Final Exam

The exam is asynchronous and online. It is open-book, open-notes, open-Internet, but it must be done on your own without the aid of other people or software. The total point value is 120 points. Good luck!

Problem 1. (35 points) Short answer questions. Except where noted, explanations are not required but may be given for partial credit.

(1.1) (4 pts) In 2-3 tree deletion, under what circumstances do you perform a merge as opposed to adoption (key rotation)?

(1.2) (4 pts) Although AA trees are a variant of red-black trees, nodes are not colored. What is the condition that determines whether a node in an AA tree is red or black?

(1.3) (4 pts) You access an element in a splay tree, and then a few operations later you access the very same element. Which property of splay trees best explains why sequences of operations like this are processed efficiently? (Choose from: static optimality, static finger theorem, dynamic finger theorem, working set theorem, scanning theorem)

(1.4) (4 pts) What is the purpose of the *next-leaf* pointer in B+ trees?

(1.5) (4 pts) We claimed that scapegoat trees are efficient in the amortized sense, but `find` operations in scapegoat trees are efficient even in the worst case. Why is this?

(1.6) (10 pts) You are using hashing with open addressing. Suppose that the table has just one empty slot in it. In which of the following cases are you *guaranteed* to succeed in finding the empty slot? (Select all that apply.)

(a) Linear probing (under any circumstances)
(b) Quadratic probing (under any circumstances)
(c) Quadratic probing, where the table size m is a prime number
(d) Double hashing (under any circumstances)
(e) Double hashing, where the table size m and hash function $h(x)$ are relatively prime
(f) Double hashing, where the table size m and secondary hash function $g(x)$ are relatively prime

(1.7) (5 pts) In the unstructured memory management system described in Lecture 20, what was the purpose of the `size2` field at the end of each free block? (Why was it needed?)

Problem 2. (20 points) This problem involves the tree shown in Fig. 1.

(2.1) (4 points) List the nodes according to a *preorder traversal*

(2.2) (4 points) List the nodes according to an *inorder traversal*

(2.3) (4 points) List the nodes according to a *postorder traversal*

(2.4) (8 points) Show the final result of applying the operation `splay(8)` on this tree. (You need only show the final result. Intermediate results can be shown to help with partial credit.)

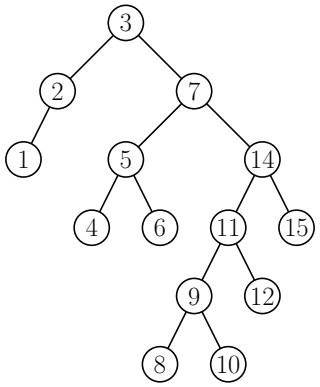

Figure 1: Tree traversal and splaying.

```
class TreapNode {
    Key key           // key
    int priority      // priority
    TreapNode left    // left child
    TreapNode right   // right child
}
```

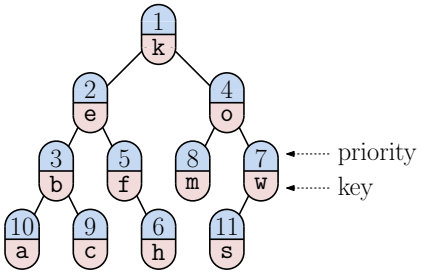

Figure 2: Treap node structure and an example.

Problem 3. (10 points) Suppose that you are given a treap data structure storing n keys. The node structure is shown in Fig. 2. You may assume that *all keys and all priorities are distinct.*

(3.1) (7 points) Present pseudocode for the operation `int minPriority(Key x0, Key x1)`, which is given two keys x_0 and x_1 (which may or may not be in the treap), and returns the lowest priority among all nodes whose keys x lie in the range $x_0 \leq x \leq x_1$. If the treap has no keys in this range, the function returns `Integer.MAX_VALUE`. Briefly explain why your function is correct.

