Today:
- recap BST ADT
  - ↳ update: getMin()
    getMax()
    getSize()
- insert implementation
- search implementation

BST ADT (sample)

```
private:
    root
    searchRecursive(node, value)
public
    init() // constructor
    insert(value)
    search(value)
    disp()  // e.g. recursive traverse
    delete(value)
    deleteTree  // destructor
    getMin()
    getMax()
    getSize()
```

Implement Insert() method

```
// loop until we find first
   empty spot in tree

void insert(newValue)
    Node *temp = root;
    Node *n = new Node;
```

insert(3)

temp = NULL

```
n->key = newValue;
Node *prev = NULL;
while (temp != NULL)
{
    prev = temp;
    // check which way to traverse
    if (n->key < temp->key)
        temp = temp->leftChild;
    else // ≥
        temp = temp->rightChild;
}
if (prev == NULL)    // if empty tree
    root = n;
else if (n->key < prev->key)    // append to left child
  {
    prev->leftChild = n;    // add n to tree
    n->parent = prev;
  }
  else
  {
    prev->rightChild = n;
    n->parent = prev;
  }
}
```

search:

```
public:
  node* search(searchkey) // root abstracted
                          from user
private:
   node* searchRecursive(node, searchkey)
        // other methods can make use of
```

```
node*  search(search key){
     return searchRecursive(root, searchkey)
}

node*  searchR(node, skey)
if(node != NULL)
{
   if (node.key == skey)
        return node;
   else if (node.key > skey)
        return searchR (node.LeftChild, skey);
   else return searchR(node.RightChild, skey);
}
else
   return NULL;
```

Search Method can also be implemented iteratively

```
  searchIterative(node, skey)
 {while(node != NULL)
```

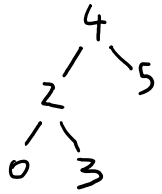
```
{while (node != NULL)
 {   if (node->key > skey)      // >
        node = node -> leftChild
     else if (node->key < skey) // <
        node = node -> rightChild
     else
        return node;
 }
```

Method to find smallest value in tree:
just need to find the left-most leaf.

```
Node*  getMin(Node *n)
{
    Node *temp = n;
    while (temp -> leftChild != NULL)
        temp = temp -> leftChild;
    return temp;
}
```

Delete

e.g. delete 2 from given tree



- Assume node to be deleted already found
- when deleting a node there are 3×2

possible cases.

1. node has no children
2. node has one child
3. node has two children

1. is root
2. is not root

- Have to come up w/ an algorithm to account for all these scenarios.