- NOTE: for each API that requires a specific Node as input, think of it as two calls, call() and call(Node)

APIs

Generalized List

```
void        addFront(Item item)
void        addBack(Item item)
Item        deleteFront()
Item        deleteBack()
void        delete(Item item)
void        add(int i, Item item)
Item        delete(int i)
boolean     contains(Item item)
boolean     isEmpty()
int         size()
```

Bag

```
void        add(Item item)
boolean     isEmpty()
int         size()
```

Queue

```
void        enqueue(Item item)
Item        dequeue()
boolean     isEmpty()
int         size()
```

Generalized Queue (similar to List)

```
boolean     isEmpty()
int         size()
void insert(Item x)
Item delete(int k)
```

Set

```
void        add(Key key)
void        delete(Key key)
boolean     contains(Key key)
boolean     isEmpty()
int         size()
```

Mathematical set operations

```
Set<Key>    complement() -> all keys not in set
void        union(Set<Key> a)
void        intersection(Set<Key> a)
```

Stack (LIFO/Pushdown)

```
void        push(Item item)
Item        pop()
boolean     isEmpty()
int         size()
```

Deque

```
boolean     isEmpty()
int         size()
void        pushLeft(Item item)
void        pushRight(Item item)
Item        popLeft()
Item        popRight()
```

Generalized Buffer

```
void        insert(Item item)
Item        delete()
void        left(int k)
void        right(int k)
int         size()
```

Stopwatch

```
void        start()
double      elapsedTime()
```

Maximum Priority Queue (or min)

```
void        insert(Key k)
boolean     contains(Value v)
Key         max() / min()
Key         deleteMax() / deleteMin()
boolean     isEmpty()
int         size()
```

Heaps

Symbol Table

```
void        put(Key key, Value val)
Value       get(Key key)
void        delete(Key key)
boolean     contains(Key key)
boolean     isEmpty()
int         size()
(keys)      keys()
(values)    values()

// additional
Key         min()
Key         max()
Key         floor(Key key)
Key         ceiling(Key key)
int         rank(Key key)
Key         select(int k)
void        deleteMin()
void        deleteMax()
int         size(Key low, Key hi)
(keys)      keys(Key low, Key hi)
(keys)      keys()
```

Search Tree (as symbol table)

```
int         size()
int         size(Node x)
Value       get(Key key)
Value       get(Node x, Key key)
void        put(Key key, Value value)
void        put(Node x, Key key, Value value)

// additional
Key         select(int k)
Node        select(Node x, int k)
int         rank(Key key)
int         rank(Node x, Key key)
void        deleteMin()
Node        deleteMin(Node x)
void        deleteMax()
Node        deleteMax(Node x)
void        delete(Key key)
Node        delete(Node x, Key key)

(keys)      keys()
(keys)      keys(Key low, Key hi)
void        keys(Node x, Queue<Key> queue, Key low, Key hi)
```

Hash Symbol Table (in addition to above)

```
int         hash(Key key)

// additional
void        resize()  // for linear probing
```

Undirected Graph

```
int         numVertices()
int         numEdges()
void        addEdge(int v, int w)
(vertices)  adjacentVertices(int v)

// additional
int         degree(Graph G, int v)
int         maxDegree(Graph G)
int         avgDegree(Graph G)
int         numberOfSelfLoops(Graph G)

// find connections (dfs/bfs)
            Search(Graph G, int s)
```

```
void        search(Graph G, int v) (dfs/bfs)
boolean     marked(int v)
int         count()

// find paths
            Paths(Graph G, int s)
boolean     hasPathTo(int v)
(ints)      pathTo(int v)

// connections
            ConnectedComponent(Graph G)
boolean     connected(int v, int w)
int         count() // number of connected components)
int         id(int v) // component id
```

Symbol Graph (degrees of separation)

Directed Graphs

```
// same as undirected graph with additional
Digraph     reverse()

// cycles
boolean     hasCycle()
(ints)      cycle()
```

Topological Sort

- NOTE: a graph must be directed and acyclic to have a topological sort order (DAG)

Strong Connectivity of Graphs

Edge (for edge-weighted graph and minimum spanning tree)

```
double      weight()
int         either()
int         other(int v)
int         compareTo(Edge that)
```

Edge-weighted graph (as above)

MST (Prim, Kruskal)

Shortest Path (edge-weighted (acyclic) digraph)

- Dijkstra
- Acyclic
- Bellman-Ford

Strings

```
copy
length
compare
concatenate
indexing
substrings

// sorts
key-indexed counting for sorting
LSD (least significant digit) string sort
MSD (most significant digit) string sort

insertion-sort
quicksort
three-way quicksort
three-way string quicksort
mergesort
```

Tries (uses same API a search trees above)

- a search tree built from the characters of the search string as keys

Ternary Search Trees

String symbol table types

- BST
- 2-3/red-black
- linear probing

- trie (r-way trie)
- trie search (TST)

Search

- key-indexed search
- sequential search
- binary search
- radix search
- external searching

