RB-TREES AVL-TREES

insert, insert-fixup, delete, delete-fixup are O(lg(n)), roatation is O(1) Insertion Alg:

Max rotations after insert: 2 ; max rotations after delete 3 let w be the newly inserted node.

Insertion Alg: (1) Perform standard BST insert for w

p = parent; gp = grandparent; x is the newly inserted node. (2) Starting from w, travel up and find the first unbalanced node. Let z be the unbalanced node and y be

(1) Perform standard BST insert & make x red the child node of z on the path from w to z, and x be the grandchild of z on the path from w to z.

(2) If x is root, make x black (3) 4 cases from here:

(3) Do the following if x’s p is NOT black or x is NOT the root (a) y is left child of z and x is left child of y (left left case)

(a) If x’s uncle is red -> right rotate z

(i) Set color of p & uncle as black (b) y is left child of z and x is right child of y (left right case)

(ii) Set color of gp as red -> left rotate y -> right rotate z

(iii) Let x = x’s grp. Repeat (2) & (3) (c) y is right child of z and x is right child of y (right right case)

(b) if x’s uncle is black -> left rotate z

(i) p is left child of gp & x is left child of p (d) y is right child of z and x is left child of y (Right Left Case)

-> right rotate gp -> swap colors of p & gp -> right rotate y -> left rotate z

(ii) p is left child of gp & x is right child of p

-> left rotate p -> apply (i) just above

(iii) p is right child of gp & x is right child of p

-> left rotate gp -> swap color of p & gp

(iv) p is right child of gp & x is left child of p

-> right rotate p -> apply (iii) just above

BINOMIAL HEAPS

Insert takes O(lg n) but amortized is O(1)

Find-min takes O(lg n) but O(1) if you keep min pointer

Delete-min is O(lg n)

Merge O(lg n)

Props:

(1) Each binomial tree in a heap obeys min-heap ordering

(2) There can only be either ONE or ZERO binomial trees

for each order, including zero order

\* (1) implies a B.H. w/ n nodes consists of lg(n) + 1 bin. trees

\* (2) implies each binomial tree w/ n nodes corresponds to

one digit in a binary representation of the number n.

Eg. 13 = (1101)­2 = 2^3 + 2^2 + 2^0

DISJOINT SETS

make-set & find-set: constant

GRAPHS

undirected, simple: there is at most one edge between any two vertices.

the degree of undirected, simple: the total number of edges emanating from

the vertex.

undirected, simple, connected: a **path** exists between every pair of vertices

undirected, simple, regular: All vertices have the same degree.

undirected, simple, complete: there exists an edge between every pair of verts.

undirected, simple, connected, acyclic: the vertex count is one greater than the edge

count.

planar: it can be drawn in a plane with no crossed edges.

Walk: a sequence of vertices with an edge between vi and vi+1

path: A path is a walk that does not include any vertex twice, except that

its first vertex might be the same as its last.

trail: A trail is a walk that does not pass over the same edge twice. A trail

might visit the same vertex twice, but only if it comes and goes from a

different edge each time.

cycle: A cycle is a path that begins and ends on the same vertex.

circuit: A circuit is a trail that begins and ends on the same vertex.

euler trail: A trail which involves every edge exactly once.

Hamiltonian path: A path which involves every vertex.

example of an in-between 2/3 case:

T(n) = 2T(n/2) + nlog(n)

Log base 2 of 2 is 1, and nlog(n) is always going to be more than n^1, so we can't use case 2.

By case 3, f(n) dominates over recursion. We can see the relationship between the two by putting them into the ratio:

             f(n)/n^(log base b of a)

If case 3 is satisfied, the ratio will grow to infinity and will be asymptotically larger than n^ε for some ε > 0. Remember that for some function f(x) to be asymptotically larger than g(x), lim(x→∞) f(x)/g(x) = ∞

For nlog(n)/n^1 = log(n), there is no possible value for ε that will satisfy lim(n→∞) log(n)/n^ε = ∞, so log(n) is not asymptotically larger. There is no ε that will ensure that the rate of growth for log(n) is faster than n^ε as n grows to infinity. So we can't use case 2 or case 3 - it's in-between.

Conversely:

T(n) = 3T(n/4) + nlog(n)

Log base 4 of 3= .792. The ratio nlogn/n^.792 grows to infinity. It's asymptotically larger than n^ε for some ε > 0.

lim(n→∞) (nlogn/n^.792)/n^ε = ∞

lim(n→∞) nlogn/(n^.792 \* n^ε) = ∞

lim(n→∞) nlogn/(n^.792+ε) = ∞

We know that nlogn > n^1, so as long as our denominator is less than that, this ratio will grow to infinity. So here, any 0 < ε < .208 will work and therefore nlogn/n^.792 will be asymptotically larger than n^ε and case 3 holds.

**BST: Terms: Full** each node has either 2 or no children. **Perfect** a full tree with all the leaves at same level. **Complete** like perfect except lowest level can be piled to the left. **Balanced** longest & shortest path equal. **Root** no parent. **Leaf** a node with no children. **Parent** other than root every node has one. **Child** other than leaves each node has one.

