Preorder. The first element given is the root. We can then search for this element in the inorder traversal. The elements of the tree are now uniquely partitioned to left & right subtrees by this element.

① the subsequence of elements preceding the root in the inorder sequence is the inorder sequence for the left subtree

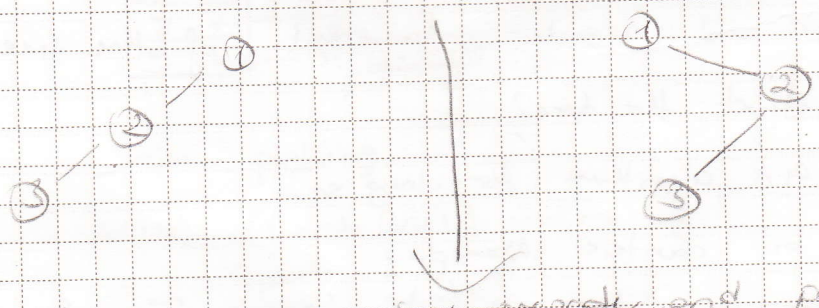
" succeeding " " right subtree

② We can recursively reconstruct left and right subtrees from the preorder and inorder traversals.

Example: Given: preorder and postorder ?

If we have left only or right only subtrees in the tree then the preorder and postorder traversals are not necessary unique.

Example: Preorder (1 2 3)
Postorder (3 2 1)



Same preorder and postorder we cannot reconstruct

Red Black Trees

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both of its children are black and its opposite
5. For each node, all the paths from the node to descendant leaves contain the same number of black nodes.

Black Height

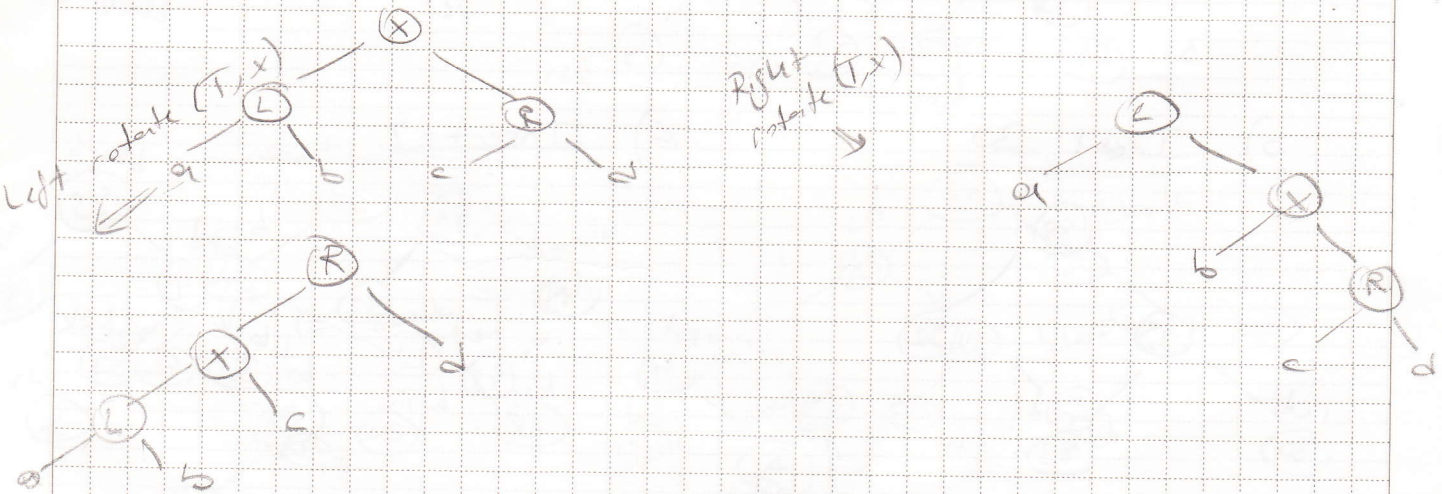
of black nodes on any path from (but not including) a node x down to a leaf denoted by bh(x).