Substring search

- brute force
- Knuth-Morris-Pratt (relate to DFAs)
- Boyer-Moore
- Rabin-Karp fingerprint search

Regular Expressions

- NOTE: uses NFAs and digraphs

Data Compression

- (bit operations)
- run-length encoding
- bitmaps
- Huffman compression
- LZW compression
- LZ77

Context

- event-driven simulation
- collisions
- B-trees
- B-tree set
- suffix arrays
- network flow
- Ford-Fulkerson max flow

- Reductions (shortest path and max flow)

NP-complete problems

- Boolean satisfiability
- integer linear programming
- load balancing
- vertex cover
- Hamiltonian path
- Protein folding
- Ising model
- Risk portfolio for a return

Sorting

- stable-sort
- selection
- insertion
- bubble sort
- shellsort
- mergesort
- quicksort (fastest general purpose)
    - 3-way quicksort/3-way radix quicksort
    - partitioning & stack size
    - recursive & non-recursive
    - binary quicksort
- heapsort
- priority queue sort
- radix sort
    - MSD sort
    - LSD sort
- special purpose sorts
    - Batcher's odd-even mergesort
    - shuffle/unshuffle

    - network sorts
    - split-interleave merging
    - external sorting
    - 3-way merge
    - block sort
- (timsort)
- (comb sort)
- (counting sort)
- (radix sort)
- (shuffling and Fisher-Yates shuffle)
- array sorting
- index and pointer sorting
- in-place sorting
- linked-list sorting
- sorting by key-indexed counting

Hashing

- hash functions
    - modular hash functions for integers
    - modular hash functions for characters
    - hash functions for string keys
        - character strings
        - universal hash function
- separate chaining
- linear probing
- double hashing
- dynamic hash tables

Numerical Methods

- polynomial interpolation
- least-squares estimation
- solution of equations

Encryption

- DES
- RSA

Geometric

- line segment intersections, points, etc
- convex hulls
- arc length on spherical surfaces

Graph Algorithms

- properties and representations
- graph search
    - mazes
    - dfs
        - dfs algorithms
    - separability & biconnectivity
    - bfs
    - generalized search
    - backtracking
    - (Bloom filter)
- digraphs and dags
    - search
    - reachability and transitive closure
    - equivalence relations and partial orders
    - topological sorting
    - strong components
- minimum spanning trees
    - Prim
    - Kruskal
    - Boruvka
    - Euclidean MST

- shortest paths
    - Dijkstra
    - all-pairs shortest path
    - acyclic networks
    - Euclidean networks
    - reduction
    - negative weights
- network flow
    - flow networks
    - augmenting path maxflow
    - preflow-push maxflow
    - maxflow reductions
    - mincost flows
    - network simplex algorithm
    - mincost-flow reductions
- coloring

Representations

Lists (stack, queue, bag, etc)

- linked list
    - circular
    - doubly-linked
    - singly-linked
    - head only
    - dummy head
    - head, dummy tail
    - head, tail
    - dummy head, dummy tail
    - no container, only head node reference

- statically sized array
- dynamically sized array
- array with pointers/references to nodes (review this in C++ algs)

Sets

- lists (all types)
- BST
- B-tree
- hash table (for unordered)

Symbol Table

- associative arrays
- search tree
- hash table
- symbol graphs

Graphs

- list of sets
- list of lists
- adjacency map/dictionary/table with edge weights
- map/dictionary/table with adjacency sets
- adjacency matrix (with arrays/lists, rows & columns represent vertices)
- incidence matrix (rows represent vertices, columns represent edges)
- weight matrix (using infinity for missing edges)
- union find-style (an array of connected components with the index of the component base)
- lists of edges (pairs representing an edge or an "Edge" type)
- NOTE: adjacency lists work well for sparse graphs, matrices work well for dense graphs

Trees

- list of lists
- nodes with children and/or parent
  - children can be pointers/references as members
  - children can be list of pointers/references
- binary string

- string of parens
- (connected acyclic graph)

list types

- skip list
- doubly connected list
- (more)

Tree types

- general tree
- rooted trees
- ordered trees
- binary trees
- m-ary trees
- free trees
- BST
    - randomized
    - splay
    - top-down 2-3-4
    - red-black
    - skip-lists
- binary search tree
- 2-3 trees
- red-black trees
- AVL trees
- B-trees
- digital search trees
- tries
    - Patricia tries
    - multiway tries

    - TSTs
- heaps
- (R-tree, R* tree, R+ tree)
- (hash tree?)
- (left child, right sibling)
- (multiway trees)
- (space-partitioning trees)

Graph types

- sparse
- dense
- directed
- undirected
- cyclic
- acyclic
- connected
- disconnected
- (relationship to trees)
- NOTE:
    - see https://en.wikipedia.org/wiki/List_of_data_structures
    - see https://en.wikipedia.org/wiki/List_of_algorithms

Later

- simple custom memory management when given a block of memory using:
    - block partition pools
    - Arenas
    - Spans?
    - general partitioning
- Descriptions of simplified dynamic memory management techniques are in Modern Compiler Design and TAOC v1