**RB:** **Prop: 1.** Every node is either red or blk. **2.** The root is blk. **3.** Every leaf (NIL) is black. **4.** No red node has a red parent. **5.** All nodes have a consistent black height (all paths from a node to the reachable leaves have the same # of black nodes.

**Insertion:** Takes 0(logn) time. Max # of recolors 0(logn). Max 2 rotations. **Deletion:** Takes 0(logn) time. Max # recolors is 0(nlogn). Max 3 rotations

**Terms: Uncle** sibling of parent **Niece** closest child of sibling - if you are a right child, your niece is a right child - if you are a left child, your niece is a left child **Nephew** furthest child of sibling - if you are a right child, your nephew is a left child - if you are a left child, your nephew is a right child **Black height** the number of black nodes encountered on the way to a leaf - sometimes abbreviated BH **Linear** true if the parent and child are both left children or are both right children

**Questions:** Consider a black interior node *n* in a red-black tree and any path from *n* to a leaf. Which of the following is a constraint on these trees, where *R* is the number of red nodes and *B* is the number of black nodes? **R ≤ B + 1** Consider a red node *n* in a red-black tree and the length of the shortest possible path from *n* to a leaf, *S*, and the length of the longest possible path from *n* to a leaf, *L*. Which of the following is a constraint on these trees? **L = 2S + 1** Consider a node with 2 children in a RB tree. If one of the children is a leaf, what is the most number of decendants the other children can have? **6** Inserting 7 consecutive # in order yields how many single/double rotations? **3 & 0**

**AVL: Prop: 1.** Leaves have a balance factor of 0. **2.** Interior nodes have a balance factor of -1, 0, or 1.

**Insertion:** takes0(logn) time. # of rotations is 0(1). **Deletion:** takes0(logn) time. # of rotations is 0(logn).

**Terms: Favorite** the child who roots the taller subtree - if the parent's balance factor is 1, the left child is the favorite -- if the parent's balance factor is -1, the right child is the favorite -- if the parent's balance factor is 0, no child is the favorite **Linear** true if the parent and child are both left children or are both right children. **Rotation** make a child a parent and the former parent a child.

**Questions:** Consider an AVL tree holding the numbers 0-9, what is the largest # that can be the root? **5** Consider a non root node x with a single child in an AVL tree. Must that node x have a sibling and, if so, what is the most number of descendants the sibling of x can have? **6** Consider a node with 2 children in an AVL tree. If one of the children is a leaf, what is the most # of decendants the other child can have? **2**

**Disjoint:** **Make-set** is Constant **Find-Set** is Constant **Union** is linear **Total Work no pref/smaller bcom rep** is quadratic **Total Work larger bcom rep** is log linear;

**Disjoint Tree:** **Make-set** is constant **Findset** is linear **Findset (w/path or union)** is log **Total Work(nothing)** quad **Total Work(path or union)** is loglinear **Total work(both)** is linear **Path**: speed up union and findset **Union** speed up findset

**Graphs:** **Walk** sequence of edges such that ei and ei+1 have no common vertex **Trail** walk with no edge appearing more than once **Path** a trail with no vertex appearing more than once **Euler trail** trail which involves every edges once (longest trail in undirected graph) **Hamiltonian path** path which involves every vertex(longest path in undirected graph); \*\*Longest path in undirected gr bounded by minimum degree of graph.

**Binomial Heap: Runtime: Find-min** is Θ(1) **Extract-min/Insert/decrease key** is Θ(logn) **Union** Θ(n)

**Properties:** Binomial heap H is an ordered list of binomial trees that satisfy **1.** Each binomial tree is heap-ordered, ie. For each node its key is greater or equal to the key of its parent. **2.** There is at most one binomial tree in H whose root has a given degree. **3.** Binomial trees are contained in the list in the order of increasing degrees.

**Fibonacci Heap: Runtime Find-min** is Θ(1) **Extract-min** is Θ(logn) **Insert/decrease key/Union** is Θ(1)

**Sorting:** Find the most extreme value in the unsorted portion and place it at the boundary of the sorted and unsorted portions? **Selection Sort** Sort the lower half of the items to be sorted, then sort the upper half, then arrange things such that the largest item in the lower half is less than or equal to the smallest item in the upper half ? **Mergesort** Take the first value in the unsorted portion and place it where it belongs in the sorted portion? **Insertion sort** Pick a value and arrange things such that the largest item in the lower portion is less than or equal to the value and that the smallest item in the upper portion is greater than or equal to the value, then sort the lower portion, then sort the upper ? **Quicksort** Which sort optimizes the worst case behavior of bubble sort? **Selection Sort** What is the best/worst time case complexity for mergesort? **nlogn** If quicksort is implemented such that the pivot is chosen to be the first element in the array, the worst case behavior of the sort is: **Quadratic** Best case quicksort? **nlogn** Best case selection sort? **linear** Worst case Classical selection sort? **Quadratic** Best case insertion sort? **Linear** Worst case classical in sertion sort? **Quadratic** Extract min



