CSC 402

Data Structures I
Lecture 7

Homework 5

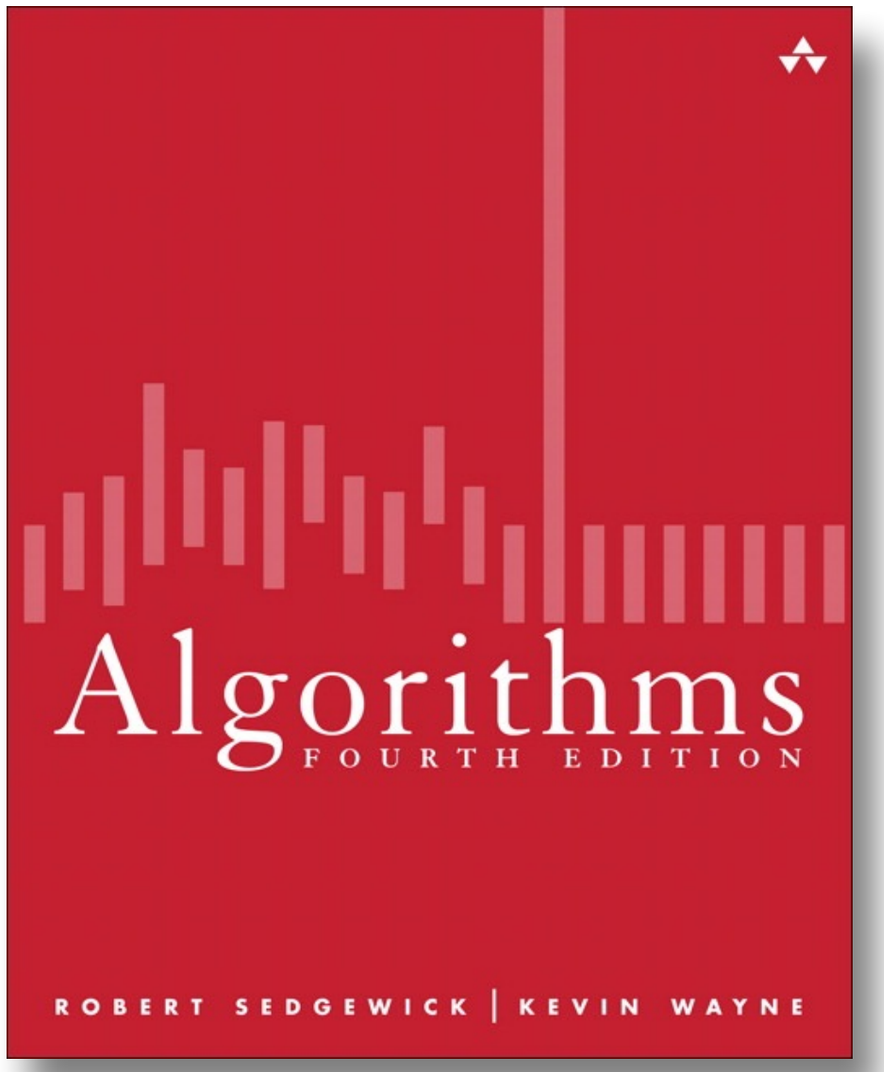
- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

- Also recognize an invalid entry

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```



Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end im
Exiting...
```

```
/*
 * The Contact class will have 3 properties:
 *      firstname   - String
 *      lastname    - String
 *      homestate   - String
 *      age         - Integer
 */

public class Contact {

}
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

- Also recognize an invalid entry

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

- Assignment - Write
have; using the da
at least 5 Contact
using Comparators
requirements:

- A prompt for the s
exit):

