Matthew Tran
CS61B 1167

- Big Theta (Θ)
  - $k_1 \cdot f(N) \le R(N) \le k_2 \cdot f(N)$
    - exist $k_1 > 0, k_2 > 0$
    - R(N) in same Θ(f(N))
  - "equal"
- Big O
  - "less than or equal"
  - $R(N) \le k_2 \cdot f(N)$
- WeightedQuickUnionDS
  - array, initially -1
  - keep track tree size at root, -size
  - link root of smaller tree to big tree
  - connect O(log N)
  - isConnect O(log N)
    - [illegible] if same root
  - path compression
    - when finding root, set everything to root
    - "amortized constant time"
      - O(N + M · log* N)
      - 1st [illegible] expensive but cheaper over time
      - log* - iterative log, to increase by 1, need to be 2^n power [illegible]
  - not BST, can have multiple at root
  - Weighted QU
    - allow heavy tree attach to lighter
- Binary Search Tree (BST)
  - rooted binary tree w/ BST property
    - left smaller, right bigger
  - Search - O(log N)
    - [illegible] is optimal
  - Inserting - O(log N)
  - Deletion - O(log N)
    - leaf - just remove (0 children)
  - depth - root 0, 1st child 1, etc "# links"
  - height - depth of deepest leaf
    - determine worst case runtime
    - math proofs 4, 311 - ln N
  - average depth - avg depth of node
    - determine average runtime to find node
    - 2 ln N
- B Tree ("Splitting Tree")
  - "stuff" nodes so always [illegible] height
    - too difficult, give middle to parent
    - chain reaction, root splits if needed
    - L = max items in node (2-3 has 3 [illegible] 3 children)
  - Invariants
    - always bushy, might vary in height
    - all leaves equidistant from root
    - nonleaf node, k items, k+1 children
  - height
    - best $\log_{L+1}(N)$
    - worst $\log_2(N)$
    - Θ(log N)
  - find/contains
    - worst case # [illegible] nodes - HH
    - worst case # items - L
    - O(HL) = O(L log N) = O(log N)
  - add
    - O(log N)
  - Tree Rotation
    - 2n-6 rotations to one tree to another
    - right rot - node x is its rightchild of its left child
    - left rot - node x its leftchild of rightchild
- Left Leaning Red Black Tree (LLRB)
  - "glue links" - small item to left, red
  - black links connect 2 node to 3 node
  - height is just # [illegible] edges
  - Searching - like BST
  - every path from root to leaf same black links
  - determine valid LLRB - draw equivalent 2-3 Tree
  - insertion - always use red link
    - left leaning violation - "right leaning 3 node" - rotate left
    - incorrect 4 node violation - "2 consec left links" - rotate right
    - Temporary 4 node - "2 red children" - color flip, make above red, lower black
  - height O(log N)
  - contains O(log N)
  - insert O(log N)

- Big Strats
  - nested loops
    - picture, exact sum, examples
  - Binary Search O(log N)
  - Selection Sort - $O(N^2)$
  - Merge Sort O(N log N)
    - split in 2, merge N
- Hash Table
  - buckets be collisions
  - Θ(Q) - worst case longest list, Q gives O(N)
  - use modulus of # buckets for fewer buckets
  - keep N/M as O(1), increase buckets
    - M is # of buckets, N total # bucket load
    - N/M ?= 1.5, load factor, 2x M
  - postively hashcode
  - don't store stuff if hashcode can change
  - O(1) everything, except resize
  - use prime base to avoid overflow small
- Priority Queue (PQ)
  - Heap
    - add O(log N)
    - getSmallest O(1)
    - removeSmallest O(log N)
    - props
      - min-heap - every node smaller
      - complete - all nodes far left
    - getSmallest - return root
    - add
      - insert bottom
      - swim up
    - remove
      - remove root
      - move bottom node to top
      - sink down, swap smaller child up
    - array rep
      - 2k+1, 2k+2
      - (k-1)/2
- Hash Map / Tries
  - string keyed map
  - each node stores letter
  - node points to other nodes
  - runtime
    - # of keys
      - add - Θ(1)
      - contains - Θ(1)
    - length of key, L
      - add Θ(L)
      - contains Θ(L)
  - true if end of string
  - good for prefixes
  - put(key, value)
    - if put same key, replace value
  - Careful when modifying stuff already inside
  - Big O, if limit buckets
    - get - beginning or end, needs search
    - put - add beginning or end, still iterate
  - "don't assume anything about data of keys"
    - all can be in one bucket
- can rotate even if 2 children