The height of RB tree is at most $2\log(n+1)$ ← # of interval nodes

→ search
→ min
→ max
→ successor
→ predecessor

$O(h) = O(\log n)$

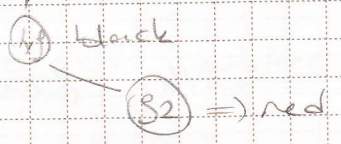
Rotation



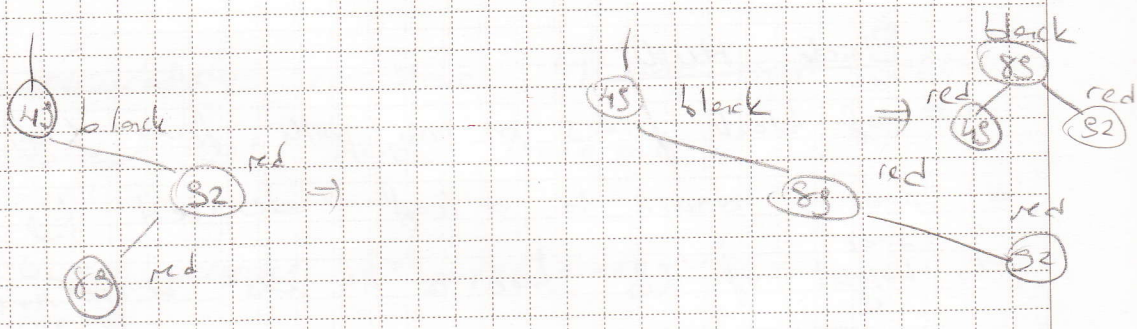
Example: 49, 32, 88, 83, 30, 2, 63, 15, 55, 32 ⇒ Red-black tree

