CS3500: Object-Oriented Design
Spring 2014

Class 9
2.7.2014

Northeastern University
College *of* Computer and Information Science

Today...

- Assignments
- Binary Search
- Total Order
- Binary Search Trees

Assignments 4 Recap

```
import java.util.Iterator;
import java.util.NoSuchElementException;
/**
 * IntegerIterator iterates through Integers 0-MAX
 */
public class IntegerIterator implements Iterator<Integer>{
    int n; //store the current value to return
    final int MAX = 10; //Max value to return

    /**
     * Initializes the state of the iterator to the starting
     * value of 0
     */
    public IntegerIterator(){
        n = 0;
    }
    /**
     * @return whether there is another integer in the iterator, which is determined by whether n<=MAX
     */
    public boolean hasNext(){
        return (n <= MAX);
    }
    /**
     * Returns the next Integer and updates n
     * @return the next Integer
     */
    public Integer next(){
        if(hasNext()){
            Integer result = new Integer(n);
            n = n + 1;
            return result;
        } else{
            throw new NoSuchElementException();
        }
    }
    /**
     * remove is an Unsupported Operation
     */
    public void remove(){
        throw new UnsupportedOperationException("remove");
    }
}
```

State of iterator (field/s)

Initalize state (field/s)

Does the iterator have another element?

Keys to next method:
- check that next element exists
- return the next element
- update state in preparation for following call to next

Questions to consider when writing an iterator

- State
  - What will be the state (field/s) of the iterator?
- Constructor
  - What will the state be initialized to?
  - What will be passed to the constructor? (What parameters will the constructor take?)
- hasNext method
  - How do we know if there is another element in the iterator? (What condition can we test?)
- next method
  - Does another element exist?
  - What is the next element to return?
  - How must we update the state in order to prepare for following call to next?

Generic Syntax

Generic class syntax (Sestoft, p.78):

```
class-modifiers class C<T1, ..., Tn> class-base-clause
{ class-body }
```

Generic method syntax (Sestoft, p.86):

```
method-modifiers <T1, ..., Tn> returntype m(formal-list)
{ method-body }
```

Example:

```
public abstract class MyMap<K, V> implements Iterable<K>{
    ...
    public static <K, V> MyMap<K, V> empty() {
        return new Empty<K, V>();
    }
    ...
}
```

Binary Search

Northeastern University
College *of* Computer and Information Science

Binary Search

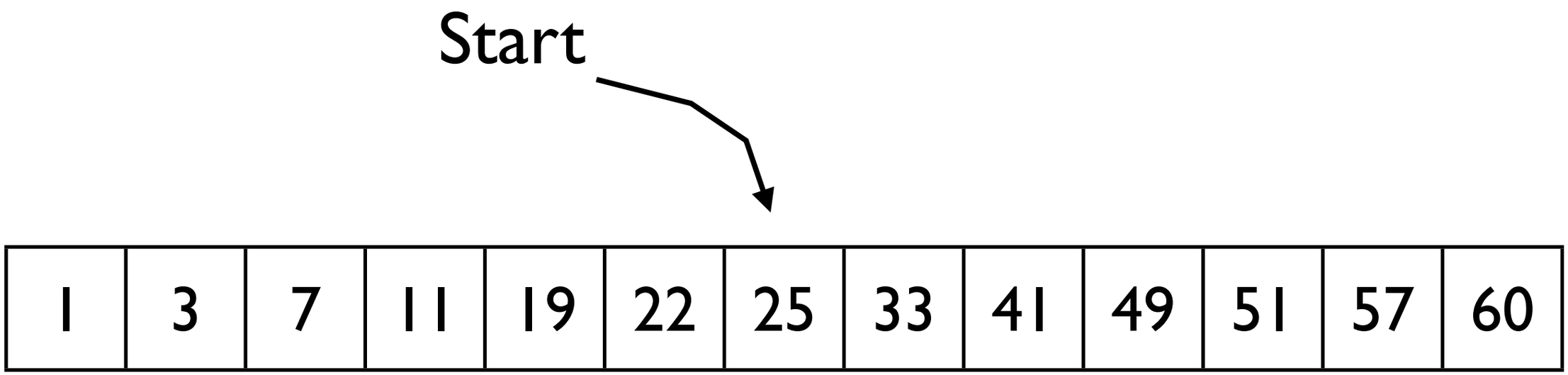
If a set *S* is represented by a sorted linear sequence, then we can determine whether *x* is an element of *S* in logarithmic time by using binary search.

