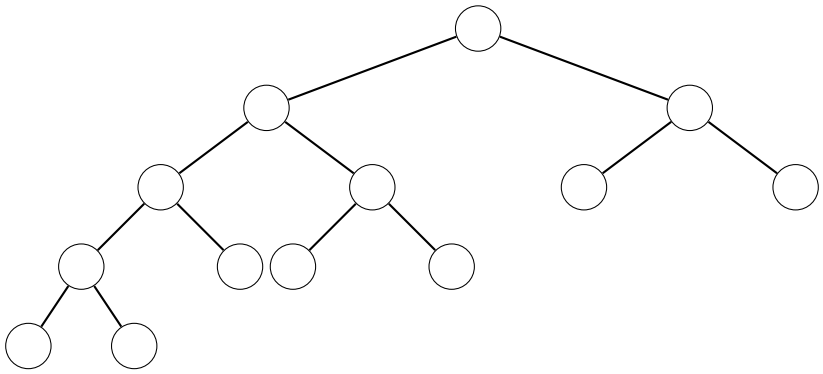
CS161 Homework 3 **Due:** 29 April 2016, 12 noon
Submit on Gradescope **Handed out:** 22 April 2016

Instructions: Please answer the following questions to the best of your ability. If you are asked to design an algorithm, please describe it clearly and concisely, prove its correctness, and analyze its running time. If you are asked to show your work, please include relevant calculations for deriving your answer. If you are asked to explain your answer, give a short ($\sim$ 1 sentence) intuitive description of your answer. If you are asked to prove a result, please write a complete proof at the level of detail and rigor expected in prior CS Theory classes (i.e. 103). When writing proofs, please strive for clarity and brevity (in that order). Cite any sources you reference.

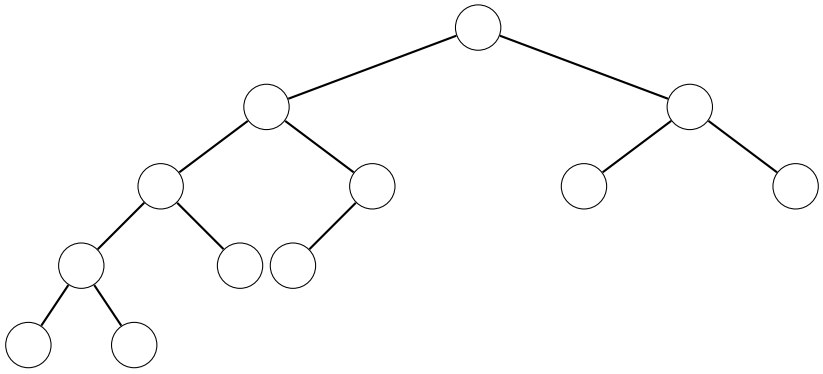
1 (12 points) Red-Black trees and Red-Purple trees

It's Barr the Bear's birthday! To celebrate, he decides to save the planet ... by planting trees! In particular, he wants to plant Red-Black trees (RB trees) in his garden. Unfortunately, Barr the Bear's birthday coincides with Prank Day celebrated by the penguins, and Poe the Penguin has been up to some mischief. He "pranks" the RB trees into Red-Purple trees (RP trees) by altering the structure of the trees in some way.

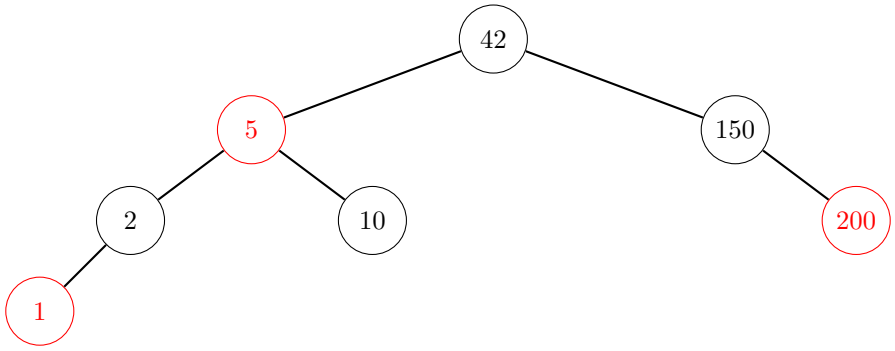
(a) If you believe a tree is a valid RB tree, provide a valid coloring for the tree – to convince Barr to keep it. Otherwise, concisely prove why there is no valid coloring – to convince Barr to destroy it.



(b) If you believe a tree is a valid RB tree, provide a valid coloring for the tree – to convince Barr to keep it. Otherwise, concisely prove why there is no valid coloring – to convince Barr to destroy it.



(c) After removing the RP trees, Barr now wants to upgrade his RB tree shown below by inserting the number 161 into it. Show how this can be done such that after the insertion it remains to be a valid RB tree. For full-credit, you should show intermediate steps (i.e., by showing how the tree is being transformed into the final one) by drawing a few trees.



