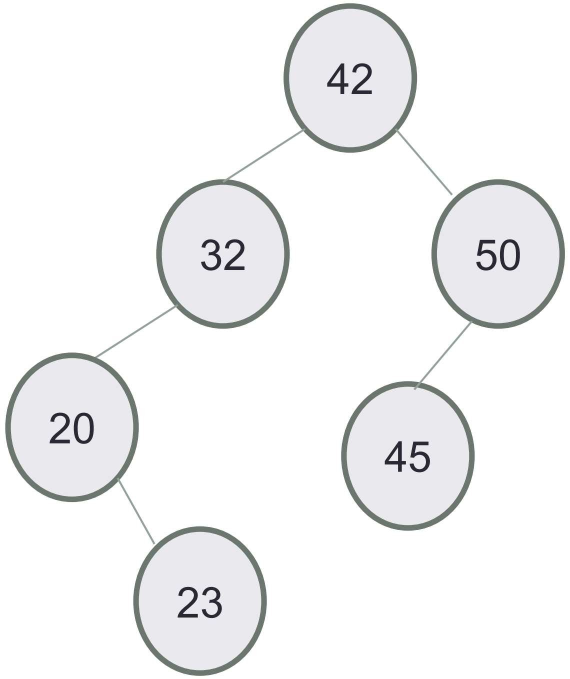
BINARY SEARCH TREES (CONTD)

Problem Solving with Computers-II

Predecessor: Next smallest element



- What is the predecessor of 32?
- Case 1: Node has a left child

  find the max element in the node's left subtree

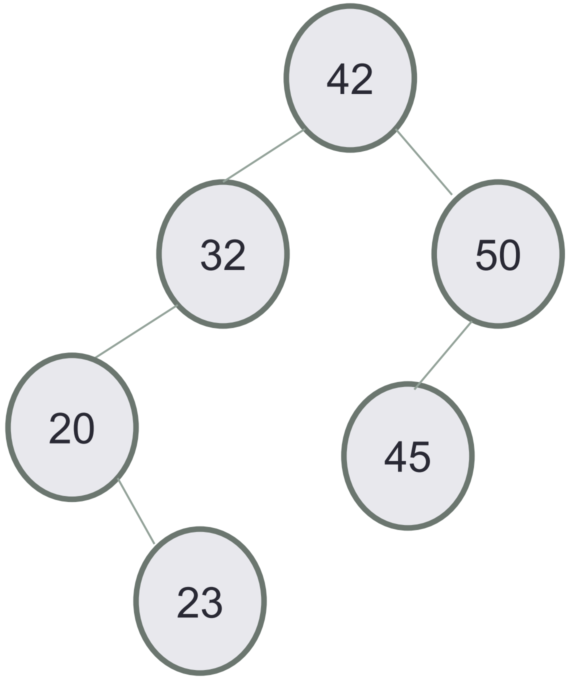
- What is the predecessor of 45?

- Case 2: No left child,

Traverse parent pointers until you take a left turn

O(H) worst case

Successor: Next largest element



- What is the successor of 20? 23
- Case 1: Node has a right child

  find the min element in the node’s right subtree

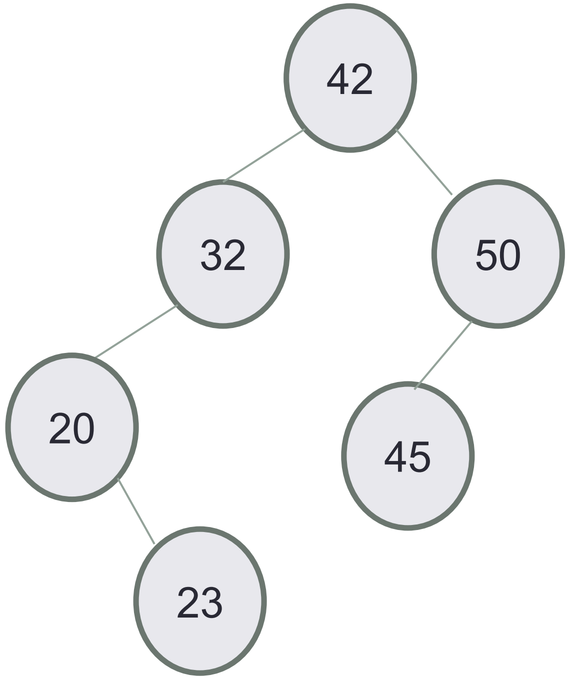
- What is the successor of 23? 32

- Case 2: No right child,

 Traverse parent pointers until you reach a node with value greater than key

O(H) worst case

Delete: Given key k, delete the node with value k



- Case 1: Node has no children
  - Delete the node and update the parent's left/right child pointer as the case may be
- Case 2:Node has one child (either left or right) which may have other children
  - Promote the only child to take node's spot, delete the node.