Binary Search

1	3	7	11	19	22	25	33	41	49	51	57	60

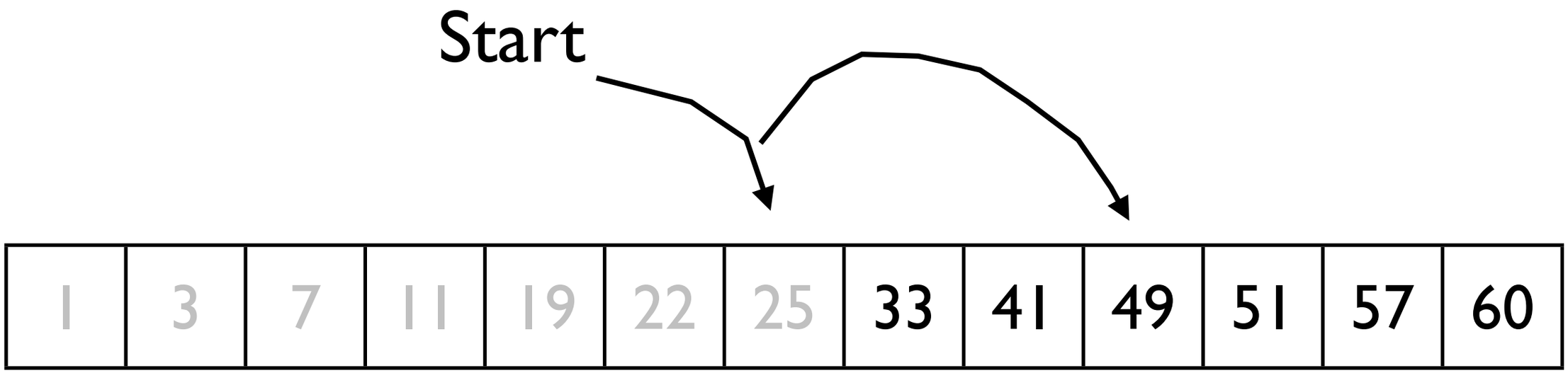
Search for 51

Binary Search



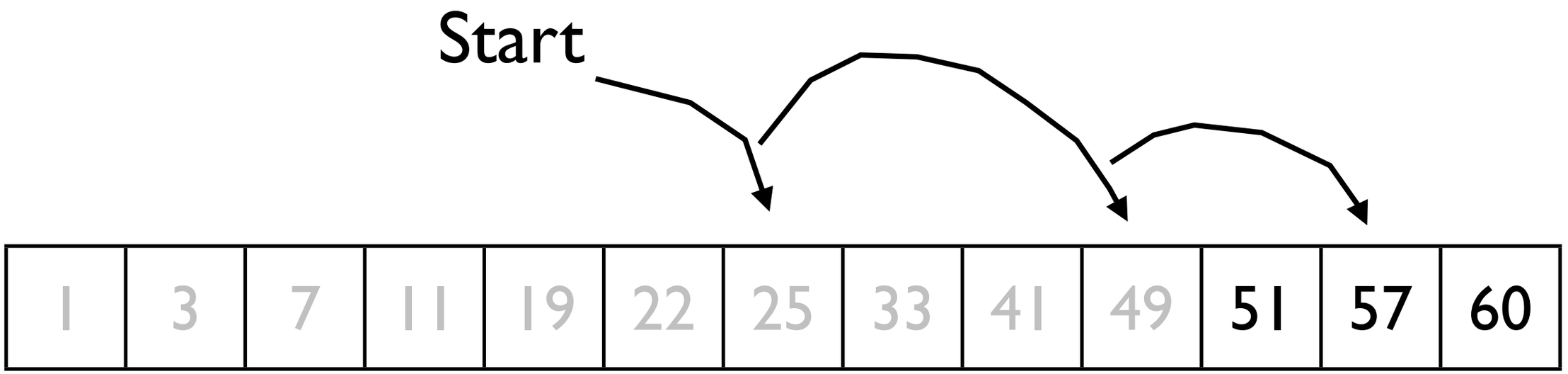
Search for 51

Binary Search



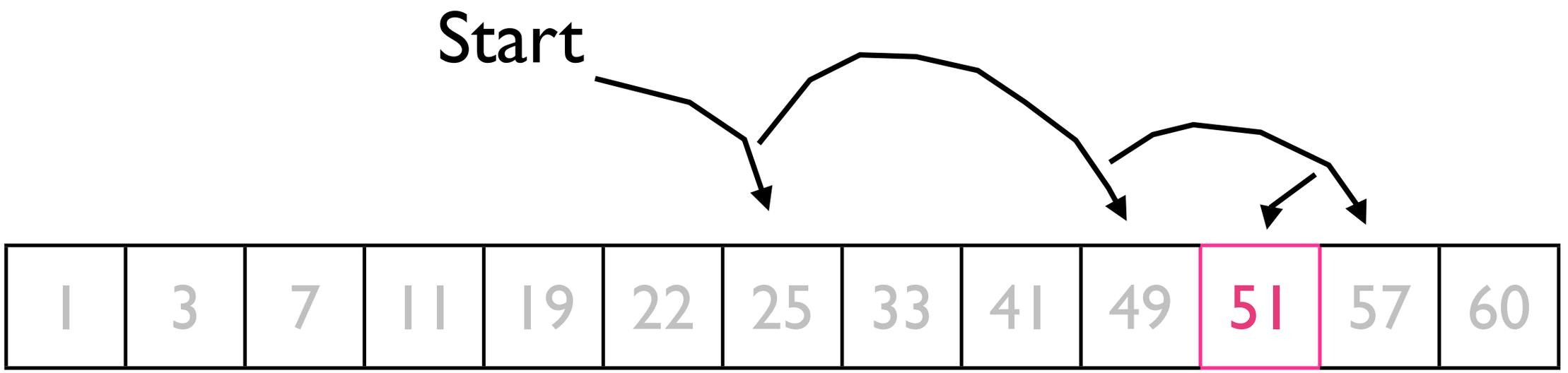
Search for 51

Binary Search



Search for 51

Binary Search



Search for 51

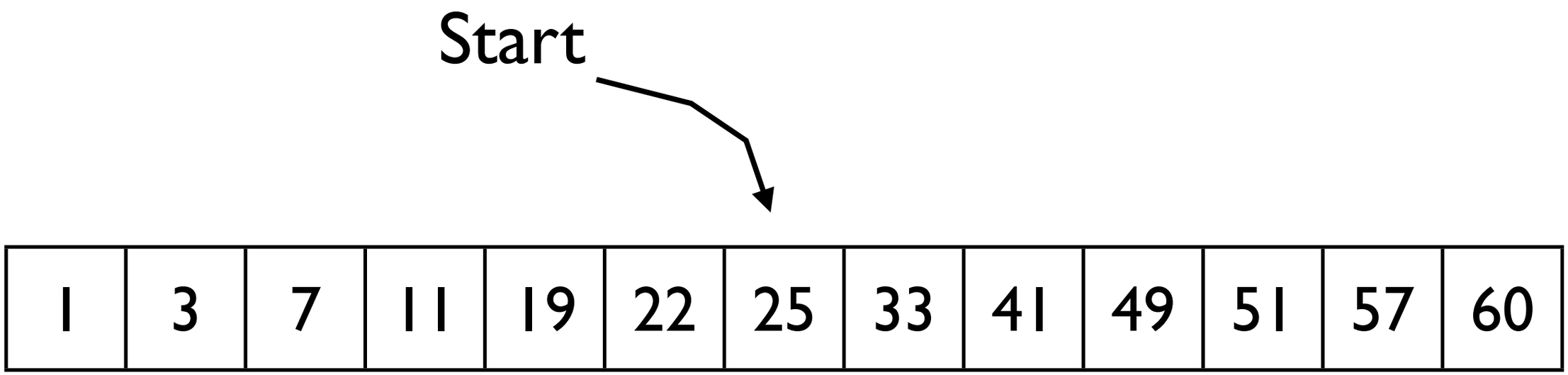
FOUND

Binary Search

1	3	7	11	19	22	25	33	41	49	51	57	60

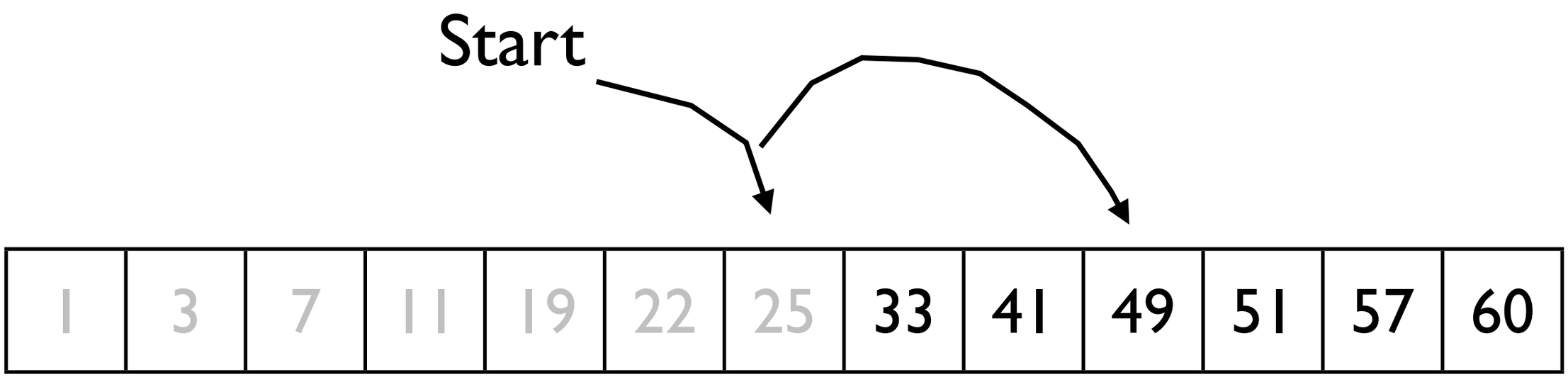
Search for 53

Binary Search



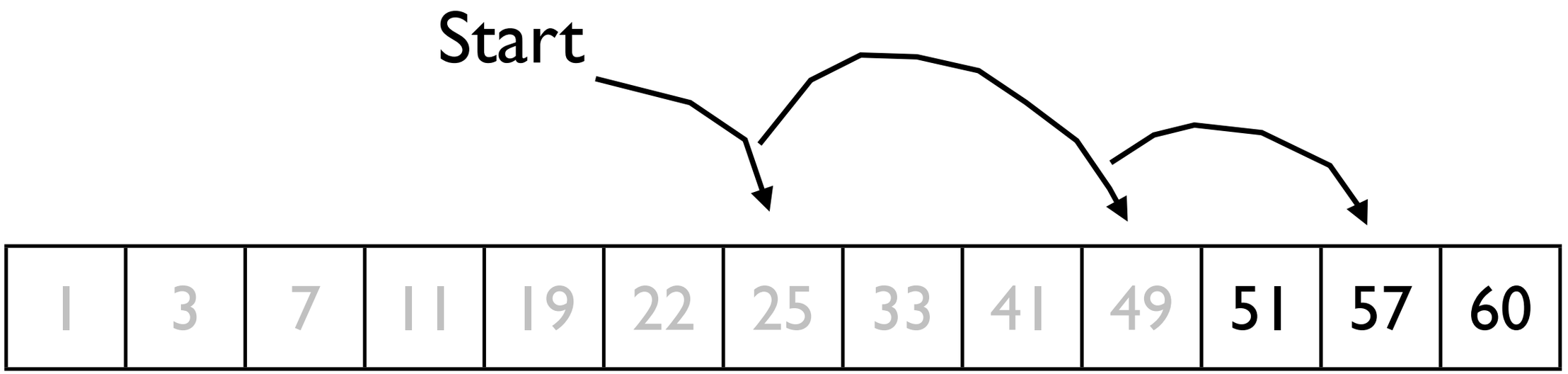
Search for 53

Binary Search



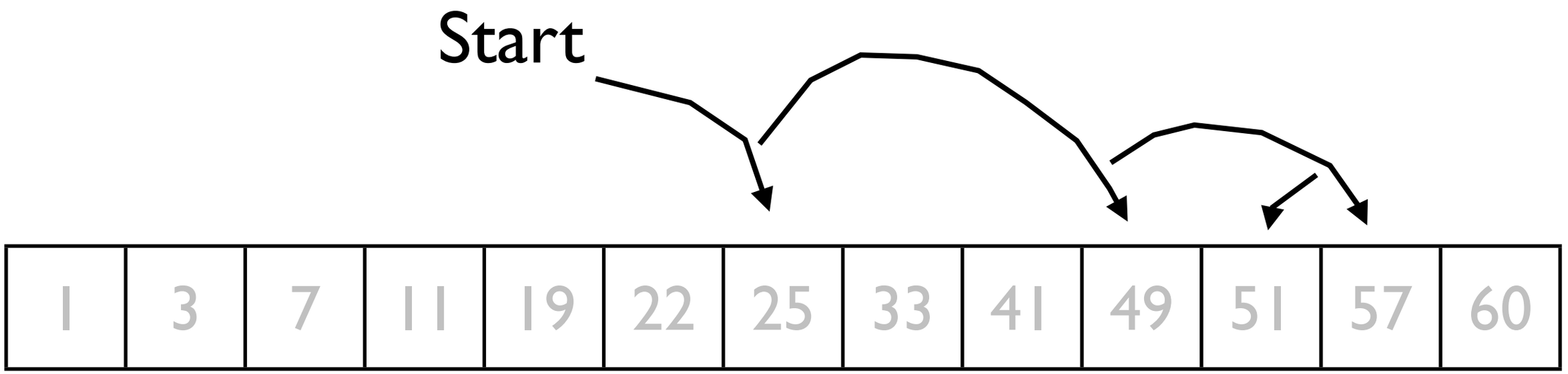
Search for 53

Binary Search



Search for 53

Binary Search



Search for 53

NOT FOUND

```
/**
 * PRE: 0 <= min <= max <= data.length
 * @param data sorted array of ints
 * @param min min index to search
 * @param max max index to search
 * @param target value searching for in data
 * @return index of target in data,
 *         or -1 if target is not in data
 */
 public static int binarySearch(int[] data, int min, int max,
                                int target){

 }
```

```
/**
 * PRE: 0 <= min <= max <= data.length
 * @param data sorted array of ints
 * @param min min index to search
 * @param max max index to search
 * @param target value searching for in data
 * @return index of target in data,
 *       or -1 if target is not in data
 */
 public static int binarySearch(int[] data, int min, int max,
                                int target){
    int index = -1; //not in array
    int midpoint = (min+max)/2;

    if(data[midpoint] == target)
        index = midpoint;
    else
        if(data[midpoint] > target){
         if(min <= midpoint-1)
              index = binarySearch(data, min, midpoint-1, target);
        }else{
         if(midpoint+1 <= max)
              index = binarySearch(data, midpoint+1, max, target);
        }
     return index;
}
```

Total Order

Total Order

A *total order* on some set D is a binary relation R on D such that

- R is transitive
- R is anti-symmetric
- R satisfies the law of trichotomy

Total Order

A *total order* on some set D is a binary relation R on D such that

- R is **transitive**: if xRy and yRz, then xRz
- R is anti-symmetric
- R satisfies the law of trichotomy

Total Order

A *total order* on some set D is a binary relation R on D such that

- R is transitive
- R is **anti-symmetric**: if xRy and yRx, then x = y
- R satisfies the law of trichotomy

Total Order

A *total order* on some set D is a binary relation R on D such that

- R is transitive
- R is anti-symmetric
- R satisfies the **law of trichotomy**: ∀x, ∀y either xRy or yRx

  The law of trichotomy (a division into three categories) can also be phrased as ∀x, ∀y either

  x=y

  or(x!=y and xRy)

  or(x!=y and yRx)

Examples of Total Orders

Usual Ordering on Integers

(R :<=)

- R is transitive
- R is anti-symmetric
- R satisfies the law of trichotomy

Reverse Ordering on Integers

(R :>=)

- R is transitive
- R is anti-symmetric
- R satisfies the law of trichotomy

Ordering on Integers

- every even integer is less than every odd integer
- the even integers are ordered by the usual <=
- the odd integers are ordered by the reverse >=

- R is transitive
- R is anti-symmetric
- R satisfies the law of trichotomy

Trees

Tree Basics
[Lewis & Chase]

- Tree: "a non-linear structure in which elements are organized into a hierarchy"
  - Tree contains nodes (elements) and edges (connect nodes)
- Root: single node at top level of tree
- "The nodes at lower levels of the tree are the children of nodes at the previous level. Nodes that have the same parent are called siblings."
- Leaf: "node that does not have any children"
- Internal node: "node that is not the root and has at least one child"

Binary Trees

Northeastern University
College *of* Computer and Information Science

Labeled Binary Tree (LBT)

- an empty tree
- a node with three components:
  - a label
  - a left subtree, which is a labeled binary tree
  - a right subtree, which is a labeled binary tree

Binary Search Trees

Northeastern University
College *of* Computer and Information Science

Binary Search Tree (BST)

- t is empty
- t is a node
  - a label
  - the left subtree of t is a BST,
  - the right subtree of t is a BST,
  - every label within the left subtree of t is less than the label of t,
  - every label within the right subtree of t is greater than the label of t

BST Invariants

BST Invariants

- No duplicates
- left is a BST
- right is a BST
- all elements in left BST are less than current
- all elements in right BST are more than current

Checking invariants at run time.