- QuadTree
  - NW, NE, SW, SE [illegible]
  - insert - add to node
  - Range Search (points in box)
    - start at biggest [illegible]
    - add points
    - terminate when null subspace
- Kd tree
  - each 2 space
    - part x
    - depth 2 - Y
    - depth 3 - x, etc
  - nearest
    - start root, check better subspace if can be better
  - pruning
- Tree Traversal
  - BFS
    - level order
  - DFS
    - preorder - root, left, right
    - inorder - left, root, right
    - postorder - left, right, root
- Graphs
  - s-t connectivity
    - mark node, if s≠t return true
    - check all neighbors
    - DFS
  - Shortest path
    - queue w/ s
    - while !q.empty
      - remove v from queue
      - for unmarked neighbors of v
        - mark n
        - set edgeTo[n] = v
        - add n to queue
    - //use BFS, check 1st neighbors 1st
  - Graph Representation
    - #1 - adjacency matrix
    - #2 - HashSet<Edge>
    - #3** - adjacency lists
      - print all Θ(V+E)
- Dijkstra's Algorithm
  - pseudocode
    - PQ.add(source, 0)
    - for all v: PQ.add(v, ∞)
    - while PQ.notEmpty
      - p = PQ.removeSmallest
      - relax all edges from p
    - Relax(p, q) //edge
      - if distTo(p) + w < distTo(q)
        - distTo(q) = distTo(p) + w
        - edgeTo(q) = p
        - PQ.changePriority(q, distTo(q))
    - //basically makes changes if better
  - Invariants
    - edgeTo(v) is best known predecessor
    - distTo(v) is best known total d
  - O(E log V)
- A*
  - Dijkstra's not efficient b/c search radius
  - visit order d(source, v) + h(v, goal)
    - h is heuristic, guess of better
    - if constant, become Dijkstra's

- Delegation - uses something
- TreeMap(K, V) is in java.util
- careful w/ compile stuff, if remove stuff, is "this" still it?
- Ω Big omega - best case
  - Big O - ≤
  - Big Θ - equality

- Cycle Detection
  - O(V+E) if just check all marks neighbors
  - O(V + E log* V) if use WeightedQuickUnionUF (w QUUF)
- Minimum Spanning Tree
  - subgraph - connected, acyclic, includes all vertices
  - min total weight
  - Cut Property
    - cut - assign nodes to 2 non-empty sets
    - crossing edge connects sets
    - min crossing edge in MST
    - Proof by Contradiction
  - Prim's Algorithm (like Dijkstra's)
    - pick node add to PQ, add all others ∞
    - while PQ not empty:
      - remove closest v in PQ
      - relax all edges
        - //but distTo is just distance of edge, not to tree
        - //to tree and add edges
      - //vertex removal is like adding an edge
    - O(E log V)
  - Kruskal's Algorithm
    - sort edges
    - add in order of increasing weight if no cycle made
      - //use WQUPC
    - until V-1 edges total
    - O(E log E) / O(E log* V) //O(E log* V) if presorted
      - sorting
- Big O chart
  - E > V
    - Dijkstra's - O(E log V)
    - Prim's - O(E log V)
    - Kruskal's - O(E log E)
      - sorted - O(E log* V)
  - s-t paths DFS - O(V+E) time, Θ(V) space
  - s-t paths BFS - O(V+E) time, Θ(V) space (stack)
  - PQ (for Dijkstra)
    - add O(V log V)
    - removeSmallest O(V log V)
    - changePriority O(E log V)
  - A* - depends on h()
  - adjacency matrix s-t paths - $O(V^2)$ time Θ(V) space
  - Tries - Θ(1) get, Θ(1) add
  - Heap
    - add - O(log N)
    - getSmall - Θ(1)
    - removeSmall - Θ(log N)
  - LLRB / 2-3 Tree
    - add Θ(log N), contains Θ(log N)

- Big O, don't forget to multiply if calling func N times
- assume string hashCode is Θ(N) b/c goes thru all letters