For example, in Fig. 2 the query `minPriority("c", "g")` would return 2 from node `"e"`, since it is the lowest priority among all keys x where `"c"` $\leq x \leq$ `"g"`.

(3.2) (3 points) Assuming that the treap stores n keys and has height $O(\log n)$, what is the running time of your algorithm? (Briefly justify your answer.)

Problem 4. (15 points) In this problem we will build a suffix tree for $S =$ `aabbabaabbbaaba$`.

(4.1) (4 points) List the 16 substring identifiers for the 16 suffixes of S. For the sake of uniformity, list them in order (either back to front or front to back). For example, you could start with "`$`" and end with the substring identifier for the entire string.

(4.2) (8 points) Draw a picture of the suffix tree. For the sake of uniformity, when drawing your tree, use the convention of Fig. 7 in the Lecture 19 LaTeX lecture notes. In particular, label edges of the final tree with substrings, index the suffixes from 0 to 15, and order subtrees in ascending lexicographical order (`a` < `b` < `$`).

(4.3) (3 points) Draw the root and the first few levels of your suffix tree using the convention of Fig. 5 in the Lecture 19 LaTeX lecture notes. In particular, label each node with the index field and label each edge with a single character. (It is not necessary to show the entire tree, but you may. It suffices to show the root, its children, and its grandchildren.)

Problem 5. (15 points) Throughout this problem we are given a set $P = \{p_1, \ldots, p_n\}$ of n points in 2D space stored in a point kd-tree (see Fig. 3(a)).

(5.1) (10 points) In a *segment sliding*, you are given a vertical line segment, specified by its lower endpoint q and its height h (see Fig. 3(b)). The query returns the first point $p_i \in P$ that is first hit if we slide the segment to its right. If no point of P are hit, the query returns `null`.

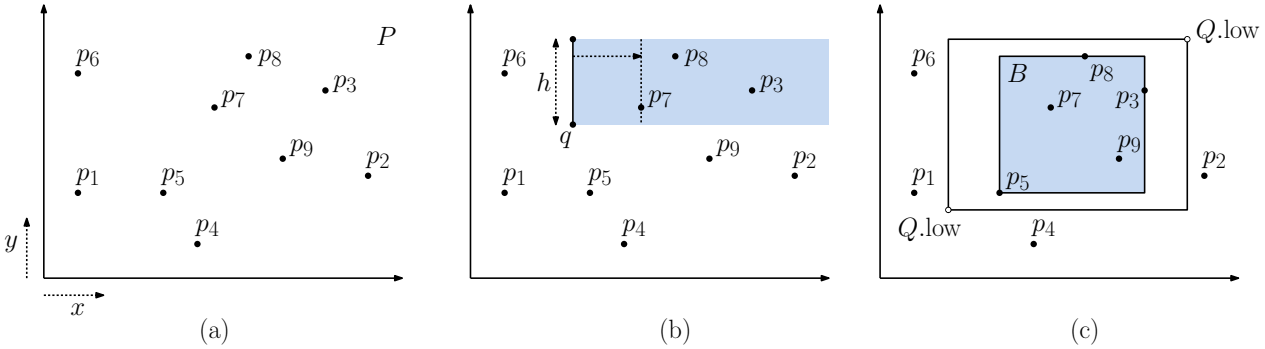

Figure 3: Segment-sliding and minimum box queries.

Give pseudo-code for an efficient algorithm, `Point segSlideRight(Point q, float h)`, which given the coordinates of the line segment, returns the answer to the segment-sliding query.

You may assume the standard kd-tree structure given in class, where each node stores a point `p.point`, a cutting dimension `p.cutDim`, and left and right child pointers `p.left` and `p.right`, respectively. You may make use of any primitive operations on points and rectangles. You may assume that there are no duplicate coordinate values among the points of P or the query point.