Checking invariants at run time.

```
private static final boolean DEBUGGING = false; //within Node subclass

...

public boolean isEmpty () {
    if (DEBUGGING) {
        if(!repOk()){
          System.out.println("!repOk");
        }
    }
     ...
}

public int size () {
    if (DEBUGGING) {
        if(!repOk()){
          System.out.println("!repOk");
        }
    }
    ...
}
...
```

Overriding the `toString()` method when debugging

Overriding the `toString()` method when debugging

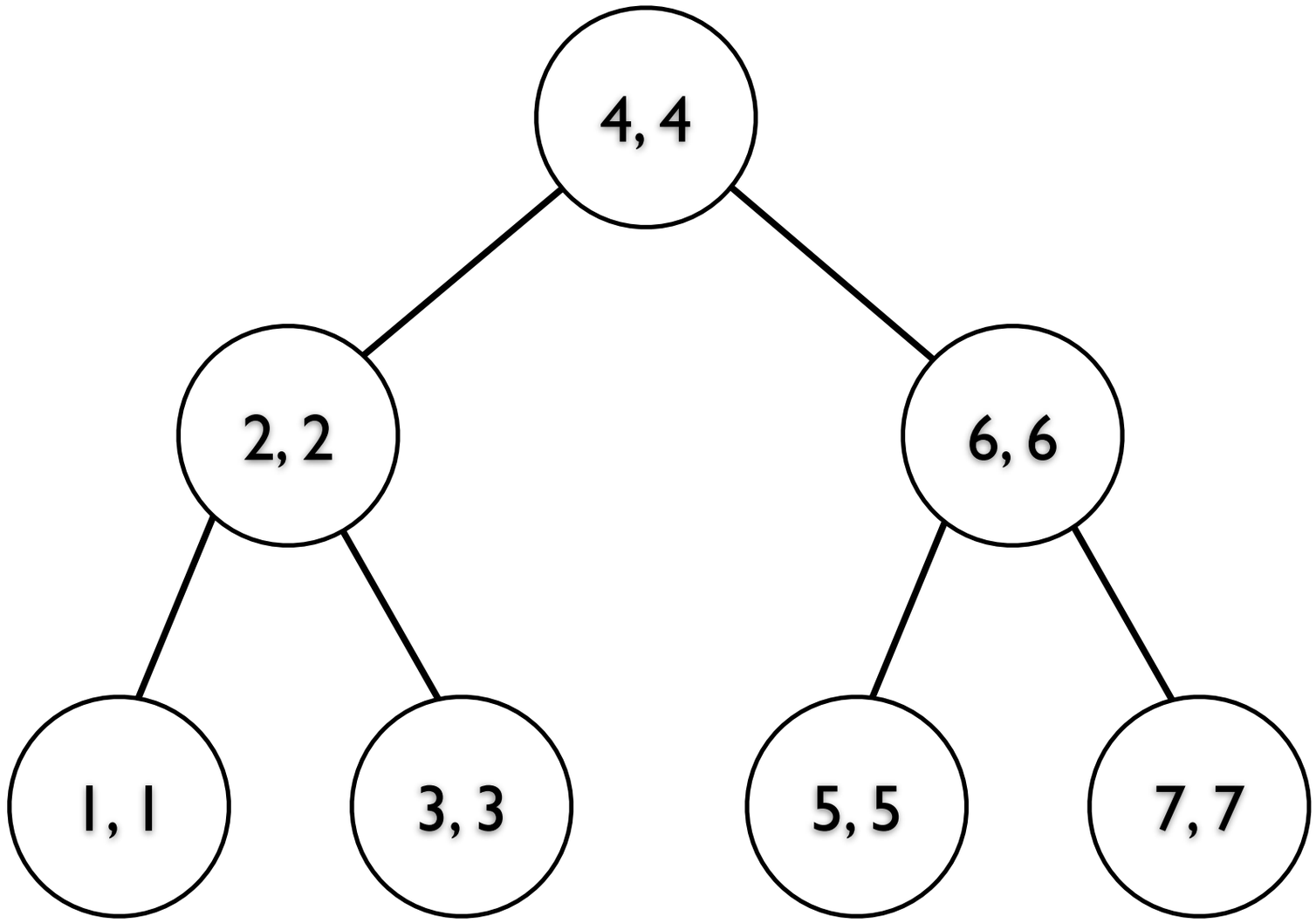
```
//within Node subclass
public String toString () {
    if (DEBUGGING) {
        if(!repOk()){
            System.out.println("!repOk");
        }
        return toString(0);
    }
    else return super.toString();
}
```

```
4, 4
   6, 6
      7, 7


      5, 5


   2, 2
      3, 3


      1, 1
```

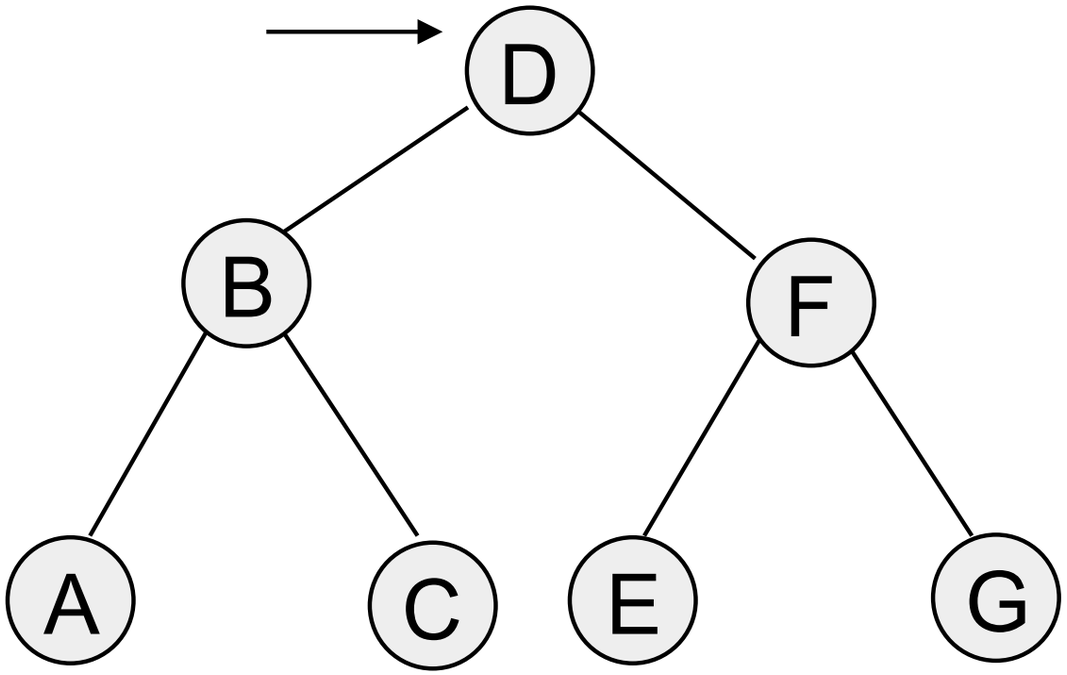


```
private String toString (int n) {
    String result = "";
    for (int i = 0; i < n; i = i + 1)
        result = result + " ";
    result = result + " " + k0 + ", " + v0;
    result = result + "\n";
    if (right.isEmpty())
        result = result + "\n";
    else {
        Node<K,V> right = (Node<K,V>) this.right;
        result = result + right.toString (n + 1);
    }
    if (left.isEmpty())
        result = result + "\n";
    else {
        Node<K,V> left = (Node<K,V>) this.left;
        result = result + left.toString (n + 1);
    }
    return result;
}
```

Tree Traversals

- We'd like to visit each data item in a tree
- Are the items randomly ordered, as in a bag or set?
- Think of visiting the data in a node, and its left and right subtrees, in some order

Preorder Traversal

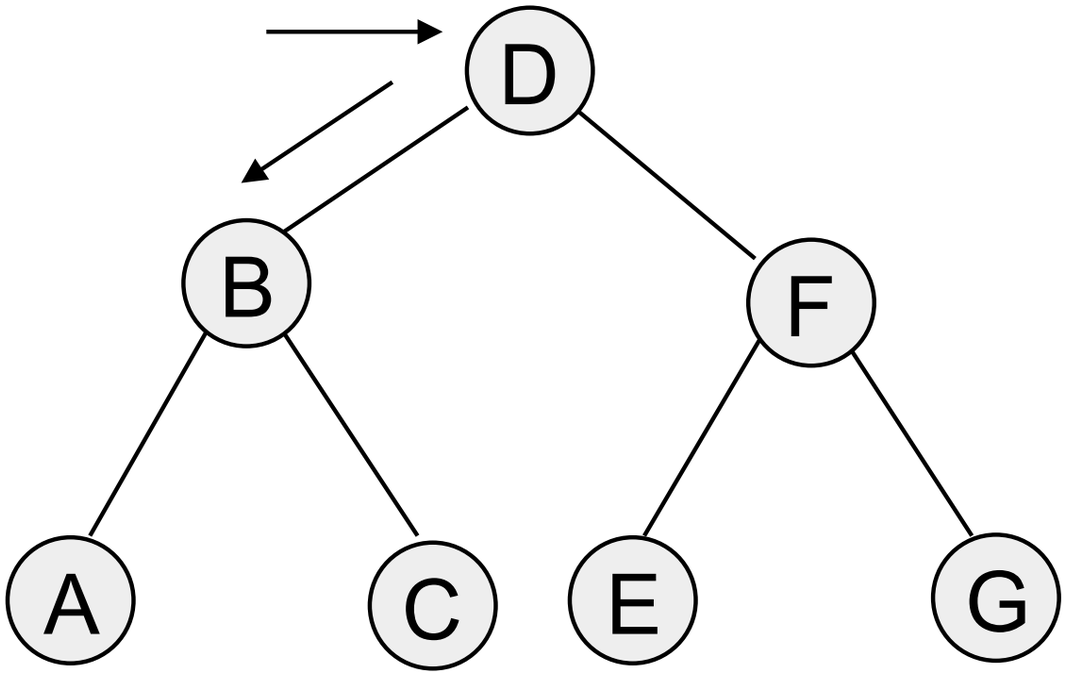


Order of nodes visited:

D

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

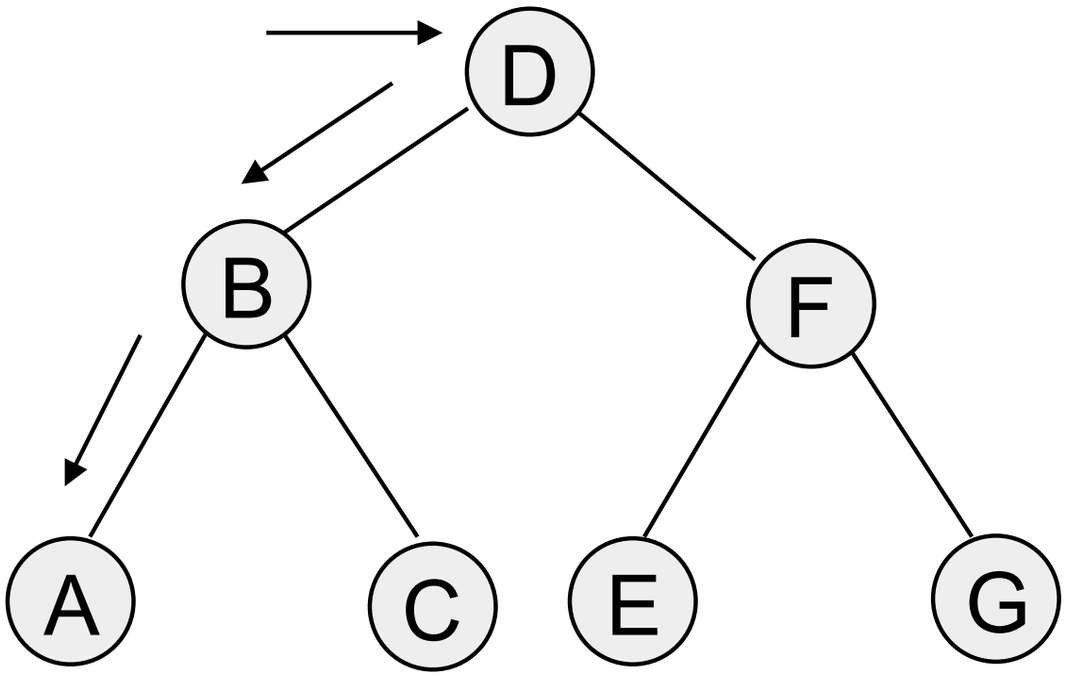


Order of nodes visited:

D B

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

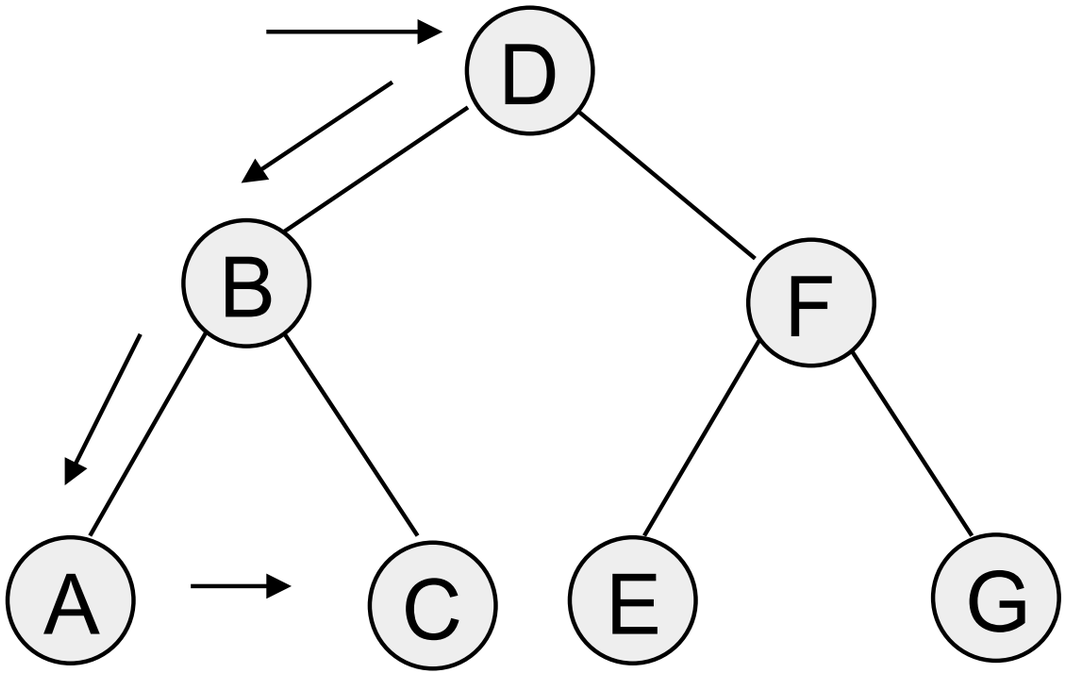


Order of nodes visited:

D B A

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

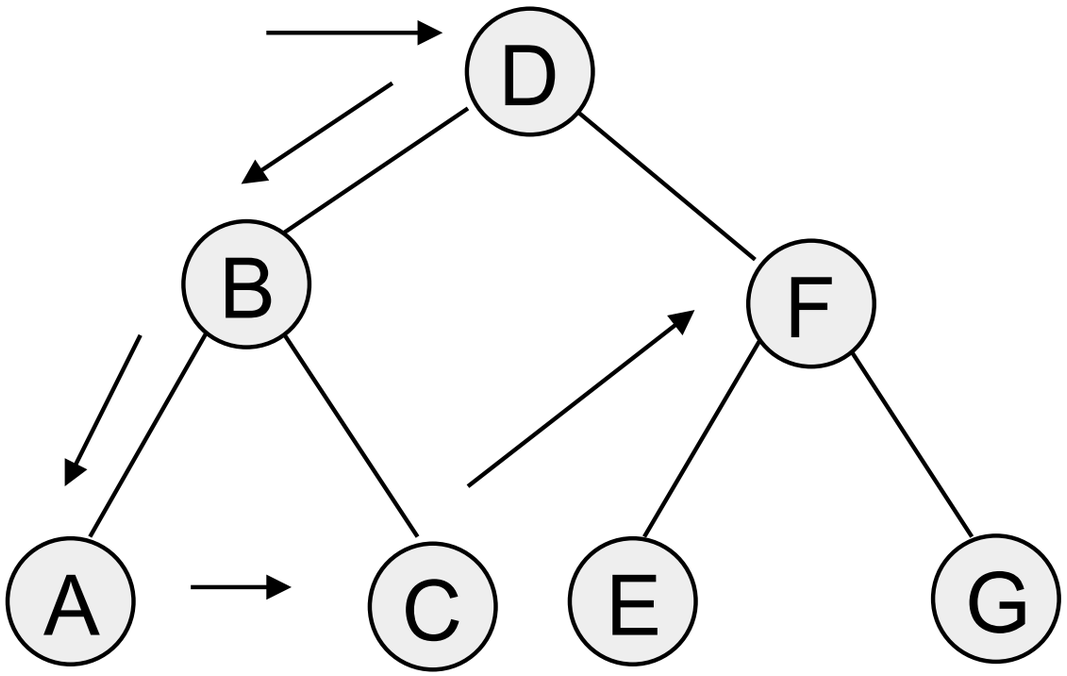


Order of nodes visited:

D B A C

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

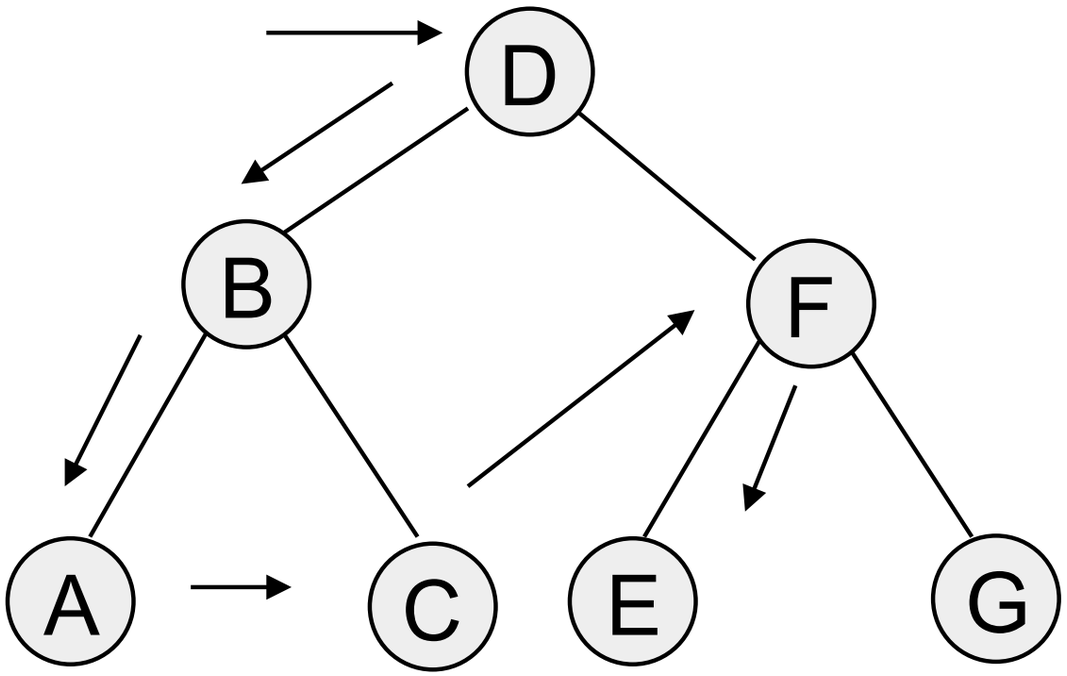


Order of nodes visited:

D B A C F

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

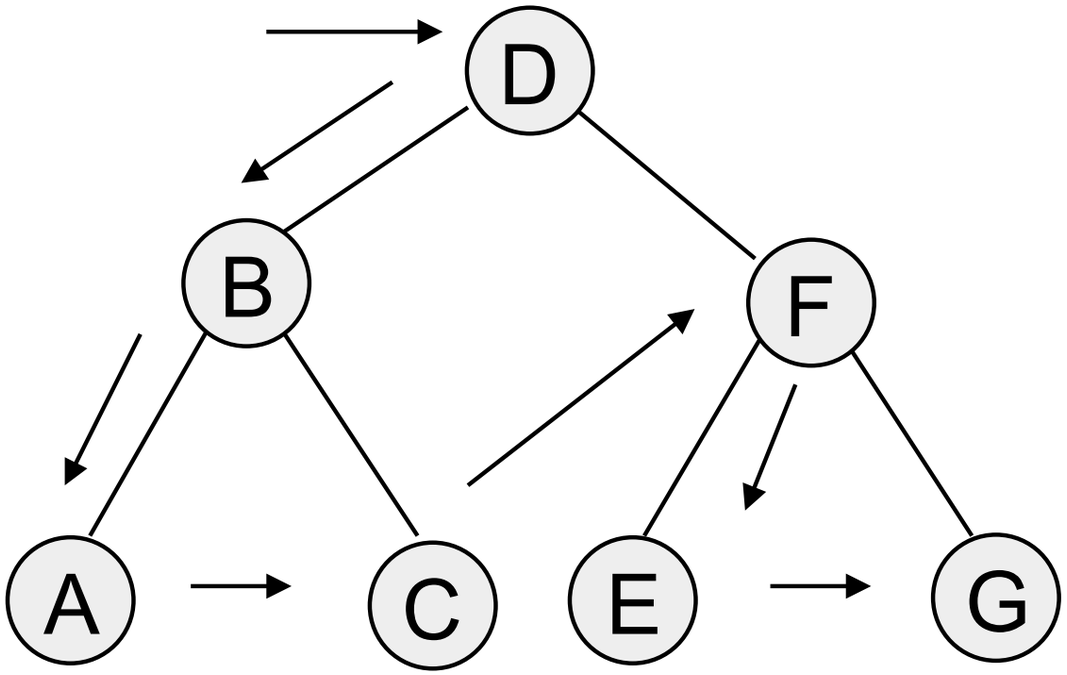


Order of nodes visited:

D B A C F E

Visit the data
Visit the left subtree
Visit the right subtree

Preorder Traversal

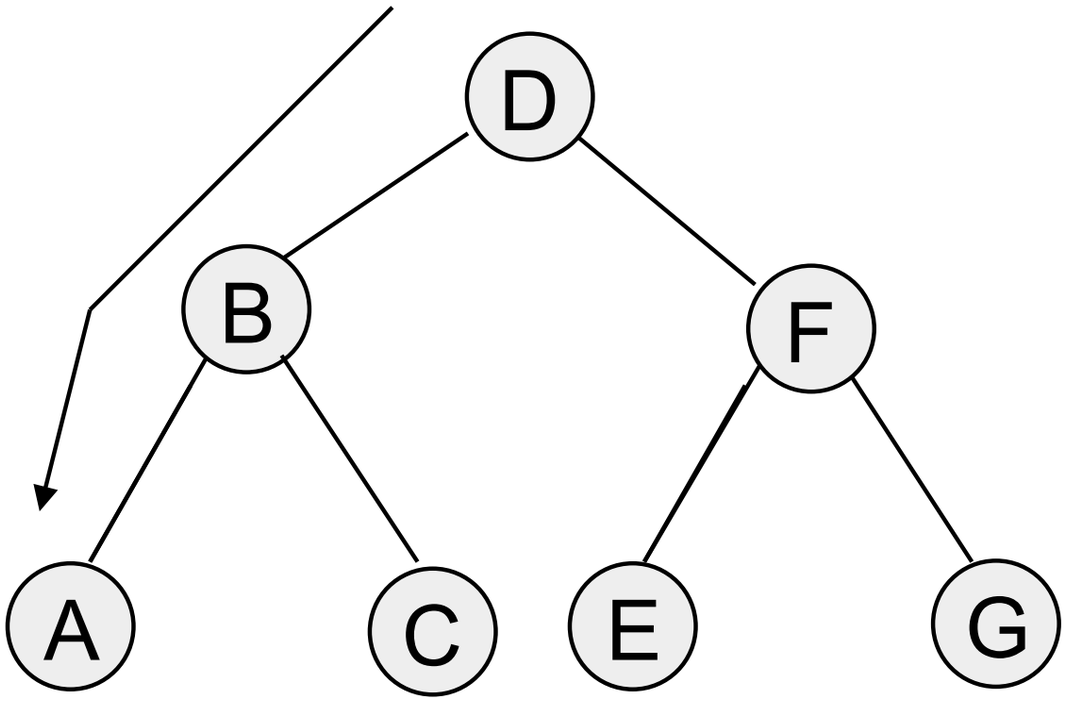


Order of nodes visited:

