Data Structures

Stacks: a list of elements, the main operations are to add and remove items to the front of the list, burying old items in the stack.

Advantage: Simple FIFO

Disadvantage: O(n) searches, narrow operation

Queues: a list of elements, the main operations are to add items to the front of the list and remove items from the end of the list

Advantage: Simple LIFO

Disadvantage: O(n) searches, narrow operation

Binary Search Tree: A tree in which each node branches out to more nodes, one with a key less than it and one with a key more (if the nodes exist).

Advantage: Fast lookup, fast insert

Disadvantages: Need comparable, can become unbalanced

So use AVL, rotate unbalanced parts

Heaps: A tree that just contains nodes and guaranteed log(n) level. In a Min-Heap, the parent of a node has a key smaller than the children, so the root now has the minimum key. In a Max Heap the opposite is true and the root has the maximum.

Advantage: Really good in finding the min and the maximum, reasonable log(n) guarantees. Can be used as a priority queue

Disadvantages: Hard to iterate over

Union Find (Disjoint Sets, up tree): Each node relays information for which group it belongs to (by tracing parentage). Can be optimized to provide better guarantees.

Advantage: Great for organizing group memberships and measuring group size

Disadvantages:

Minimum spanning tree: Union of the elements in a graph using the least cost links between them

Hash Tables: Stores values (or key/value pairs) in an array with built-in hashed (pseudorandom) assignments.

Advantage: Really fast (constant) lookups and deletions

Disadvantages: Cannot iterate over

Adjacency Lists: Array with a cell for each vertex, each cell contains a list of vertices that it is connected to.

Advantage: Great for getting a list of adjacent vertices, for sparse graphs

Disadvantages: Bad for dense graphs, looking up specific edges.

Adjacency Matrices: A two-dimensional array (number of vertices by number of vertices) each entry indicates whether there is an edge between two points and can also indicate the weight of the edge.

Advantage: Great for dense graphs, looking up specific edges

Disadvantages: Large space requirements, wasted in sparse graphs

Website with Time Complexities:
http://bigocheatsheet.com/

Abstraction

Protect the code!
Copy In, Copy Out
Mutability
Deep Copying

Map / Reduce

Fork / Join
Fork splits up processes to different processors or threads
Join combines the split up data
Map / Reduce
Map does a process on many parts of data independently, such as multiplying each value in an array by two
Reduce
Amdahl's Law
One cannot speed up a program more than a degree based on how much runtime cannot be parallelized

Sorting

What sorts do you have?
Insertion Sort: Scan through an array from the beginning to end, if you find an element smaller than your end, swap it down the list until it has been INSERTED into the right place.
Selection Sort: SELECT the smallest element in the array by scanning all of the elements and swap it to the front, then scan through the rest and SELECT the next smallest, putting it into the next bin, etc.
Quicksort: Choose a pivot point and put all elements smaller than it on one side of the array and all elements larger on the other side, recursively divide the left and right sides. It is QUICK because it can be done in place and works out well when you pick the median as a pivot.
Heapsort: Build a HEAP of the elements and then pop the min elements out repeatedly, filling in a sorted array. Alternatively, you can fill in from the back with the max elements (this allows you to do it in place)
Mergesort: Divide the array into two partitions of equal size and mergesort these partitions. Return single element arrays. Once you get back your two partitions, MERGE them together but combining each of the elements.
Bucket Sort: Have K bins for each possible key. Insert elements at their keys. O(n + k) but k may be awful if you choose the wrong set

Radix Sort: Divide the elements into bins based on the first digit, then in this order divide elements into bins based on the next digit and so forth… The complexity of this is O(nk) but if all of the elements are unique, then k >= log(n) so it really is not better than Quick, Heap, and Merge sort.

Quicksort Code

```
import java.io.*;
import java.util.*;

public class QuickSort {

   public static void Quicksort(int[] A) {
      Quicksort(A, 0, A.length – 1);
   }

   public static void Quicksort(int[] A, int left, int right) {
      int pivot_index = partition(A, left, right);
      Quicksort(A, left, pivot_index);
      Quicksort(A, pivot_index+1, right);
   }

   public static int partition(int[] A, int left, int right) {
      int pivot = A[left];
      while (left < right) {
         while (A[left] <= pivot) left++;
         while (A[right] > pivot) right--;
         int temp = A[x];
         A[x] = A[y];
         A[y] = temp;
      }
      return left;
   }
}
```