(ii) preorder traversal f c b a e d h g i

(iii) postorder traversal a b d e c g i h f

Do you notice an interesting property of the in-order traversal? What is it?

(b) Let a binary search tree be defined by the following class:

(Next Page !)

```
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
}
```

In class, we saw how to search for an element in a binary search tree. This question will demonstrate that binary search trees are more powerful. Describe an algorithm that calculates the k'th smallest element in the tree. (k, the input, is a number in $\{1, 2, \cdots, n\}$, where n is the number of nodes in the tree). You may find it helpful to modify the definition of IntTreeNode in order to accomplish this. Your algorithm should run in $O(h)$ time, where h is the height of the tree. An $O(n)$ solution will be given partial credit.

4. AVL Tree Implementation

(Next page !)

Write pseudocode for the AVL tree methods Balance, RotateLeft, and RotateRight. Assume that the rest of the data structure is implemented as in the Java code here `https://courses.cs.washington.edu/courses/cse373/18su/files/homework/AVLTree.java`.

You may use the skeleton of Balance from that code as a guide. Note you are not required to write your solution in Java, pseudocode is sufficient.

5. Hashing

Let the capacity of the hash table be 10 and the hash function be $h(x) = x$. Insert elements

$$42, 102, 12, 33, 25, 14, 62$$

(Next Page !)

to a hash table

(a) that uses linear probing

(b) that uses quadratic probing

Write down the total number of collisions and the hash table after all insertions in both cases. Why is the secondary clustering in quadratic probing less problematic than the primary clustering in linear probing (i.e. why are there fewer collisions)?

5. capacity is 10. $h(x) = x$. [42, 102, 12, 33, 25, 14, 62]

a). linear probing

0	1	2	3	4	5	6	7	8	9
		42	102	12	33	25	14	62	

Total number of collision: $\boxed{15}$

b). quadratic probing

0	1	2	3	4	5	6	7	8	9
	62	42	102	33	25	12		14	

42 % 10

102 % 10 + 1×1

12 % 10 + 2×2

33 % 10 + 1^2

25 % 10

14 % 10 + 2^2

62 % 10 + 3^2

Total number of collision: $\boxed{9}$

why?

For linear probing, when the collision happened, the table searched sequentially for an empty slot. It is accomplished using two value: ① initial value and ② an interval between successive values in modular arithmetic. The second value is repeatedly added to the initial value until a free space is found.

∴ it's new Location = (initial value + step size) % array capacity

$$H(K, i) = (H(K) + i) \bmod \boxed{n} \quad \text{for } i = 0, 1, 2, 3, \ldots n-1$$

For quadratic probing, the initial value is adding successive value of an arbitrary quadratic polynomial value. It skips regions in the table with possible clusters. So it gets wider space to put the value.

$$H(K, i) = (H(K) + i^2) \bmod m \quad \text{for } i = 0, 1^2, 2^2, 3^2, \ldots (m-1)$$

∴ $m > n$. the space is getting wider, the collisions is getting less. When it's full, resize the table.