- Read the question fully bro

Matthew Tran
CS61B 1167 pg.2

- Topological Sorting
  - need Directed Acyclic Graph (DAG)
  - algorithm
    - DFS on every indegree 0, don't clear markings between traversals
    - record postorder (only add if not marked)
    - topo order given by reverse of list
    - like to be merge backward
    - if don't have to start from indegree 0
  - all arrows point right basically
- DAG w/ negative edges SPT
  - Dijkstra - DFS or fails w/ stricter [illegible]
  - algorithm
    - find topo order $\Theta(V+E)$
    - visit in topo order
      - relax all edges
    - it works bc nothing earlier can be better, keep going downstream
  - it works once edges actually connect the
- Longest Paths Algorithm (LPT)
  - flip all weights negative, run DAG SPT, flip
- Sorting
  - Law of Trichotomy - $a<b$, $a=b$, or $a>b$
  - Law of Transitivity - $a<b$ & $b<c \Rightarrow a<c$
  - inversion - all possible pairs, less if out of order
    - want to reduce to 0 [illegible] possible
- Selection Sort
  - find smallest, swap to front
  - $\Theta(N^2)$ time
  - always looks at everything again
- Heapsort
  - Naive Heapsort
    - make separate heap
    - $\Theta(N\log N)$ time
    - $\Theta(N)$ memory
  - In-place Heapsort
    - heapification (bottom up)
      - sink nodes in reverse level order
      - make a max heap
    - loop N times
      - remove largest (max) item, swap with last item in heap, and sink
    - slowly reduce heap and leave sorted array behind
    - $\Theta(N\log N)$ time, $\Theta(1)$ memory
- Mergesort
  - split in half, merge
  - $\Theta(N\log N)$ time, $\Theta(N)$ memory
  - faster than heapsort
- Insertion Sort
  - swap until right place
  - $\Omega(N)$, $O(N^2)$ time
  - swap # = # inversions ($\Theta(N+K)$)
  - good for almost sorted
  - $\Theta(N+K)$ time, K = # inversions
  - $N \le 15$ empirically fastest sort
  - Shell's Sort (extra) - composition - [illegible]
- Quick Sort (Partition Sort)
  - pivot, move items less to left, more to right
  - recursively sort left and right
  - empirically fastest in most cases, $2N\ln N$
  - $\Omega(N\log N)$, $O(N^2)$ if bad pivot 0
  - avoiding worst case
    - sorted or all duplicates
    - fixes
      - use median (optimal)
      - random swap(?)
      - shuffle
      - partition from center (not always good)
  - Tony Hoare's In-place Partitioning
    - two pointers walk toward each other
    - left likes smaller, right likes big, swap
    - very fast quicksort
    - not stable tho

- Non-comparison Based Sorting
  - Sleep Sort (lmao)
  - Counting Sort
    - exploit space, sort by keys
    - place items directly in location
    - keys 0 to N-1, $\Theta(N)$
    - algorithm
      - count how many of each
      - make starting points
      - add stuff in, move along pointers to avoid duplicates
    - stable, but need to know all possible
    - $\Theta(N+R)$, alphabet - R
      - want $N \ge R$ for good performance
  - LSD Radix Sort
    - sort each digit right to left
    - treat empty space as 0
    - $\Theta(WN+WR)$
      - W - width of items
      - N - # items
      - R - size of alphabet
  - MSD Radix Sort
    - sort left to right
    - sort sub problems separately
    - best case - $\Theta(N+R)$ (all unique)
    - worst case $\Theta(WN+WR)$
    - bad caching, like heapsort
  - Radix sort vs Comparison Sort
    - strings and integers only
    - Intuitive Approach
      - Mergesort
        - $\Theta(N\log N)$ compares
        - $\Theta(WN\log N)$ for length W
        - good for very diff strings
      - Radix Sort
        - $\Theta(WN)$
        - good for large, similar strings
    - Mergesort Empirically Faster bc of JIT (just in time) compiler which optimizes code as it runs
    - Radix Sort Integers
      - get ith digit or mask to string
      - change base to speed up
        - too big - too much memory
        - too small - too slow
        - base 256 pretty good

- Compression
  - Prefix Free Codes
    - use fewer bits to encode same words
    - no code word is prefix of another, like a tree, chars are leaves
    - Shannon Fano Code ([illegible])
      - count relative frequencies, in order of freq
      - left half leading 0, right half leading 1
      - not optimal
  - Huffman Coding
    - count frequencies node for each, merge two smallest nodes into supernode, keep going
    - implementation
      - encoding - arrays faster, but potential memory waste
      - decoding - tries

- Avoiding Worst Case Quicksort
  - Philosophies
    - Randomness (preferred)
      - random pivot / shuffle elements
    - Smart Pivot - close to median
      - optimal but algorithm too slow
      - BFPRT (aka PICK) $\Theta(N)$
      - partitioning to find median, ends at n/2, best algorithm, [illegible] $\Theta(N)$
      - why tho
    - Introspection - switch to safer sort if too deep
    - Preprocess Array - check if quicksort would be slow (hard)