Case 3: Node has two children:

Swap the node's key value with that of its predecessor/successor, and then delete the node (with given key) --- now either case 1 or case2

Finding the Maximum of Two Integers

Here's a small function that you might write to find the maximum of two integers.

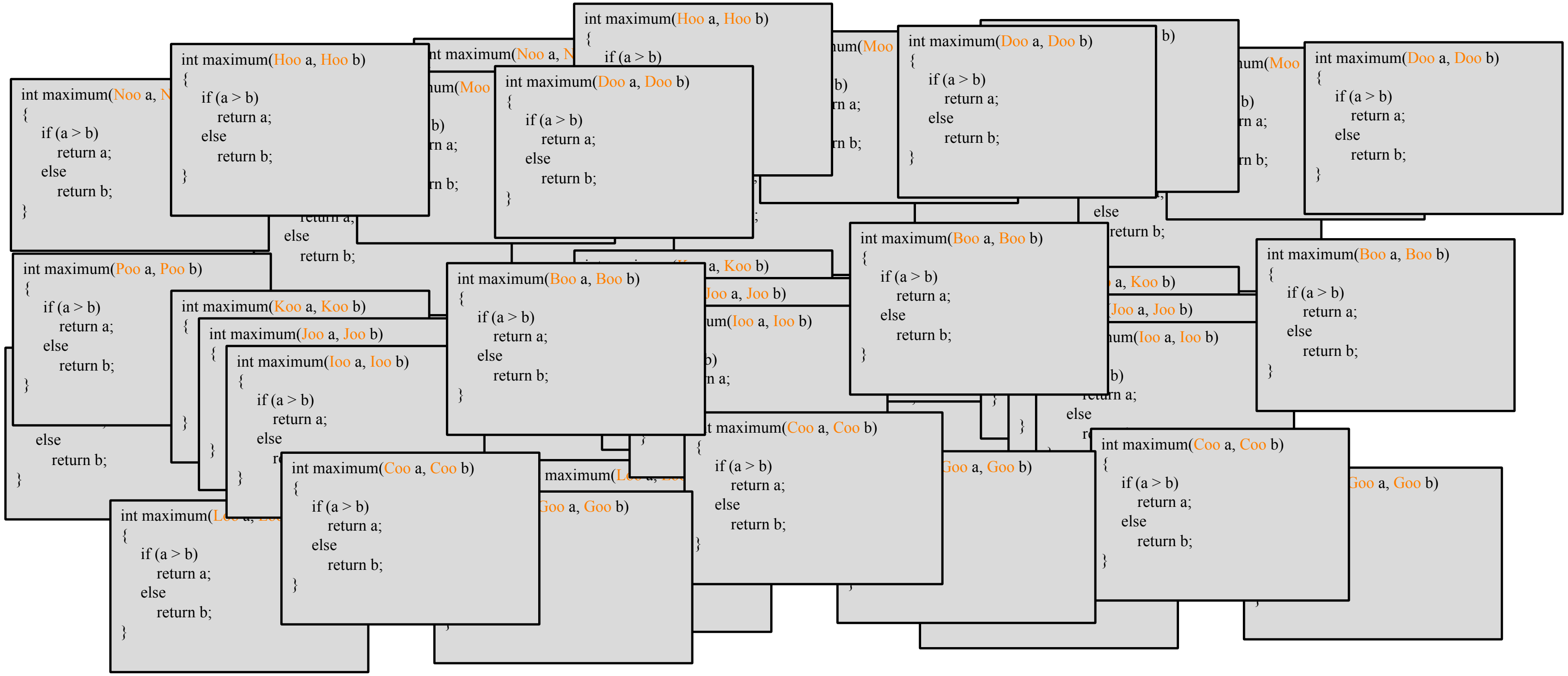
```
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Finding the Maximum of Two Points

```
point maximum(Point a, Point b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

One Hundred Million Functions...