D B A C F E G

Visit the data
Visit the left subtree
Visit the right subtree

Inorder Traversal

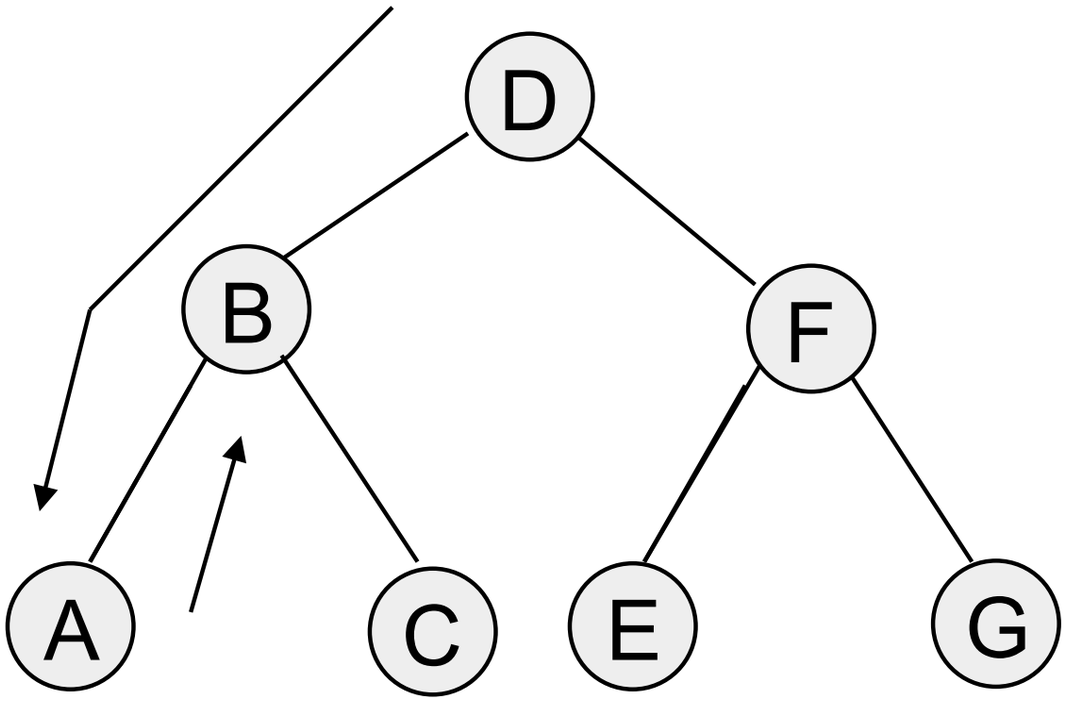


Order of nodes visited:

A

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

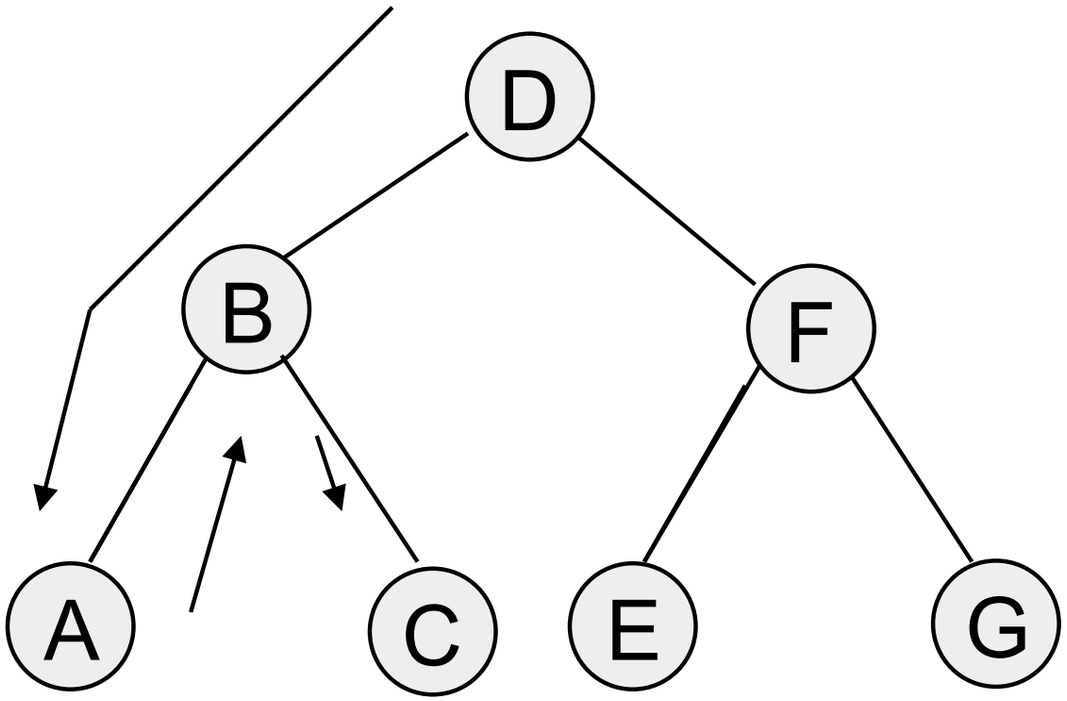


Order of nodes visited:

A B

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

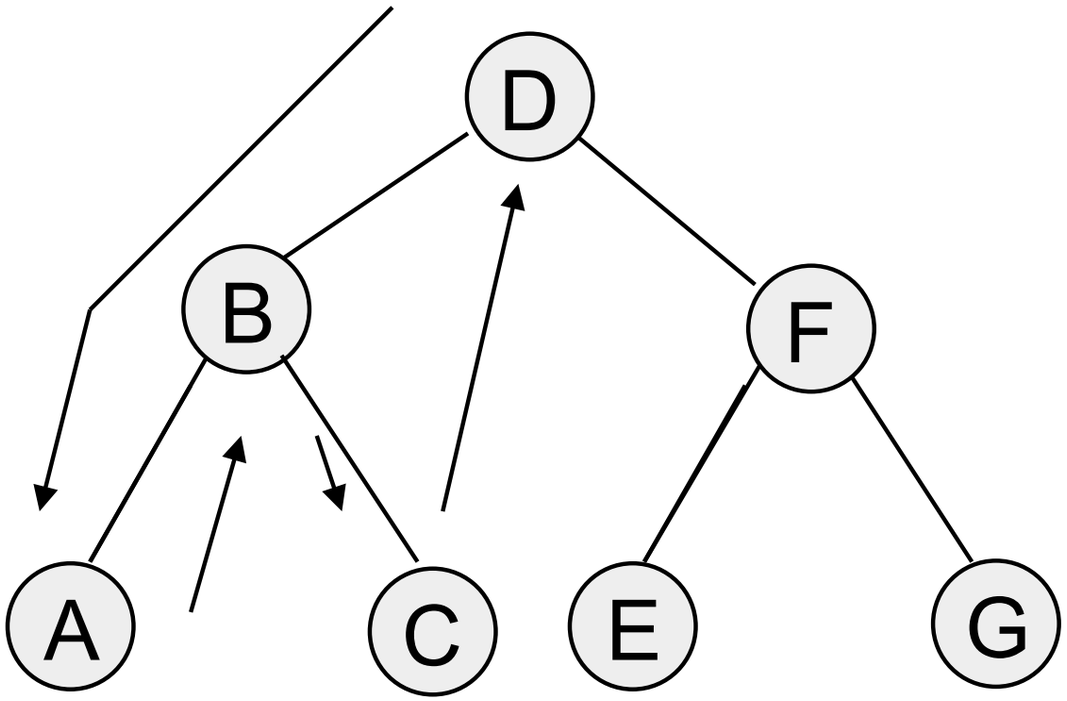


Order of nodes visited:

A B C

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

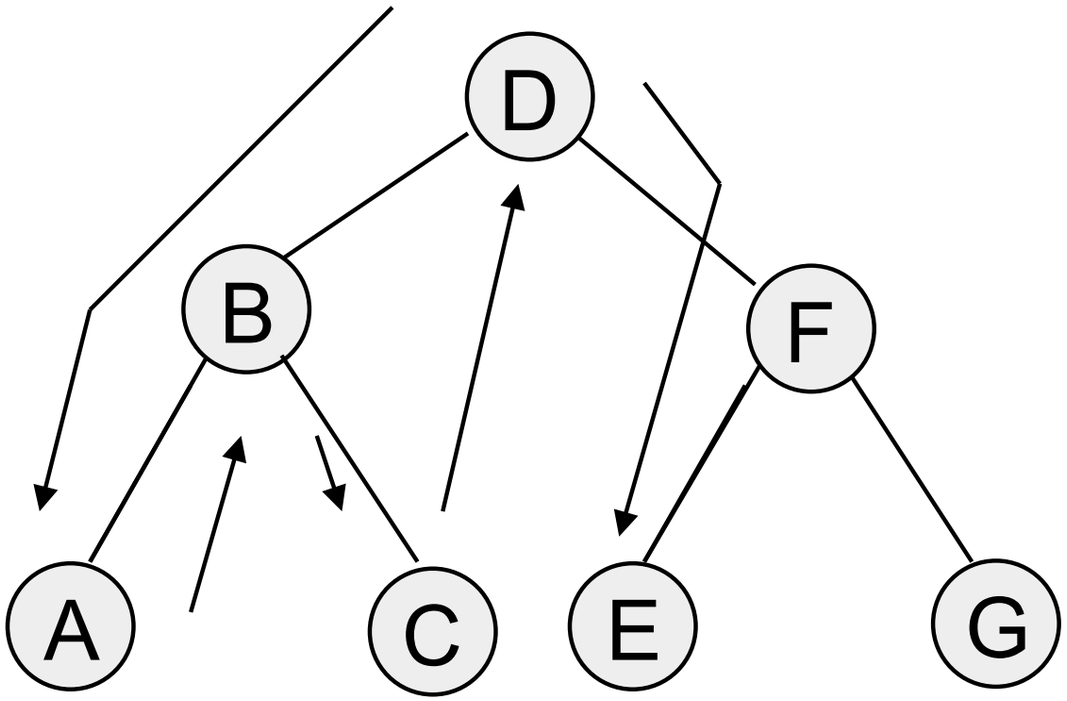


Order of nodes visited:

A B C D

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

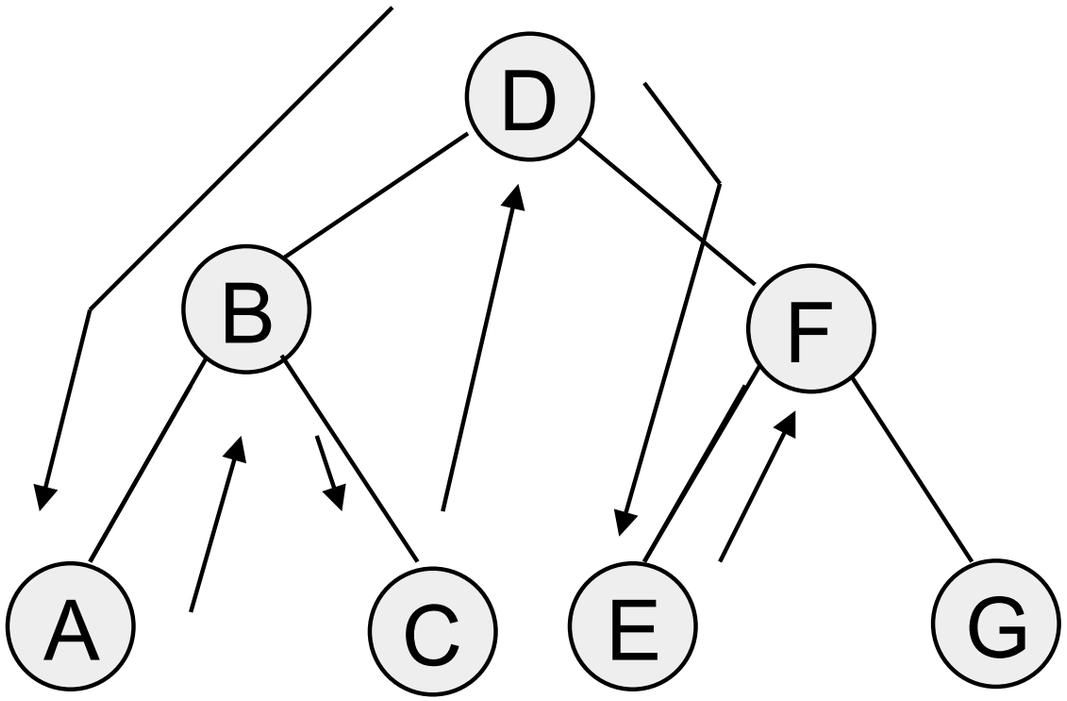


Order of nodes visited:

A B C D E

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

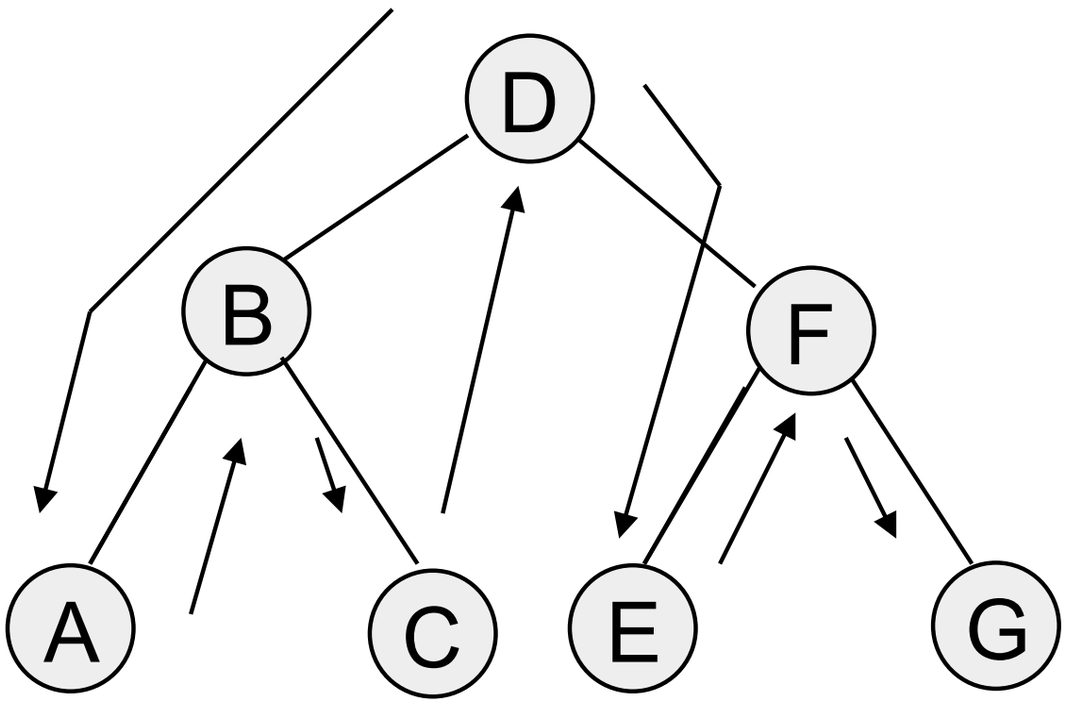


Order of nodes visited:

A B C D E F

Visit the left subtree
Visit the data
Visit the right subtree

Inorder Traversal

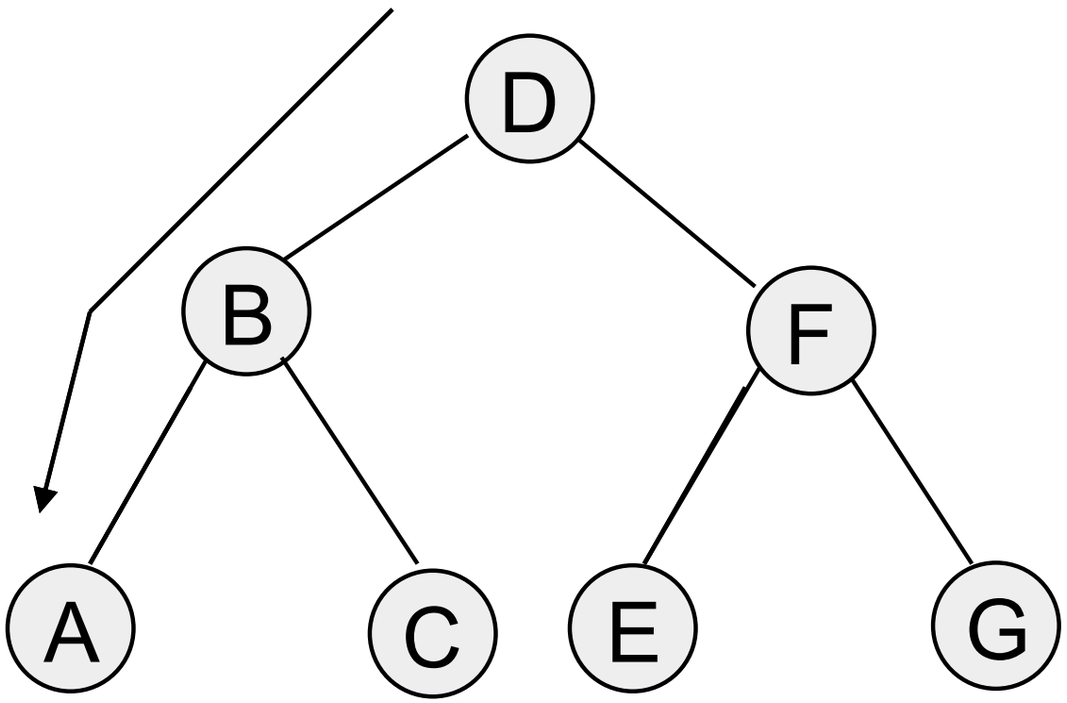


Order of nodes visited:

A B C D E F G

Visit the left subtree
Visit the data
Visit the right subtree

Postorder Traversal

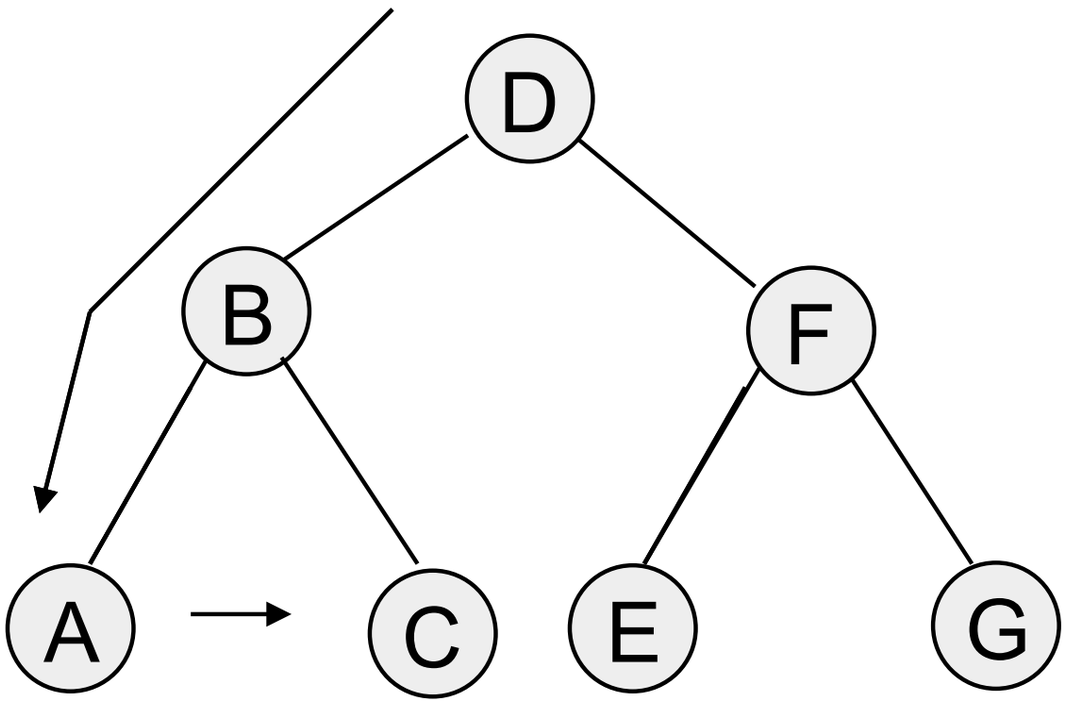


Order of nodes visited:

A

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal

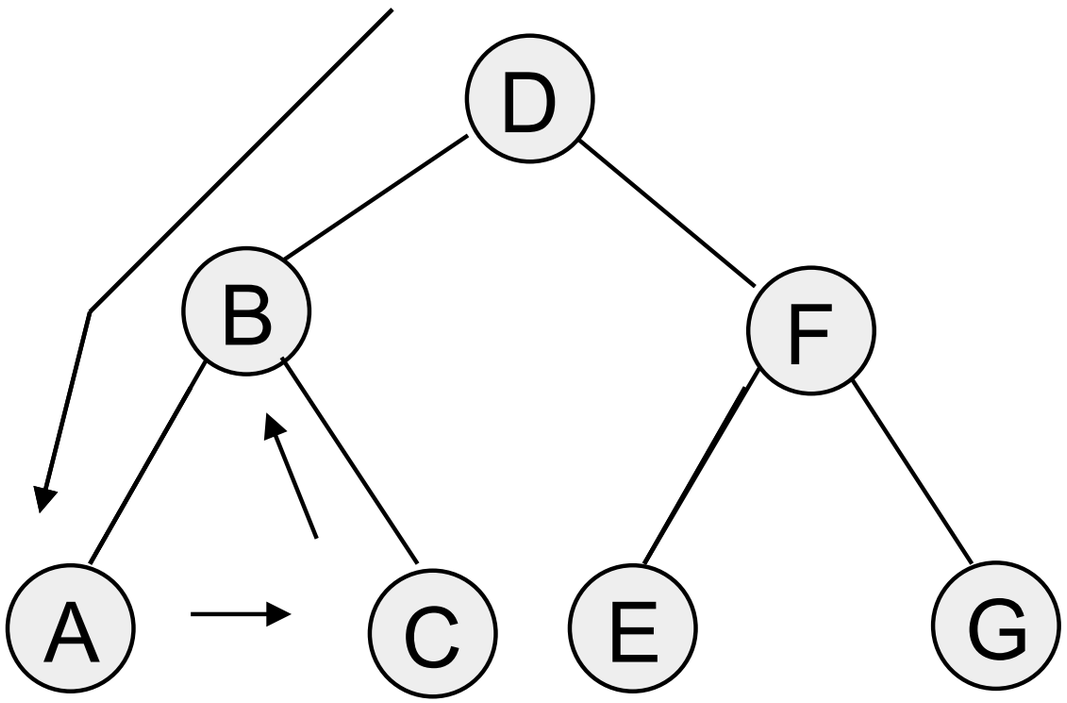


Order of nodes visited:

A C

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal

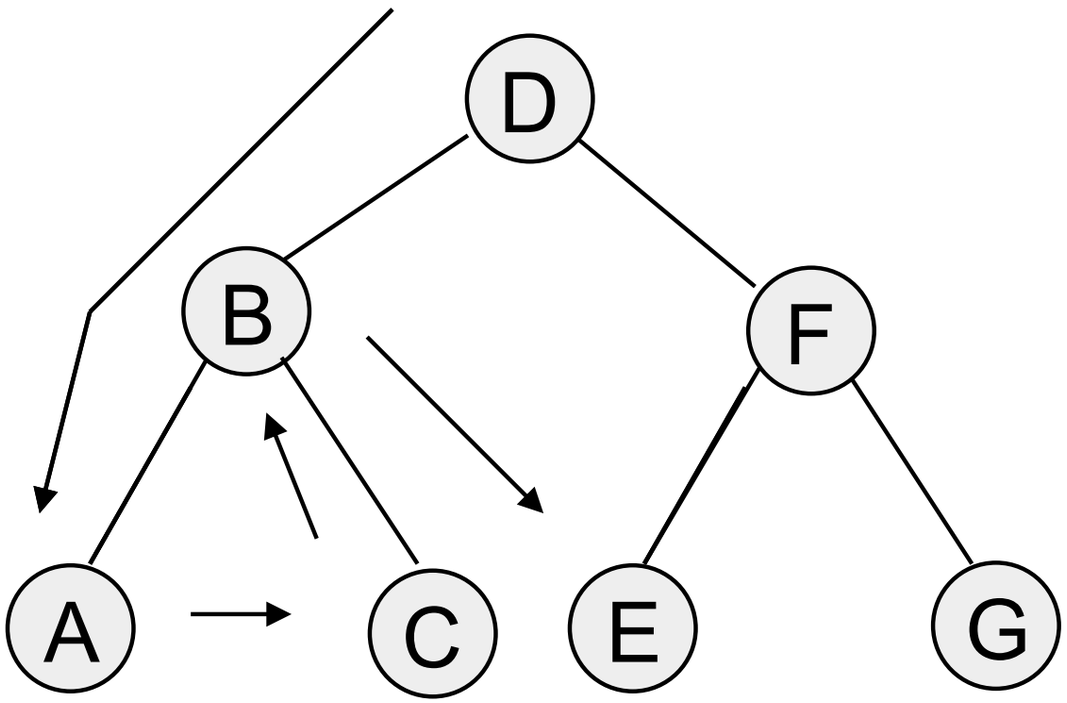


Order of nodes visited:

A C B

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal

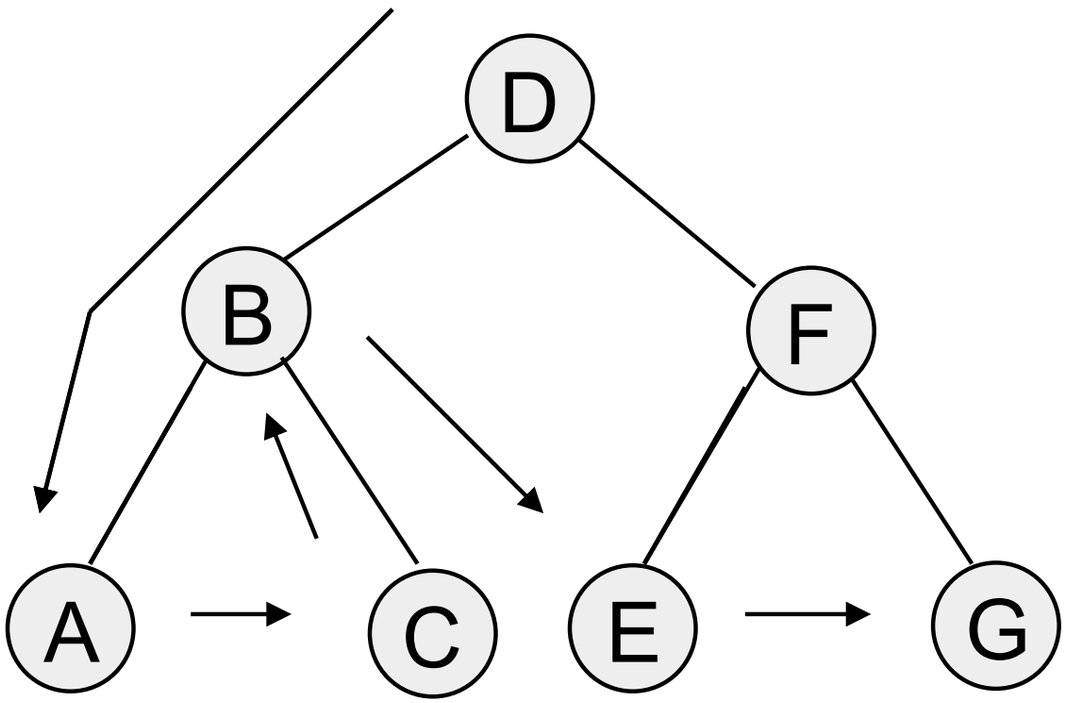


Order of nodes visited:

A C B E

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal

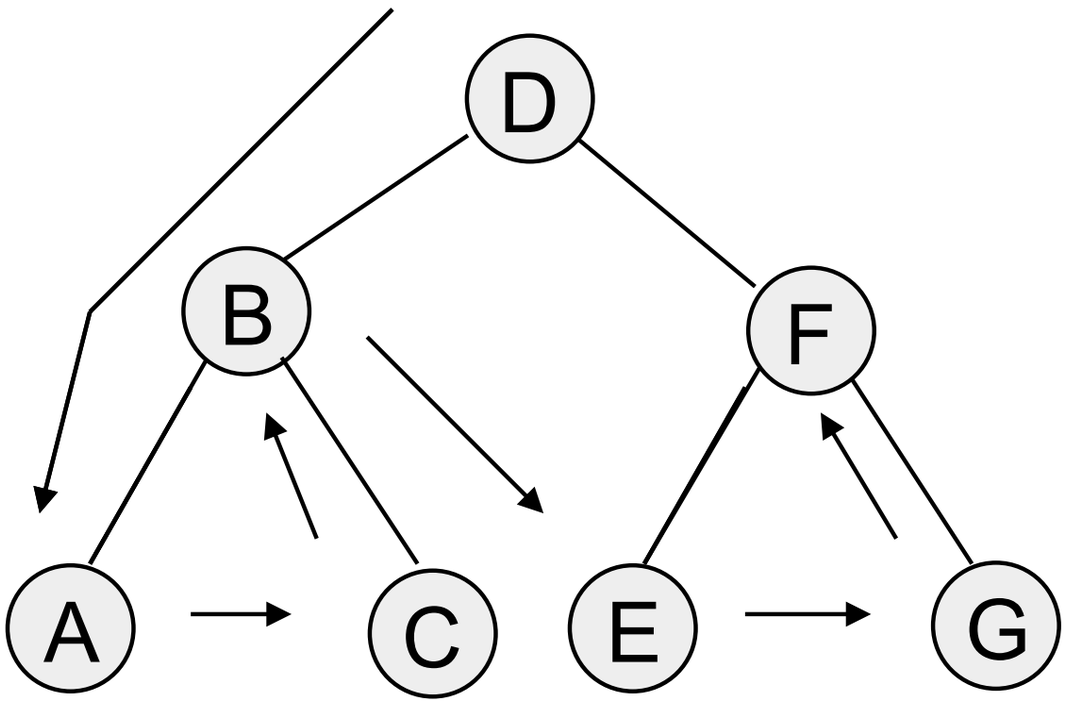


Order of nodes visited:

A C B E G

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal

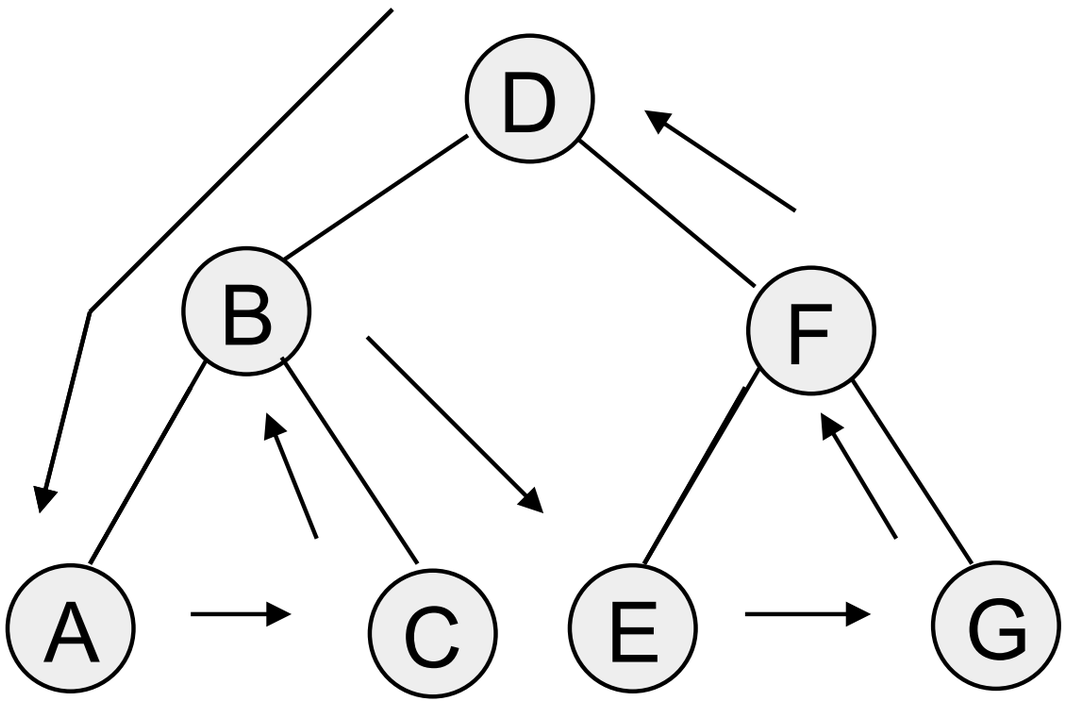


Order of nodes visited:

A C B E G F

Visit the left subtree
Visit the right subtree
Visit the data

Postorder Traversal



Order of nodes visited:

A C B E G F D

Visit the left subtree
Visit the right subtree
Visit the data

Comparator<T>

```
/**
 * A total ordering of T
 */
public interface Comparator<T>{
  /**
   * Compares its two arguments for order. Returns a negative
   *  integer, zero, or a positive integer as the first
   *  argument is less than, equal to, or greater than the
   *  second.
   * @param o1 the first object to be compared.
   * @param o2 the second object to be compared.
   * @return a negative integer, zero, or a positive integer
   *  as the first argument is less than, equal to, or greater
   *  than the second
   */
  public int compare(T o1, T o2);

  /**
   * Indicates whether some other object is "equal to" this
   *  comparator.
   * @param obj the reference object with which to compare.
   * @return whether specified object is also a comparator and it
   *  imposes the same ordering as this comparator
   */
  public boolean equals(Object obj);
}
```

http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html

```
/**
 * A comparator for Integer values.
 */
private static class UsualIntegerComparator implements Comparator<Integer> {
    /**
     * Compares its two arguments for order.
     * @param m first Integer to compare
     * @param n second Integer to compare
     * @return Returns a negative integer, zero, or a positive integer as m is
     *         less than, equal to, or greater than n
     */
    public int compare(Integer m, Integer n) {
        return m.compareTo(n);
    }
    /**
     * Is this <code>Comparator</code> same as the given object
     * @param o the given object
     * @return true if the given object is an instance of this class
     */
    public boolean equals(Object o) {
        return (o instanceof UsualIntegerComparator);
    }
    /**
     * There should be only one instance of this class = all are equal
     * @return the hash code same for all instances
     */
    public int hashCode() {
        return (this.toString().hashCode());
    }
    /**
     * @return name of class
     */
    public String toString() {
        return "UsualIntegerComparator";
    }
}
```

Assignment 5

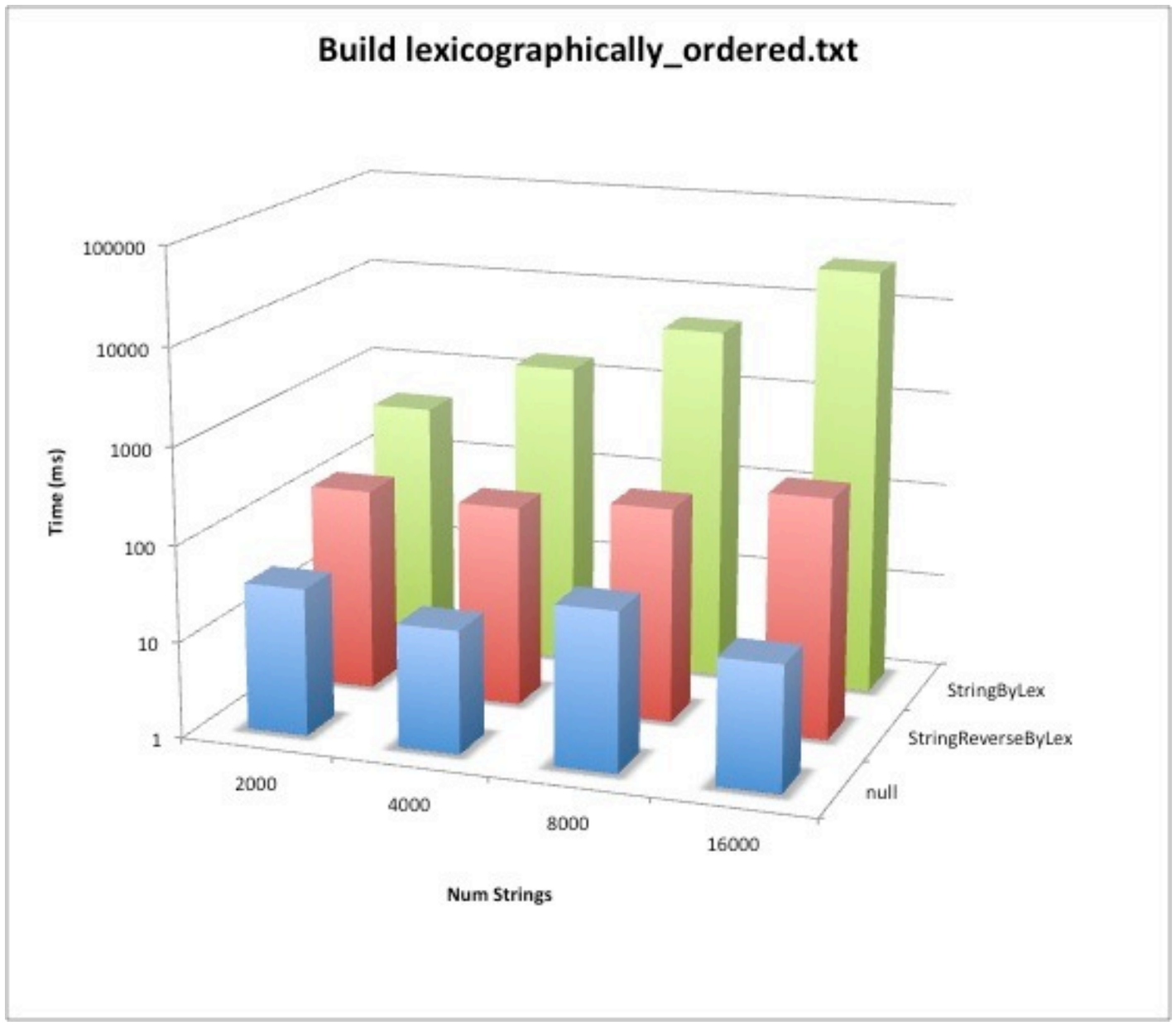
- Due Tuesday, February 11 at 11:59pm
- More efficient MyMap<K,V> using binary search tree

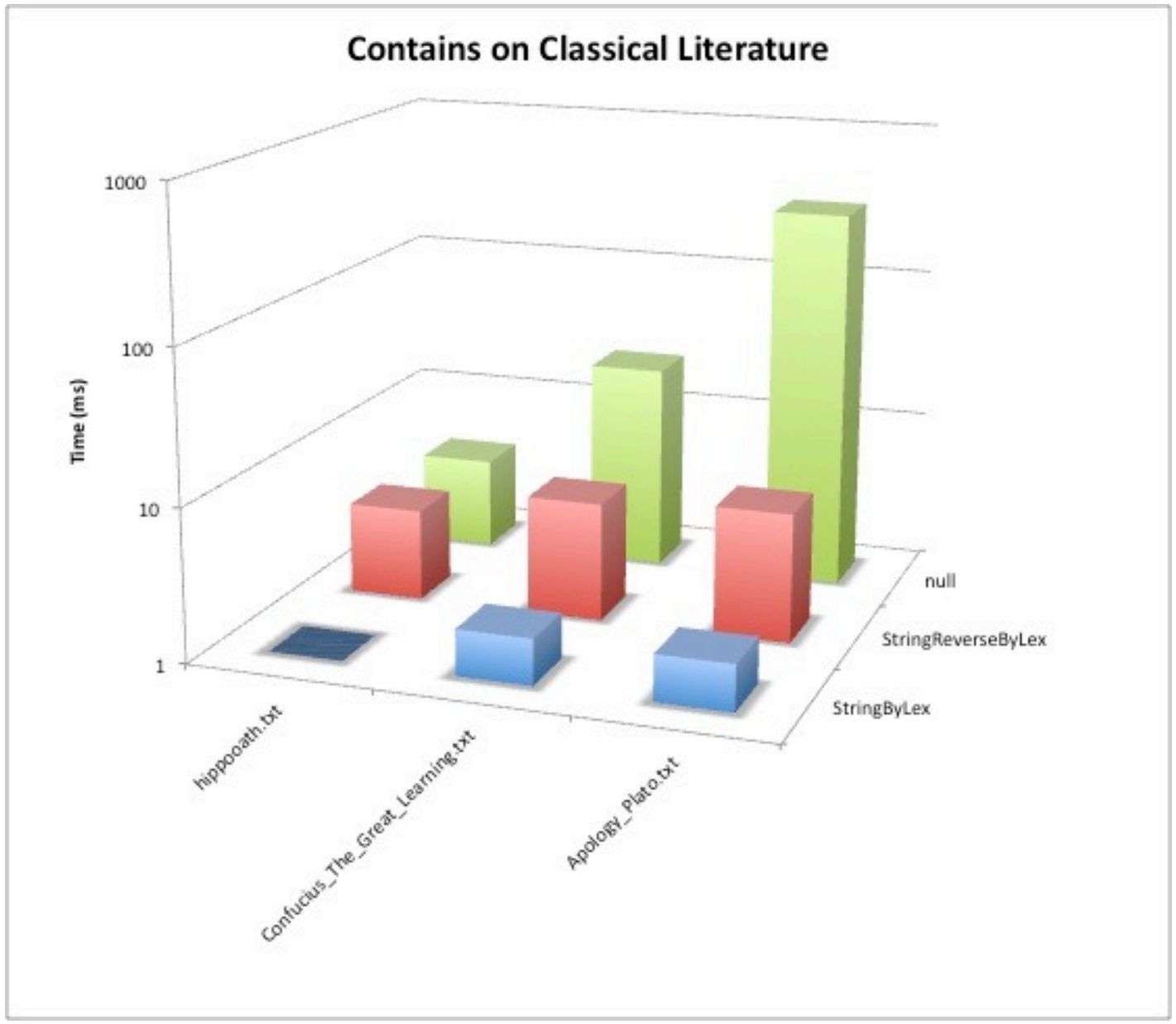
Algebraic Specs for BST

Assignment 6

- Timing program for MyMap<K,V>
- Due: Friday, February 14, 2014 at 11:59pm

	A	B	C	D	E	F	G	H	I
1	**Comparator**	**File**	**Num Strings**	**Size (#)**	**Build (ms)**	**Iterator (ms)**	**Iterate (ms)**	**Contains (ms)**	**Num Contained**
2	null	lexicographically_ordered.txt	2000	2000	34	80	2	45	77
3	null	lexicographically_ordered.txt	4000						
4	null	lexicographically_ordered.txt	8000						
5	null	lexicographically_ordered.txt	16000						
6	null	random_order.txt	2000						
7	null	random_order.txt	4000						
8	null	random_order.txt	8000						
9	null	random_order.txt	16000						
10	null	reverse_ordered.txt	2000						
11	null	reverse_ordered.txt	4000						
12	null	reverse_ordered.txt	8000						
13	null	reverse_ordered.txt	16000						
14	StringByLex	lexicographically_ordered.txt	2000						
15	StringByLex	lexicographically_ordered.txt	4000						
16	StringByLex	lexicographically_ordered.txt	8000						
17	StringByLex	lexicographically_ordered.txt	16000						
18	StringByLex	random_order.txt	2000						
19	StringByLex	random_order.txt	4000						
20	StringByLex	random_order.txt	8000						
21	StringByLex	random_order.txt	16000						
22	StringByLex	reverse_ordered.txt	2000						
23	StringByLex	reverse_ordered.txt	4000						
24	StringByLex	reverse_ordered.txt	8000						
25	StringByLex	reverse_ordered.txt	16000						
26	StringReverseByLex	lexicographically_ordered.txt	2000						
27	StringReverseByLex	lexicographically_ordered.txt	4000						
28	StringReverseByLex	lexicographically_ordered.txt	8000						
29	StringReverseByLex	lexicographically_ordered.txt	16000						
30	StringReverseByLex	random_order.txt	2000						
31	StringReverseByLex	random_order.txt	4000						
32	StringReverseByLex	random_order.txt	8000						
33	StringReverseByLex	random_order.txt	16000						
34	StringReverseByLex	reverse_ordered.txt	2000						
35	StringReverseByLex	reverse_ordered.txt	4000						
36	StringReverseByLex	reverse_ordered.txt	8000						
37	StringReverseByLex	reverse_ordered.txt	16000						
38									


Build lexicographically_ordered.txt
Time (ms)
100000
10000
1000
100
10
1
2000
4000
8000
16000
Num Strings
StringByLex
StringReverseByLex
null


Contains on Classical Literature
Time (ms)
1000
100
10
1
null
StringReverseByLex
StringByLex
hippocath.txt
Confucius_The_Great_Learning.txt
Apology_Plato.txt