# Binary Search Trees 3/23/2022

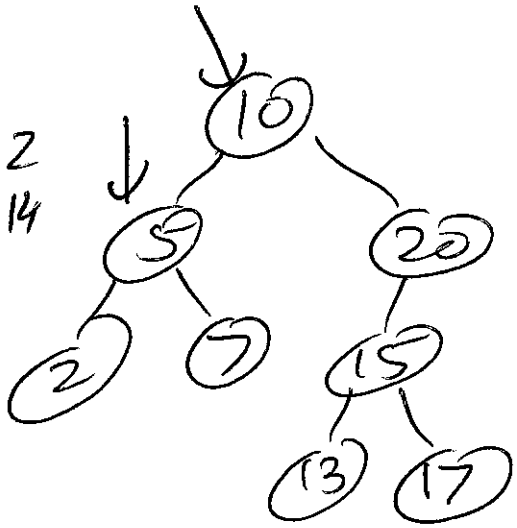
Prog 5 - will use BST

① Search
② Insert
③ Delete
④ Storing extra info struct SAVE TIME

] best, avg, worst

## ~~Insert~~ Search

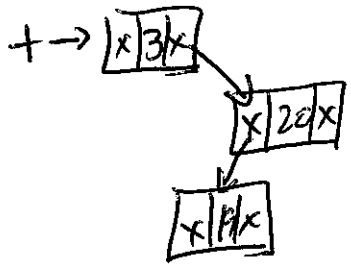
trace search 2
search 14




```
int search(bintreenode* root,
           int searchval) {

    if (root == NULL) return 0;
    if (searchval < root->data)
        return search(root->left, searchval);
    else if (searchval > root->data)
        return search(root->right, searchval);
    return 1;
}   //Exercise: Rewrite Iteratively
```

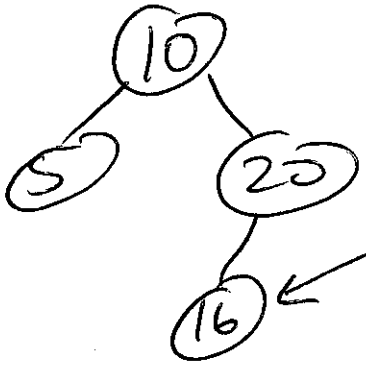
```
bintreenode* insert(bintreenode* root, int newval) {
    if (root == NULL) return newNode(newval);
    if (newval < root->data)
        root->left = insert(root->left, newval);
    else
        root->right = insert(root->right, newval);
    return root;
}
```




main

```
btn* t = NULL;
t = insert(t, 3);
t = insert(t, 20);
```

## Delete

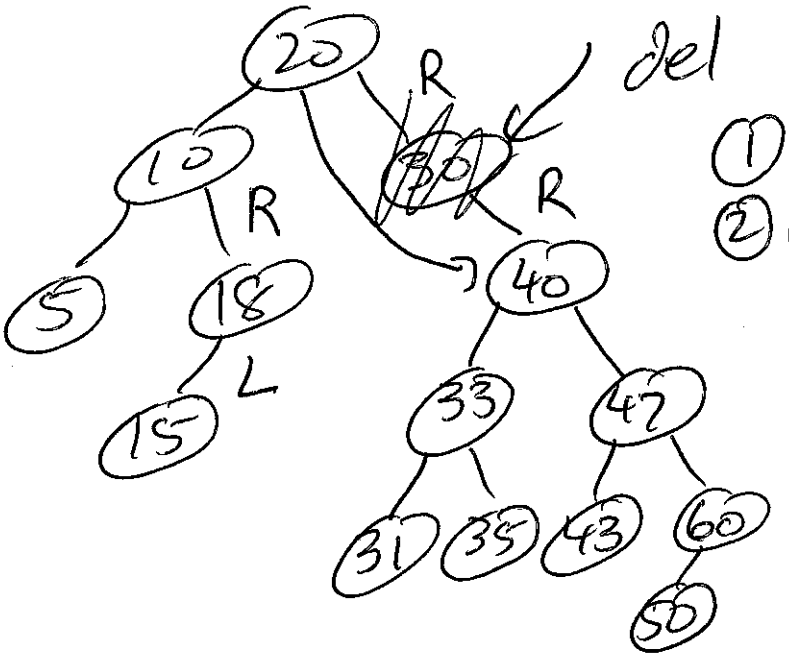



Case 1: Leaf node

(1) free mem node

(2) reset appropriate parent ptr NULL

# Case 2: Del a node w/ 1 child




(1) free node
(2) Connect its parent to its child

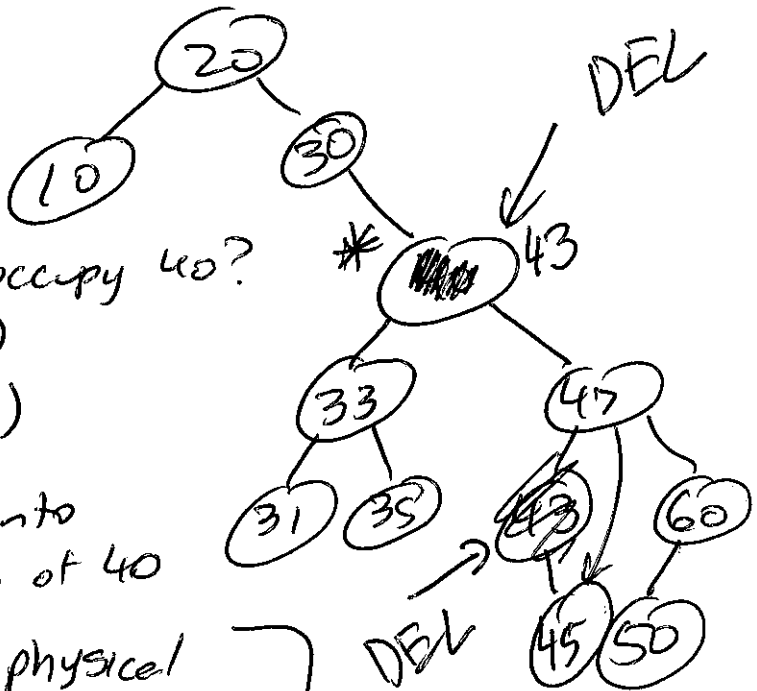
⇓

4 CASES

L, L
L, R
R, L
R, R

---

# Case 3: Del a node w/ 2 children




Who else could occupy 40?

(a) Max Left (35)

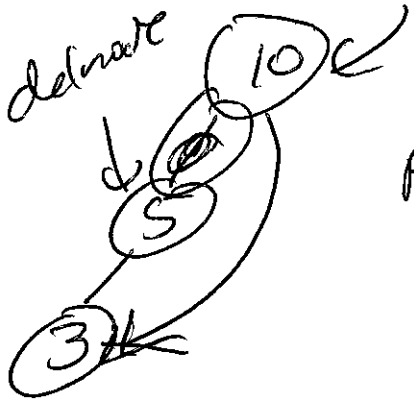
(b) Min Right (43)

(1) Move 1 of these 2 into the physical location of 40

(2) Mark the old physical location of the moved value for deletion

0 or 1 child deletion case

# Picture of LL case




par->left = par->left->left

free (delnode)

## Modifications to the struct

(1) Add parent ptr

Pro: easier to find parent

Con: maintain accuracy of every parent ptr.

(2) Store the sum of values, OR the height, or any cumulative info about that ~~sum~~ subtree in a node