# Week 1

Dynamic connectivity

We assume “is connected to” is an equivalence relationship

Reflexive, Symmetric and Transitive

Connected components:

Maximal set of components that are mutually connected

Quick-Find

Create an array of N objects. Assign each of them with an id. As connections are added, the id of one connected component assumes the id of the other component connecting to it.

Union is expensive; takes quadratic time (N array accesses)

Quick-Union

Here id points to the root of the node.

Find is expensive; could be N array access

Improvement

* Keep track of size of each tree; balance by linking root of smaller tree to root of larger tree

Find takes proportional to depth of tree (at most log N); Union takes constant time

* Path compression: after computing the root, set the id of each examined node to point to that root.

Weighted union with path compression (M unions with N objects):

It takes (N + M.log N) time to execute.

Percolation Phase Transition

Given ‘p’ site vacancy probability, when N is large, there is a sharp threshold p\* such that for p > p\*, it almost certainly percolates.

Common order of growth classifications



Binary Search

Running time: O (log N) to search in a sorted array of size N.

Notation

Big Theta: Classify algorithms

Big Oh: Develop upper bounds

Big Omega: Develop lower bounds

# Week 2

Stack: Examine the most recently added, LIFO

Queue: Examine the least recently added, FIFO

Bag: Adding items to a collection and iterating (when order doesn’t matter)

Resizing array:

Grow -Double size when full

Shrink -Halve size when it is quarter full

Implement stack or queue using linked list or resizing array.

Auto-boxing:

Automatic cast between a built-in (primitive type) and its wrapper class

[JAVA] Iterable has a method that returns an Iterator.

[JAVA] Iterator has methods hasNext and next.

Post-fix or Reverse Polish Notation

Dijkstra's two-stack (one for operands and another for operators) algorithm computes the same value if the operator occurs after the two values

Sorting

**Insertion.sort**

The client passes array of objects to sort function. The sort function calls back the object’s compareTo method as needed.

public interface Comparable <Item> {

public int compareTo (Item that);

}

Total Order

It is a binary relation “<=” that satisfies:

Anti-symmetry, Transitivity & Totality

**Examples**: Standard order, Chronological order, and Alphabetical order

Selection Sort

* In iteration ‘i’, find index min of the smallest remaining entry
* Swap a [i] and a [min].
* Running time: O (n2).
* Order of n2 comparisons and n exchanges
* Same running time irrespective of the input
* Data movement is minimal

Insertion Sort

* In iteration ‘i’, swap a[i] with each larger entry to its left
* Best case: n-1 comparisons and 0 exchanges
* Worst case: Order of n2 comparisons and n2 exchanges

Inversion

It is a pair of keys that are out of order. For partially sorted array, the insertion sort runs in linear time. Number of exchanges equals the number of inversions.

Shell sorting

Move entries more than one position at a time by h-sorting the array. An h-sorted array is h interleaved sorted subsequences.

**Algorithm**: h-sort array for decreasing sequence of values of h. Insertion sort, with stride length h.

A g-sorted array remains g-sorted after h-sorting it.

Sequence to use:

* 3.x + 1
* 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905

Shuffle sort

Knuth shuffle

* In iteration ‘i’, pick integer r between 0 and i uniformly at random
* Swap a [i] and a [r]

Shuffling is a hard problem.

Convex Hull

Convex hull of a set of N points is the smallest perimeter fence enclosing the points. The output is a sequence of vertices in a counter-clockwise order.

**Application**:

Shortest path that avoids the obstacle

Find a pair of points with the largest Euclidean distance between them

**Algorithm**:

Chose a point ‘p’ with smallest y coordinate

Consider points in order of polar angle; discard unless it creates a counter-clockwise turn. Signed area of the triangle formed by the three points determines if the points are in clockwise or anti-clockwise order.

# Week 3

Merge sort

* Divide an array into two halves
* Recursively sort each half
* Merge two halves

[JAVA] assert statement – throws exception if statement is not true

[JAVA] java -ea / -da (enable or disable asserts at runtime)

**Enhancements**:

* Use insertion sort for small sub-arrays (< 7)
* Switch the role of input and auxiliary array in each recursive call
* Stop if already sorted

Bottom-up Merge Sort

* Merge sub-arrays of size 2, 4, 8, 16, etc.
* Non-recursive version, but 10% slower

Comparison

[JAVA] Comparable interface used for natural order

[JAVA] Comparator interface sort using an alternate order

[JAVA] Use Object instead of Comparable

Decouples the definition of a data type from the definition of what it means to compare two objects of that type.

Stability

A stable sort preserves the relative order of items with equal keys

Insertion sort (equal items never move past each other) and merge sort (takes from left sub-array if keys are equal – preserves the relative order) are stable.

Selection sort and shell sort are not stable (long distance exchanges)

First sort on one column and then sort by other category

Quick Sort

* Shuffle the array
* Partition such that for some ‘j’
  + Entry a[j] is in place
  + No larger entry to the left of j
  + No smaller entry to the right of j
* Sort each piece recursively

In-place sorting algorithm; not stable

**Improvements**

For small sub-arrays, use insertion sort

Estimate partitioning element in the middle: sample 3 values and take the median.

Quick-select

Partition, and repeat in one sub-array depending on ‘j’

Running time: O (n)

Duplicate keys

**3-Way Partitioning**

Partition array into three parts - less than, greater than and equal to the pivot value