1) Insert 49

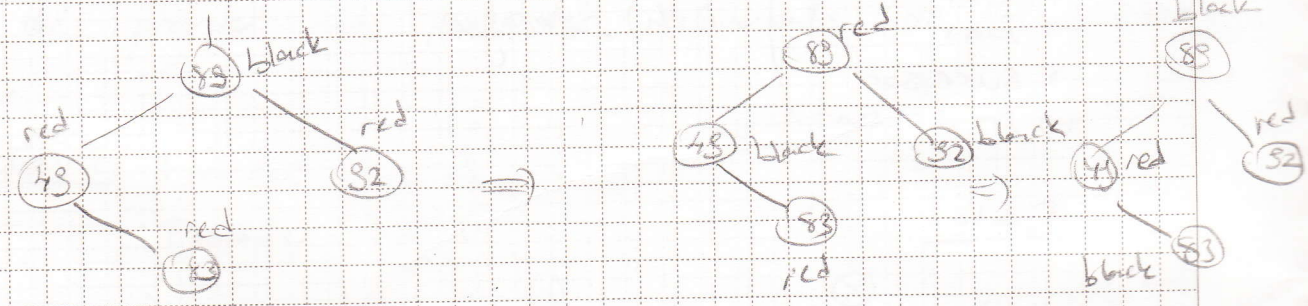
2) Insert 32



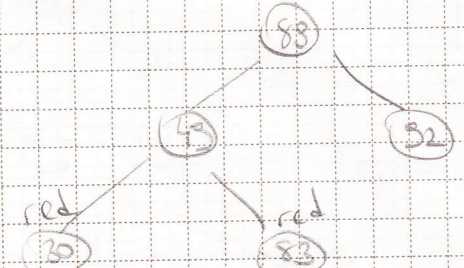
3) Insert 88



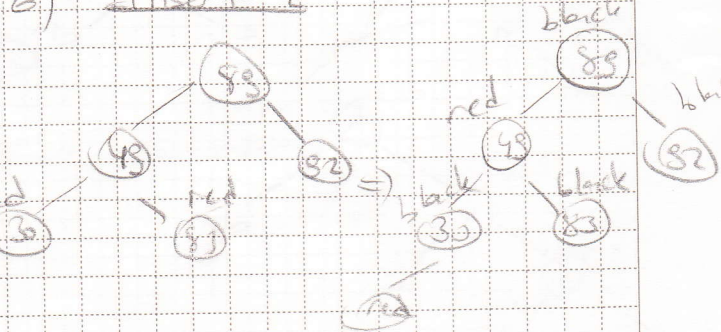
4) Insert 83



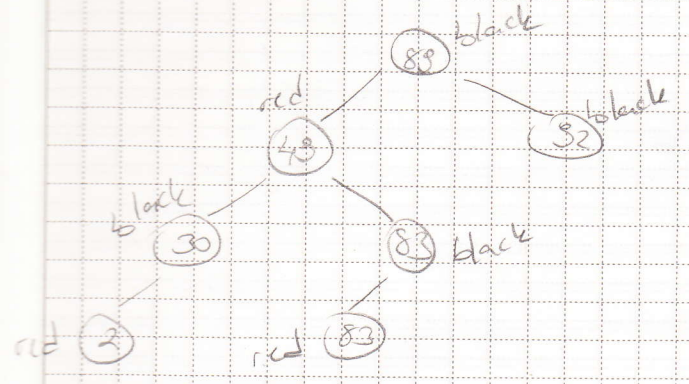
5) Insert 30



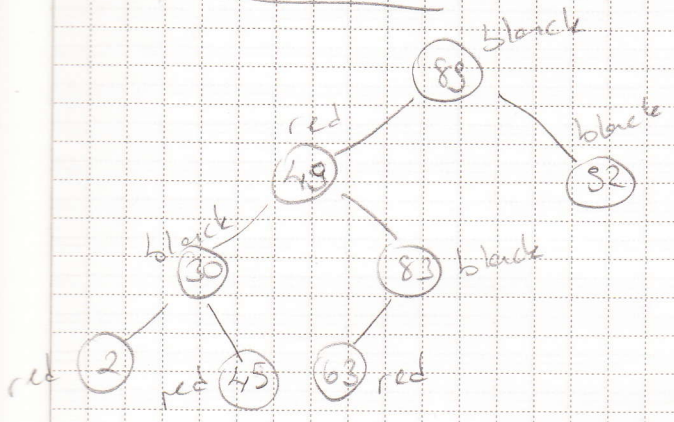
6) Insert 2



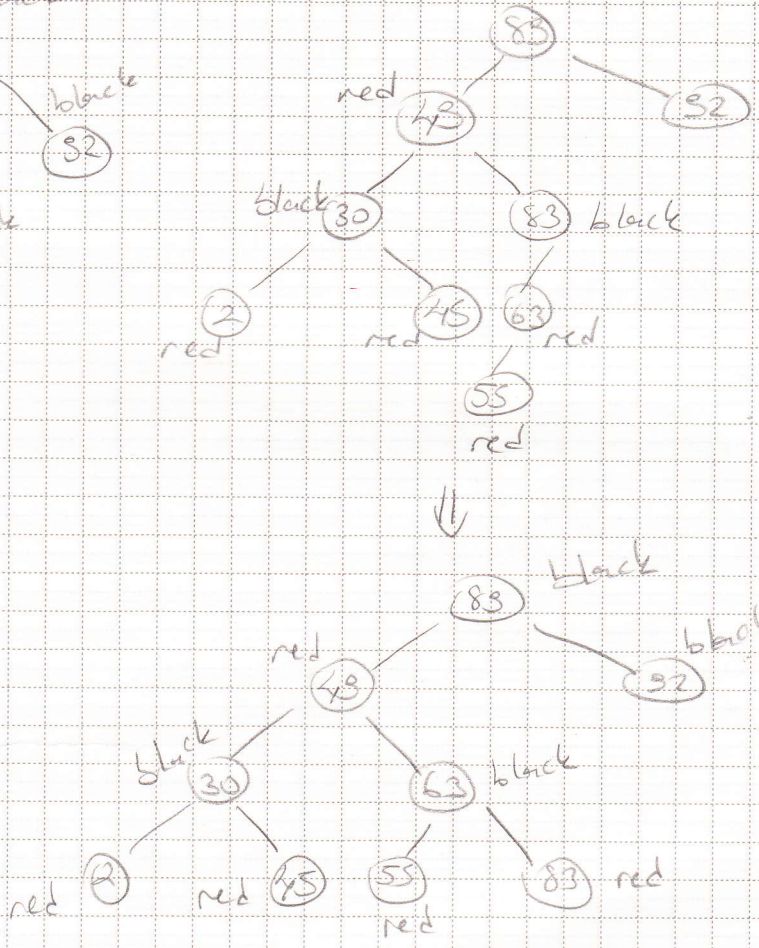
7) Insert 63



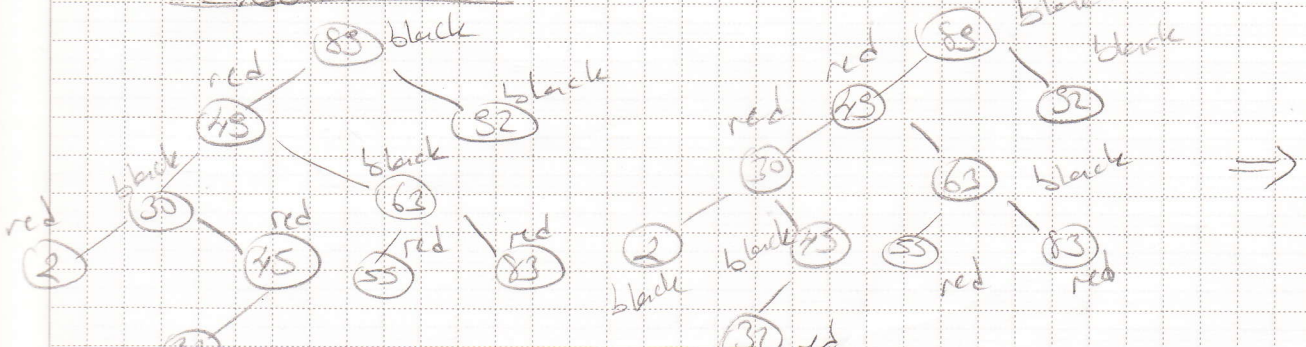
8) Insert 45

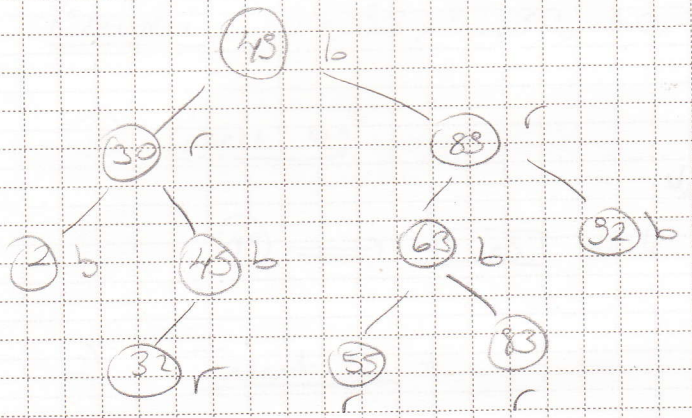


9) Insert 55



Insert 32




18
b
30
r
88
r
2
b
45
b
63
b
32
b
33
r
55
r
83
r

HASHING

-How to make searching even
O(1) with hashing

Hashing: Given a key, use a ha
function to compute an index
a hash table.
-finds, insertions, deletions.

Index = hash_function(key)

49, 11

limitations:
-table size is limited
-critical issues for implementatio

Two steps:
1) Design a good hash function
a) Fast to compute
b) Minimizing the number of co
2) Design a method to resolve c
when they occur
- seperate chaining
- open addressing

① Seperate Chaining:

- Keep a linked list of keys that hash to the same value.
- Advantage: deletion is easy
- Disadvantage = memory allocation in linked list manipulation will slow down the program.

