CMSC 420:Fall 2019 Dave Mount

Midterm Exam 2

This exam is closed-book and closed-notes. You may use one sheet of notes (front and back). Write all answers in the exam booklet. You may use any algorithms or results given in class. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

Problem 1. (30 points) Short answer questions. Except where noted, explanations are not required but may be given for partial credit.

(1.1) (8 pts) In an open addressing hashing system using **quadratic probing**, an insertion operation is guaranteed to **succeed** under which of the following conditions (select one):

(a) The table has at least one empty slot
(b) The table has at least one empty slot, and the table size m is a prime number
(c) The table is less than half full ($\lambda < \frac{1}{2}$)
(d) The table is less than half full ($\lambda < \frac{1}{2}$), and the table size m is a prime number
(e) Quadratic probing might fail under any circumstances

(1.2) (12 pts) Recall that a rehashing system is defined by two parameters $\lambda_{\min}$ and $\lambda_{\max}$, which limit the table's load factor. The list on the left below shows three objectives that a good rehashing system should have, and the list on the right shows choices made in the design of these parameters. For each objective (a)–(c), indicate which parameter choice (1)–(6) most directly **guarantees that this objective is achieved**:

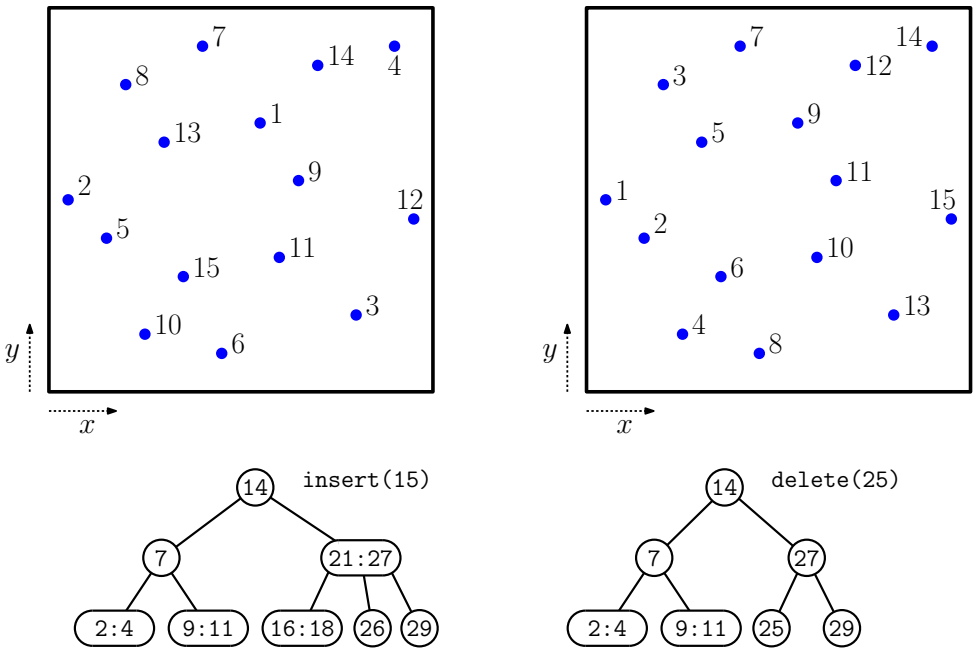
	Objectives
(a)	High space utilization (most of the table is used)
(b)	Fast search times
(c)	Rebuilding is not done too often

	Parameter choices
(1)	$\lambda_{\min}$ should not be too low
(2)	$\lambda_{\min}$ should not be too high
(3)	$\lambda_{\max}$ should not be too low
(4)	$\lambda_{\max}$ should not be too high
(5)	$\lambda_{\max} - \lambda_{\min}$ should not be too low
(6)	$\lambda_{\max} - \lambda_{\min}$ should not be too high

(1.3) (5 pts) We have n uniformly distributed points in the unit square, with no duplicate x- or y-coordinates. Suppose we insert these points into a kd-tree in *random* order (see the figure below left). As in class, we assume that the cutting dimension alternates between x and y. As a function of n what is the expected height of the tree? (No explanation needed.)

(1.4) (5 pts) Same as the previous problem, but suppose that we insert points in *ascending* order of x-coordinates, but the y-coordinates are *random* (see the above figure right). What is the expected height of the tree? (No explanation needed.)

