - Better than $O(n^2)$
- How about $O(n)$?

Radix Sort

- create groups using the “least significant key”
- regroup using a more significant key
- repeat process until sorted
- Analysis
  - efficient for large number of elements
  - independent of original ordering
  - average case is O(n)

The Magic of RadixSort

- Input list:
  126, 328, 636, 341, 416, 131, 328

- BucketSort on lower digit:
  341, 131, 126, 636, 416, 328, 328

- BucketSort result on next-higher digit:
  416, 126, 328, 328, 131, 636, 341

- BucketSort that result on highest digit:
  126, 131, 328, 328, 341, 416, 636

Bucket Sort

- Create an array of “buckets”
- One bucket for each unique value
  - Not good for large number of keys
- Sorting a deck of cards
  - How many buckets?
- Sorting exams by grade
  - How many buckets?

Sorting in Java

- The Arrays Class in Java has a method sort()

  ```
  public static void sort(Object[] items)
  ```

- Implements a modified merge sort in O(n log n)
- Uses the Comparable Interface

Big-Oh Comparison

	best	average	worst
selection	O(n2)	O(n2)	O(n2)
bubble	O(n2)	O(n2)	O(n2)
insertion	O(n)	O(n2)	O(n2)
merge	O(n log2 n)	O(n log2 n)	O(n log2 n)
quick	O(n log2 n)	O(n log2 n)	O(n2)
radix	O(n)	O(n)	O(n)

Sorting in real time

size	O(n2)	O(n log2 n)
10	93 millisecs	356 millisecs
100	8.46 secs	3.62 secs
1,000	13.91 min	36.91 secs
10,000	23.15 hrs	5.93 min
100,000	96.45 days	1 hr
1,000,000	26.41 years	10.23 hrs

Demos on different sorts

Let's see a visual representation of the most common sorting algorithms:

http://www.sorting-algorithms.com/

Sorting and Efficiency

- Efficiency depends on
  - sorting algorithm
  - initial order of the data
  - number of elements
  - swaps are more costly than comparisons

Stable Sorts

- Sorting algorithms in which two elements with the same values maintain their same relative order through execution.

before

			x			y		

after

	x	y						

Stable Sorts

- Stable sorts are:
  - Insertion
  - Bubble (exchange)
  - Merge
  - Radix
- Non-stable sorts are:
  - Selection
  - Quick

What else?

- So far, we have discussed algorithmic solutions
- How about data structures?
  - Heap Sort
  - Tree Sort

Readings

Watch the following in YouTube

- ***Sorting Algorithms - YouTube***

- Read the pdf doc on sorting

Homework

- Quiz #7 (stacks & queues) tomorrow
- No homework due next week!