- Suppose your program uses 100,000,000 different data types, and you need a maximum function for each...



A Template Function for Maximum

- When you write a template function, you choose a data type for the function to depend upon...

```
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

What are the advantages over typedef?

```
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

```
typedef int item;
item maximum(item a, item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Demo maximal.cxx

A Template Function for Maximum

Is the following a valid template function?

A. Yes
B. No

```
template <class Item>
Item maximum(Item a, Item b)
{
    Item result;
    if (a > b)
        result = a;
    else
        result = b;
    return result;
}
```

BST, with templates:

```
template<typename Data>

class BSTNode {
public:
  BSTNode<Data>* left;
  BSTNode<Data>* right;
  BSTNode<Data>* parent;
  Data const data;

  BSTNode( const Data & d ) :
      data(d) {
    left = right = parent = 0;
  }

};
```

BST, with templates:

```
template<typename Data>

class BSTNode {
public:
  BSTNode<Data>* left;
  BSTNode<Data>* right;
  BSTNode<Data>* parent;
  Data const data;

  BSTNode( const Data & d ) :
     data(d) {
    left = right = parent = 0;
  }

};
```

How would you create **a BSTNode object** on the runtime stack?

A. BSTNode n(10);
B. BSTNode<int> n;
C. BSTNode<int> n(10);
D. BSTNode<int> n = new BSTNode<int>(10);
E. More than one of these will work

{ } syntax OK too

BST, with templates:

```
template<typename Data>

class BSTNode {
public:
  BSTNode<Data>* left;
  BSTNode<Data>* right;
  BSTNode<Data>* parent;
  Data const data;

  BSTNode( const Data & d ) :
     data(d) {
    left = right = parent = 0;
  }

};
```

How would you create a **pointer** to BSTNode with integer data?

A. BSTNode* nodePtr;
B. BSTNode<int> nodePtr;
C. BSTNode<int>* nodePtr;

BST, with templates:

```
template<typename Data>

class BSTNode {
public:
  BSTNode<Data>* left;
  BSTNode<Data>* right;
  BSTNode<Data>* parent;
  Data const data;

  BSTNode( const Data & d ) :
     data(d) {
    left = right = parent = 0;
  }

};
```

Complete the line of code to create a new BSTNode object with int data on the heap and assign nodePtr to point to it.

```
BSTNode<int>* nodePtr
```

Working with a BST

```
template<typename Data>
class BST {

private:

  /** Pointer to the root of this BST, or 0 if the BST is empty */
  BSTNode<Data>* root;

public:

  /** Default constructor. Initialize an empty BST. */
  BST() : root(nullptr){  }

  void insertAsLeftChild(BSTNode<Data>* parent, const Data & item)
  {
     // Your code here
  }
```

Working with a BST: Insert

```
void insertAsLeftChild(BSTNode<Data>* parent, const Data & item)
  {
     // Your code here
  }
```

Which line of code correctly inserts the data item into the BST as the left child of the parent parameter.

```
A. parent.left = item;
B. parent->left = item;
C. parent->left = BSTNode(item);
D. parent->left = new BSTNode<Data>(item);
E. parent->left = new Data(item);
```

Working with a BST: Insert

```
void insertAsLeftChild(BSTNode<Data>* parent, const Data & item)
{
    parent->left = new BSTNode<Data>(item);
}
```

Is this function complete? (i.e. does it to everything it needs to correctly insert the node?)

A. Yes. The function correctly inserts the data
B. No. There is something missing.

Working with a BST: Insert

```
void insertAsLeftChild(BSTNode<Data>* parent, const Data & item)
{

    parent->left = new BSTNode<Data>(item);


}
```

Template classes

Using a Typedef Statement:

```
class bag
{
public:
    typedef int value_type;
    . . .
```

Using a Template Class:

```
template <class Item>
class bag
{
public:
    typedef Item value_type;
    . . .
```

Template classes: Non-member functions

```
bag operator +(const bag& b1, const bag& b2)...
```

```
template <class Item>
bag<Item> operator +(const bag<Item>& b1, const bag<Item>& b2)...
```

Template classes: Member function prototype

- Rewrite the prototype of the member function “count” using templates

Before (without templates)

```
class bag{
        public:
                typedef std::size_t size_type;
                ….
                size_type count(const value_type& target) const;
                …..
};
```

Template classes: Member function definition