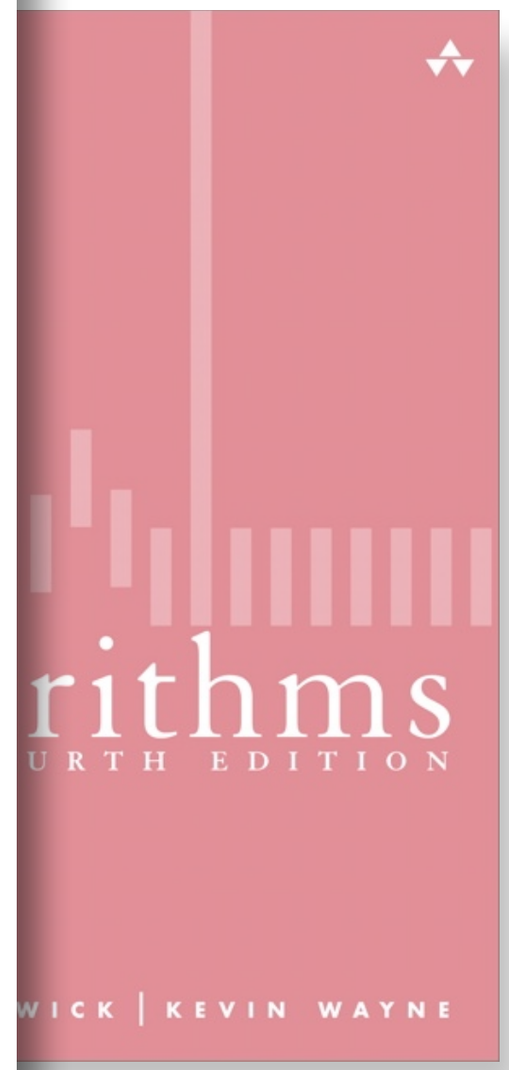
```
Sort by lastn
Sort by home
Sort by age
Enter option
Exiting...
```

- 1 = Sort by last na
- 2 = Sort by Home
- 3 = Sort by Age

- Also recognize an

```
Sort by lastr
Sort by home
Sort by age
Enter option
Invalid entry
```

```java
public class Contact {
    private String firstname;
    private String lastname;
    private String homestate;
    private Integer age;

    public Contact(String firstname, String lastname, String homestate,
            Integer age) {
        super();
        this.firstname = firstname;
        this.lastname = lastname;
        this.homestate = homestate;
        this.age = age;
    }

    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }
    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getHomestate() {
        return homestate;
    }
    public void setHomestate(String homestate) {
        this.homestate = homestate;
    }

    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Contact [firstname=" + firstname + ", lastname=" + lastname
                + ", homestate=" + homestate + ", age=" + age + "]";
    }

}
```



Homework 5

- Assignment - W
  have; using the
  at least 5 Conta
  using Compara
  requirements:

- A prompt for th
  exit):

```
Sort by l
Sort by h
Sort by a
Enter opt
Exiting..
```

- 1 = Sort by last
- 2 = Sort by Hor
- 3 = Sort by Age

- Also recognize

```
Sort by l
Sort by h
Sort by a
Enter opt
Invalid e
```

```java
import java.util.ArrayList;

public class TestSortOptions {

    public static void main(String[] args) {
        ArrayList<Contact> contacts = initializeContactsArray();
        promptForOption(contacts);
    }

    /*
     * Data Initialization
     */

    private static ArrayList<Contact> initializeContactsArray() {
        // TODO: Initialize an array of Student objects
        return null;
    }

    /*
     * Prompt for the user to enter their option from the keyboard
     *
     * 1 = Sort by last name
     * 2 = Sort by Home State
     * 3 = Sort by Age
     * 0 = End input and exit/terminate the application
     *
     */

    private static void promptForOption(ArrayList<Contact> contacts) {
        // TODO: Prompt and accept option input
    }

    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */

    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
    }

}
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the exit):
  - 1 = Sort by last
  - 2 = Sort by Home State
  - 3 = Sort by Age
- Also recognize an invalid entry

```
Sort by last
Sort by hom
Sort by age
Enter optio
Exiting...
```

```
Sort by lastname  :[1]
Sort by home state:[2]
Sort by age       :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

```
import java.util.ArrayList;

public class TestSortOptions {

    public static void main(String[] args) {
        ArrayList<Contact> contacts = initializeContactsArray();
        promptForOption(contacts);
    }

    /*
     * Data Initialization
     */

    private static ArrayList<Contact> initializeContactsArray() {
        // TODO: Initialize an array of Student objects
        ArrayList<Contact> contacts = new ArrayList<Contact>();

        contacts.add(new Contact("Joe", "Jones", "IL", new Integer(35)));
        contacts.add(new Contact("Bill", "Barnes", "OH",  new Integer(62)));
        contacts.add(new Contact("Adam", "Ant", "MI",  new Integer(14)));
        contacts.add(new Contact("Ida", "Know", "FL",  new Integer(23)));
        contacts.add(new Contact("Jane", "Doe", "CA",  new Integer(41)));

        return contacts;
    }

    // TODO: Prompt and accept option input
    }

    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */

    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
    }
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write
  have; using the dat
  at least 5 Contact o
  using Comparators
  requirements:
- A prompt for the so
  exit):