(5.2) (5 points) In a *minimum box query*, you are given an axis-aligned rectangle Q as input (given, say, by its corner points Q.low and Q.high), and the output is the smallest axis-aligned rectangle B that contains all the points of P that lie within Q (see Fig. 3(c)). If there are no points of P in Q, the query returns `null`. Present pseudo-code for an efficient algorithm, `Rectangle minBox(Rectangle Q)`, which given the query rectangle Q, returns the answer to the minimum box query. (You may reuse or modify your solution to (5.1).)

Problem 6. (25 points) In this problem, we will consider how to use/modify range trees to answer related queries. Given a set P of n points in d-dimensional space, recall that a range tree storing these points uses a total of $O(n \log^{d-1} n)$ storage. Given an axis-aligned rectangle R in d-dimensional space, in $O(\log^d n)$ time it is possible to identify a set of $O(\log^d n)$ subtrees in the range tree, such that the points lying within these subtrees form a partition of $P \cap R$.

In all cases, the input set P is a set of n points with positive coordinates. Note that the tree that you construct might be for a different set of points, and it may even be in a different dimension. In each case, describe the points that are stored in the range tree and how the search process works. Justify your algorithm's correctness and derive its running time.

(6.1) (10 points) In a *skewed interval min query* (SIM query), you are given two y-coordinates, y_0 and y_1, where $y_0 < y_1$. Among all the points of P whose y-coordinate lies within the interval $[y_0, y_1]$, the answer is the point $p_i = (x_i, y_i)$ that minimizes $x_i + y_i$. More visually, among all the points of P that lie within the horizontal strip $y_0 \le y \le y_1$, find the point that is first hit by a line of slope -1 sweeping up from the origin. If there are no points in the strip, the query returns the value `null`. If there are multiple points that satisfy the x-conditions and have the same y-coordinates, any of them may be returned. (In Fig. 4(a), the query returns point b.)

Your data structure should use $O(n \log n)$ storage and answer queries in $O(\log^2 n)$ time.

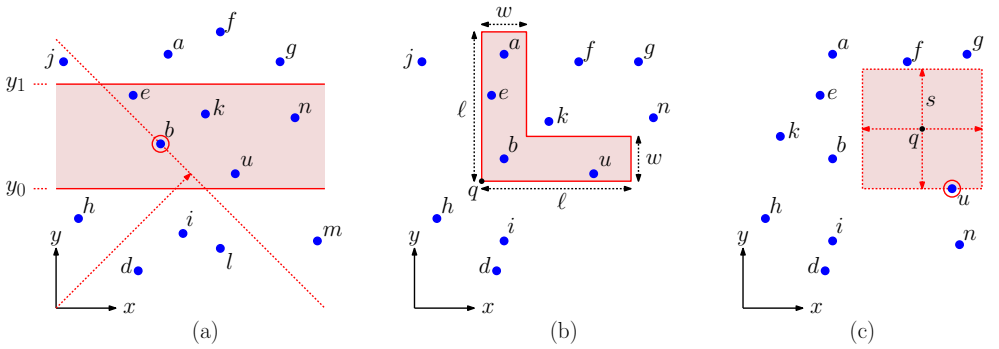

Figure 4: Using range trees to answer various queries.

(6.2) (5 points) An *L-shaped counting query* involves counting all the points lying within an L-shaped region. The query is described by three parameters, the lower-left corner q of the L-shape, the length ℓ which is the total height and width of the shape, and the width w of the segments of the L-shape (see Fig. 4(b)). The query returns a count of all the points that lie within this region.

Your data structure should use $O(n \log n)$ storage and answer queries in $O(\log^2 n)$ time.

(6.3) (10 points) A *maximum empty square query* is given a query point q, and it returns half the side length of the largest box centered at q that contains no point P in its interior. If q coincides with a point of P, the query returns 0. (In Fig. 4(c), the query returns the value s.)

Your data structure should use $O(n \log^2 n)$ storage and answer queries in $O(\log^3 n)$ time.