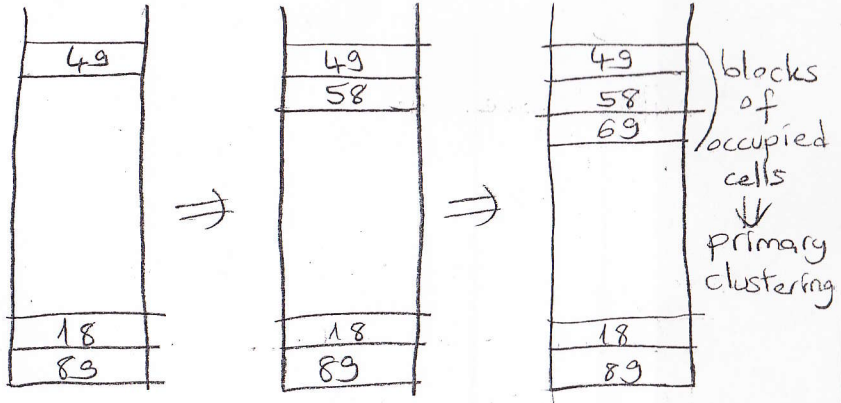
② Open addressing=

- Linear probing
  $h_i(k) = (hash(k) + i) \bmod m$ → primary clustering
  probed sequentially
- Quadratic probing:
  $h_i(k) = (hash(k) + i^2) \bmod m$ → secondary clustering
- Double hashing: $h_i(k) = [hash(k) + i * hash2(k)] \bmod m$

Cluster = a block of contiguously [illegible] table entries.

Example= Insert 49, 58, 69 with $hash(k) = k \bmod 10$ and linear probing

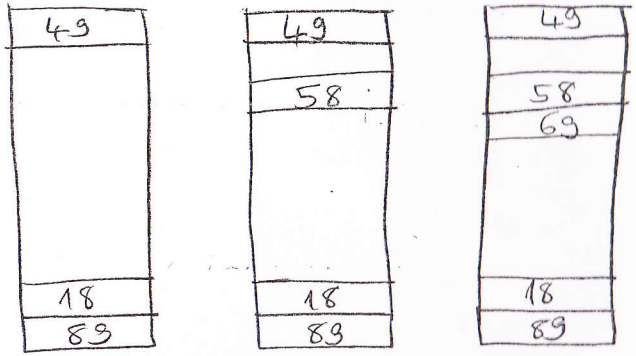
49 mod 10



Any key that hashed into the cluster will require several attempts to resolve the collision.

Quadratic Probing

→ helps to elimate primary clustering
→ two keys with different positions will have different probe sequences.



-Keys that hash to the same position will probe the same alternative cells ⇒ secondary clustering

Given 4371, 1323, 6173, 4199, 4344, 9679, 1989 hash-function $h(x) = x \bmod 10$

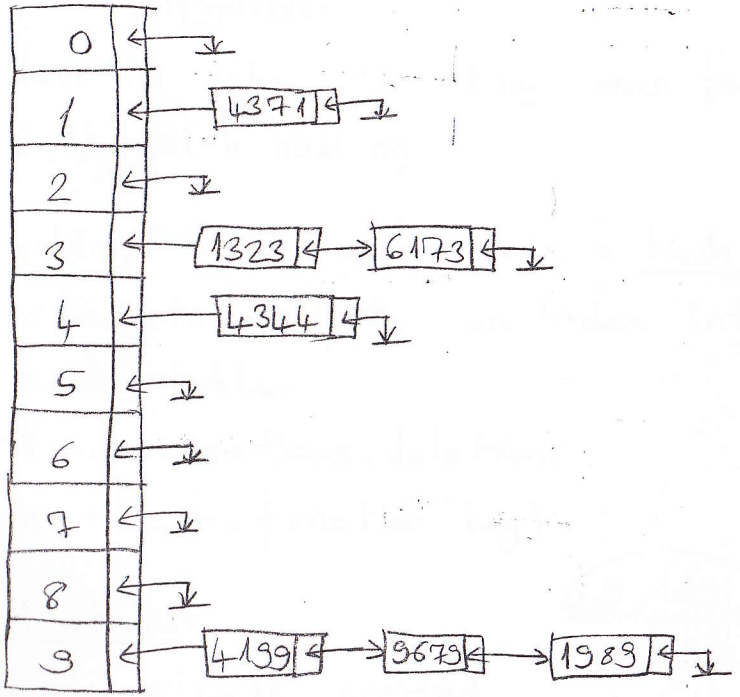
Q: What are the results of the following?

(a) Seperate chaining hash table.

(b) Open addressing hash table using linear probing.

(c) Open addressing hash table with second hash function $h_2(x) = 7 - (x \bmod 7)$

(a)



(b)