Problem 2. (20 points) Consider the B-trees of order 3 shown in the figure below. Let us assume two conventions: key rotation (when possible) has precedence over splitting/merging. Second, when splitting a node, if the number of keys shared by the two new nodes is an odd number, the leftmost node receives the larger number of keys.



(2.1) (10 pts) Show the B-tree that results after inserting the key 15 into the tree on the left. (Intermediate results are not required, but may be given to help assigning partial credit.)

(2.2) (10 pts) Show the B-tree that results after deleting the key 25 from the tree on the right.

Problem 3. (20 points) In scapegoat trees, we showed that if `size(u.child)`/`size(u)` $\leq \frac{2}{3}$ for every node of a tree, then the tree's height is at most $\log_{3/2} n$. In this problem, we will generalize this condition to:

$$\frac{\texttt{size(u.child)}}{\texttt{size(u)}} \leq \alpha, \qquad (*)$$

for some constant α.

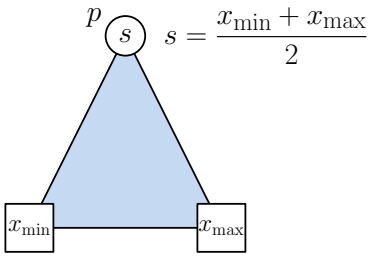
(3.1) (8 pts) Why does it **not** make sense to set α larger than 1 or smaller than $\frac{1}{2}$?

(3.2) (4 pts) If every node of an n-node tree satisfies condition $(*)$ above, what can be said about the height of the tree as a function of n and α?

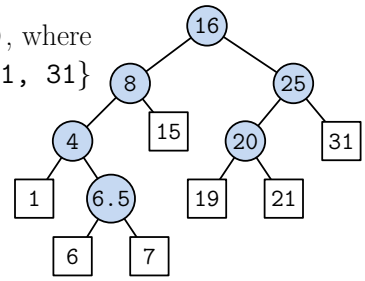
(3.3) (8 pts) Briefly justify your answer to (3.2).

Problem 4. (15 points) We say that an extended binary search tree is *geometrically-balanced* if the splitter value stored in each internal node `p` is midway between the smallest and largest keys of its external nodes. More formally, if the smallest external node in the subtree rooted at `p` has the value $x_{\min}$ and the largest external node has the value $x_{\max}$, then `p`'s splitter is $(x_{\min} + x_{\max})/2$ (see the figure below).

Given a sorted array $A[0 \ldots n-1]$ containing $n \geq 1$ numeric keys, present pseudo-code for a function that builds a geometrically-balanced extended binary search tree, whose external nodes are the elements of A. **Convention:** If a key is *equal* to an internal node's splitter value, then the key is stored in the *left subtree*.

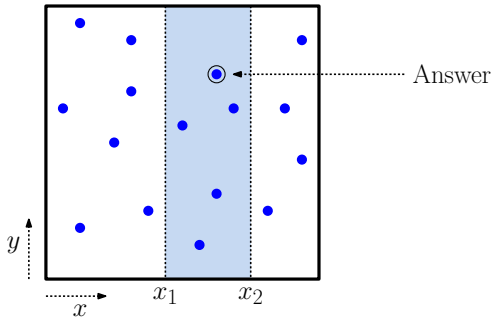


`buildTree(A)`, where
`A = {1, 6, 7, 15, 19, 21, 31}`



Briefly explain any assumptions you make about underlying primitive operations (e.g., constructors for your internals and external nodes). Any running time is okay.

Problem 5. (15 points) Given a set P of n points in the real plane, a *partial-range max query* is given two x-coordinates x_1 and x_2, and the problem is to find the point $p \in P$ that lies in the vertical strip bounded by x_1 and x_2 (that is, $x_1 \leq p.x \leq x_2$) and has the maximum y-coordinate (see the figure below).



Present pseudo-code for an efficient algorithm to solve partial-range max queries, assuming that the points are stored in a kd-tree. You may make use of any primitive operations on points and rectangles (but please explain them). You may assume that there are no duplicate coordinate values, and no coordinates are equal to x_1 and x_2. If you solve the problem recursively, indicate what the initial call is from the root level.