2 (14 points) Ternary Search Trees on 2D Plane

In this problem, we will be talking about ternary search trees (TST). Ternary search trees are similar to binary search trees but rather than having just 2 pointers (left and right), they have 3 pointers (left, middle, and right).

A TST T is a ternary tree in which each node contains a point of the form (x, y) for some $x, y \in \mathbf{Z}$. Moreover, no two points in T can share the same x value and no two points can share the same y value.

Then the following properties hold for every node u with key (x, y) in any TST:

- Any point (x_L, y_L) in the left-subtree of u has $x_L < x$ and $y_L < y$.
- Any point (x_M, y_M) in the middle-subtree of u has $x_M > x$ and $y_M < y$.
- Any point (x_R, y_R) in the right-subtree of u has $x_R > x$ and $y_R > y$.

Clarification: You can assume that for any node v in TST T, you can compute the number of nodes that belong to the subtree of v in time $O(1)$ (e.g., it is stored as part of TST).

(a) Prove or disprove: For any set of n distinct points (where no two points share the same x-coordinate or y-coordinate), there exists a ternary search tree T on these n points with $h(T) = O(f(n))$:

(i) when $f(n) = n$,
(ii) when $f(n) = \log n$.

To prove the claim, provide a concise proof. To disprove the claim, provide a counterexample.

(b) Suppose that you are given a point (x', y') and a TST T which contains n points and has height h. You wish to determine whether (x', y') is contained in T or not.

Design an efficient algorithm to solve this problem. Prove the correctness of the algorithm. Analyze its running time in terms of n and also in terms of h.

(c) Now, suppose that given a TST T on n nodes and height h and a point (x', y'), you wish to determine the number of points (x, y) in the tree such that $x' \leq x$ and $y \leq y'$. Give a recursive algorithm that searches for these points, starting from the root of T. At each recursive step at a node u of T, how many children of u does your algorithm need to recurse on in the worst case? Analyze the (*worst case*) runtime $U(h)$ of your algorithm in terms of h. What is the (*worst case*) running time in terms of n (when h can be arbitrary)? What would the running time be in terms of n if the tree is completely balanced and $h = \log_3 n$?

(d) *Bonus:* (4 points) Show that your running time $U(h)$ above is tight by providing an example tree and input point that cause the algorithm to use at least $\Omega(U(h))$ time.

3 (12 points) Hash Table

Consider a hash table with n buckets and n elements hashed into it. We will show that with high probability, the size of the largest bucket is going to be very small.

We will assume that the hash function being considered satisfies the simple uniform hashing property. That is, the probability that an element is inserted into any of the n buckets is $1/n$ and independent of the elements already there.

(a) Let b_i be the number of elements in the i-th bucket. Then b_i follows a binomial distribution. What are the parameters of this distribution? What is $E[b_i]$?

(b) Show that $\lim_{n\to\infty} \Pr(\max_i b_i \geq \ln n) = 0$. In other words, for sufficiently large n, we can be assured that with high probability, none of the buckets will have more than $\ln n$ elements hashed to them. You may find the following inequalities helpful:

The **Chernoff Bound**: If X is a binomially distributed random variable with $E[X] = \mu$, then for any $\delta > 0$, $\Pr(X > (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$. (Correction: See Piazza Post 324.)

The **Union Bound**: Given n events $A_1, A_2, \ldots A_n$, we have $\Pr(\cup_{i=1}^n A_i) \leq \sum_{i=1}^n \Pr(A_i)$. In another words, the probability of at least one of the n events occurring is at most the total sum of the individual event probabilities.

4 (16 points) Hash Table: Simple Uniform Hashing

Poe the Penguin understands and likes hash tables. Barr the Bear understands hash tables but does not like chaining. Help Barr choose the size of his hash table such that he would not need to worry about chaining. Let n be the number of items that Barr has and m be the size of his hash table to be determined. We assume n is sufficiently large and m is much larger than n. Assume Barr's hash function satisfies the simple uniform hashing property.

(a) What is the probability that a particular slot is empty after hashing n items? Derive a lower bound in the form of $c_0 + c_1\frac{n}{m}$ where c_0 and c_1 are constants. *Hint: Take the two most significant terms in the binomial expansion* $(x+y)^n = \sum_{j=0}^n \binom{n}{j} x^{n-j}y^j$.

(b) What is the probability that a particular slot has exactly one item? Derive a lower bound in the form of $\frac{f_1(n)}{m} + \frac{f_2(n)}{m^2}$ where f_1 and f_2 are polynomial functions in n.

(c) Derive an upper bound on the probability that there exists a slot with at least two items. Use the bounds from Parts (a) - (b).

(d) Let $\epsilon > 0$ be given. Using Part (c), determine m in terms of n and ϵ, such that the probability that all slots have at most one item (no chaining!) is at least $1 - 1/n^\epsilon$. Note: m should be in the form $n^{g(\epsilon)}$ where g is a linear function in ϵ.