```
bag::size_type bag::count(const value_type& target) const ...
```

The function's return type is specified as `bag::size_type`. But this return type is specified before the compiler realizes that this is a bag member function. So we must put the keyword *typename* before `bag<Item>::size_type`. We also use `Item` instead of `value_type`:

```
template <class Item>
typename bag<Item>::size_type  bag<Item>::count
    (const Item & target) const ...
```

Template classes: Including the implementation

```
#include "bag4.template"   // Include the implementation.
```

How to Convert a Container Class to a Template

1. The template prefix precedes each function prototype or implementation.
2. Outside the class definition, place the word <Item> with the class name, such as bag<Item>.
3. Use the name Item instead of value_type.
4. Outside of member functions and the class definition itself, add the keyword *typename* before any use of one of the class's type names. For example:

   typename bag<Item>::size_type
5. The implementation file name now ends with .template (instead of .cxx), and it is included in the header by an include directive.
6. Eliminate any using directives in the implementation file. Therefore, we must then write std:: in front of any Standard Library function such as std::copy.
7. Some compilers require any default argument to be in both the prototype and the function implementation.

Review and demo bag4

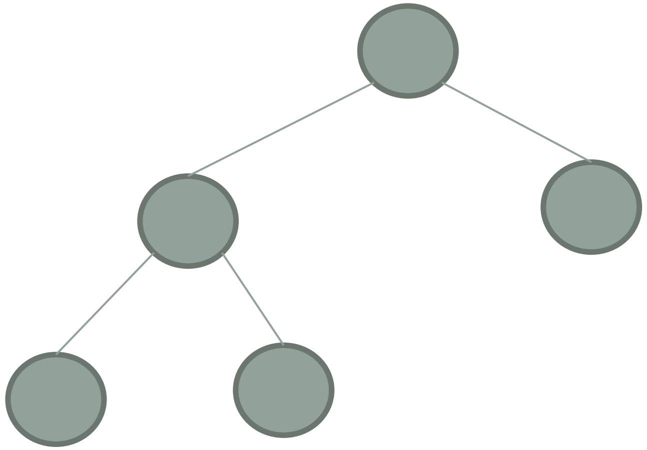
Using a template class

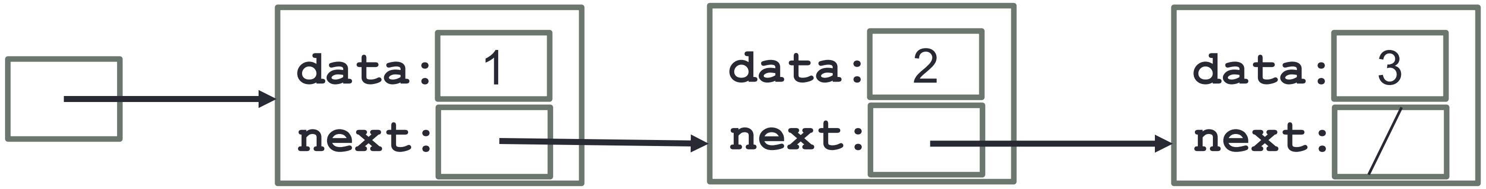
```
bag<string> adjectives; // Contains adjectives typed by user
bag<int>    ages;       // Contains ages in the teens
bag<string> names;      // Contains names typed by user
```

Worst case analysis

Are binary search trees *really* faster than linked lists for finding elements?

- A. Yes
- B. No





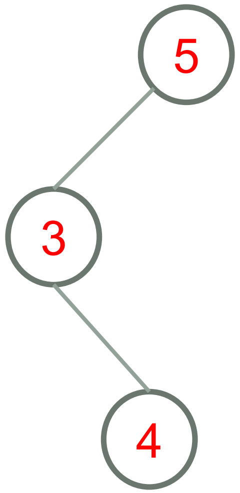
Summary of operations

Average case analysis of a "successful" find

Given a BST having N nodes x_1, .. $x_{N,}$ such that key(x_i) = k_i

How many compares to locate a key in the BST?

1. Worst case:
2. Best case:
3. Average case:



Here is the result! Proof is a bit involved but if you are interested in the proof, come to office hours

$D_{avg}(N)$	Average #comparisons to find a single item in any BST with N nodes

$$D_{avg}(N) \approx 1.386 \ \log_2 N$$

Conclusion: The average time to find an element in a BST with no restrictions on shape is Θ(log N).