- Huffman Philosophies
  - one encoding per file type
    - faster to prebuilt code
    - suboptimal
  - unique code for each file, send with
    - slower, have to use space for bit stream
- Other encoding mechanisms
  - Run-length encoding
  - LZW: search for patterns
  - exploit redundancy
- Comparing compression algorithms
  - honest, fair comparison - send decompressed w/ compressed
  - Kolmogorov complexity - impossible to write, shortest bit stream that outputs
  - Space-bounded compression - impossible can find kolmogorov
  - Space/time bounded compression
    - $O(T \cdot 2^S)$ works but slow af.
- P=NP
  - P - efficiently solvable problems
  - NP - efficiently verifiable solutions for problems
  - most say no
  - all math proofs are NP, so if P=NP, then can prove anything easily
- Tony Hoare Partitioning
  - left right pointers
  - when stop at something don't like, then swap
  - stop when pointers cross
  - basically when hits equal pivot, constantly swap it around
  - after cross, then swap w/ pivot
    - for test, usually use 1st as pivot
    - ignore pivot while swapping until end
- Exam Tips
  - don't forget all duplicates
  - careful w/ encapsulation
  - worst case compare to cost $\frac{n(n+1)}{2}$
    - help $n+2+\dots+1$
  - worst case [illegible] $n+n-1$
  - bottom up heapification $n-1$, $n+1$

- More on Sorting
  - stability, duplicates in order
  - optimization
    - <15 items switch to insertion sort
    - adaptive, exploit existing order
      - insertion, smoothsort, timsort
    - exploit restrictions on set of keys
  - java Arrays.sort
    - quicksort on primitive
    - mergesort (timsort) of objects
  - N log N best one can do
    - N log N = log(N!) asymptotically
    - for sequential comparisons only

- Midterm 1 Stuff
  - javac + java + class + javadoc + object
  - class - type checked, [illegible], faster
  - javac - if [illegible], will also recompile
  - need main() for run natively
  - vocab
    - instance var - property of instance
    - constructor - determines how to construct class
    - non-static/instance method - need instance
    - declaration of variable - claims a name
    - instantiation of class - object - need memory
    - assignment
    - invocation - call method
  - static vs. non-static
    - even if an instance and static var, still one static var, will always run [illegible]
    - static invoke using class name
    - static can't access instance vars
    - use "this" to refer to self
  - public static void main (String[] args) {}
    - args has arguments from CLI
  - 8 primitives
    - byte, short, int, long, float, double, boolean, char
  - Golden Rule of Equals (GRoE)
    - y = x copies bits over, always
      - bit in object copies address technically
    - == just checks the bits
  - Class Initialization
    - instance var - allocate memory, set to 0, null
  - public vs private
    - private class - [illegible]
  - invariant - guaranteed to be true
  - Generic Types
    - ex. SLList<T>, T is the [illegible]
    - java uses <>, can give type [illegible]
  - SLList - singly linked list
  - DLList - doubly linked list
  - AList
    - get really fast for arrays
    - addLast - next item inserted at size
    - System.arraycopy(source, sourceInd, target, targetInd, L)
    - multiplicative resize
    - usage ratio = size/arr.length
      - halve if ratio < 0.25
    - java won't know to cast, can't make generic array
    - null deleted items
  - Testing
    - ad hoc testing - not recommended
    - org.junit.Assert.* - better
      - import org.junit.Test; @Test
      - import static org.junit.Assert.*;
      - TDD - test driven development, test first
  - overloading - don't do
  - is-a relationship - inheritance
    - interface - no private allowed
      - list method names
      - can implement if want (default)
      - can multiple interfaces
  - dynamic method selection
    - looks for subclass method w/ same signature as static
  - inheritance - extends
    - use @Override
    - use super to call super method
    - always runs default no args constructor if u don't specify
    - use casting
      - static type temporarily changed
    - don't use hiding
  - compareTo()
    - usually compareTo(Object o), but use implements Comparable<object type> to avoid casting
    - compare(T i, T j) // comparator interface
      - useful for implementing different compare types
  - List - java interface
  - Sets - no duplicates, no order
  - iterators
    - orig class - implements Iterable<T> - allows foreach loop to identify and use
      - iterator() method that returns Iterator
      - that class implements Iterator<T>
        - hasNext();
        - next();

- toString()
  - called implicitly when printing
  - default [illegible]
  - StringBuilder alt faster
- boolean equals(Object o)
- static methods don't get the generic, which makes sense
  - to do public static <T> returns [illegible]
  - put generic before return type