```
Sort by lastnar
Sort by home st
Sort by age
Enter option or
Exiting...
```

- 1 = Sort by last nam
- 2 = Sort by Home S
- 3 = Sort by Age

- Also recognize an i

```
Sort by lastnq
Sort by home s
Sort by age
Enter option o
Invalid entry.
```

```
/*
 * Prompt for the user to enter their option from the keyboard
 *
 * 1 = Sort by last name
 * 2 = Sort by Home State
 * 3 = Sort by Age
 * 0 = End input and exit/terminate the application
 *
 */

private static void promptForOption(ArrayList<Contact> contacts) {
    // TODO: Prompt and accept option input
    Scanner console = new Scanner(System.in);

    System.out.println("Sort by lastname   :[1]");
    System.out.println("Sort by home state :[2]");
    System.out.println("Sort by age        :[3]");
    System.out.print("Enter option or 0 to end input: ");
    int inputValue = console.nextInt();

    switch(inputValue) {
        case 0:
            System.out.println("Exiting...");
            System.exit(0);
            break;

        case 1:
            sortByLastName(contacts);
            break;

        case 2:
            sortByHomeState(contacts);
            break;

        case 3:
            sortByAge(contacts);
            break;

        default:
            System.out.println("Invalid entry.");
            break;
    }
}
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:

- A prompt for the exit):

```
Sort by last
Sort by hom
Sort by age
Enter optio
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

- Also recognize an invalid entry

```
Sort by lastname  :[1]
Sort by home state:[2]
Sort by age       :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

```
import java.util.ArrayList;

public class TestSortOptions {

    public static void main(String[] args) {
        ArrayList<Contact> contacts = initializeContactsArray();
        promptForOption(contacts);
    }
    /*
     * Data Initialization
     */

    private static ArrayList<Contact> initializeContactsArray() {
        // TODO: Initialize an array of Student objects
```

```
    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */

    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
        for(Contact contact : contacts) {
            System.out.println(contact);
        }
    }
```

```
    private static void promptForOption(ArrayList<Contact> contacts) {
        // TODO: Prompt and accept option input
    }

    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */

    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
    }
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the so
exit):

```
Sort by lastnam
Sort by home st
Sort by age
Enter option or
Exiting...
```

- 1 = Sort by last nam
- 2 = Sort by Home S
- 3 = Sort by Age

- Also recognize an invalid entry

```
Sort by lastname  :[1]
Sort by home state:[2]
Sort by age       :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

```
import java.util.ArrayList;

public class TestSortOptions {

    public static void main(String[] args) {
        ArrayList<Contact> contacts = initializeContactsArray();
        promptForOption(contacts);
    }

    /*
     * Data Initialization
     */

    /*
     * Sorting Options
     */
    private static void sortByLastName(ArrayList<Contact> contacts) {
        Collections.sort(contacts, new LastNameComparator());
        displayContacts(contacts);
    }

    private static void sortByHomeState(ArrayList<Contact> contacts) {
        Collections.sort(contacts, new HomeStateComparator());
        displayContacts(contacts);
    }

    private static void sortByAge(ArrayList<Contact> contacts) {
        Collections.sort(contacts, new AgeComparator());
        displayContacts(contacts);
    }

    /*
     * Display the Contact information sorted by using the selected option from
     * the above "promptForOption" method result
     */

    private static void displayContacts(ArrayList<Contact> contacts) {
        // TODO: Display the contents of the Contacts Array
    }
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 1
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
```

- Also recognize an invalid entry

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. T
requirements:

- A prompt for the sort
exit):

```
Sort by lastname
Sort by home sta
Sort by age
Enter option or 0}to end imput: 0
Exiting...
```

```
import java.util.Comparator;

public class LastNameComparator implements Comparator<Contact> {

    @Override
    public int compare(Contact contact1, Contact contact2) {
        return contact1.getLastname().compareTo(contact2.getLastname());
    }

}
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 1
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
```

- Also recognize an invalid entry

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 2
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
```

- Also recognize an invali

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort (1, 2, 3 or 0 to exit):

```
Sort by lastname
Sort by home stat
Sort by age
Enter option or 0
Exiting...
```

- 1 = Sort by last name
- 2 = Sort by Home State
- 3 = Sort by Age

- Also recognize an invali

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

```
import java.util.Comparator;

public class HomeStateComparator implements Comparator<Contact> {

    @Override
    public int compare(Contact contact1, Contact contact2) {
        return contact1.getHomestate().compareTo(contact2.getHomestate());
    }

}
```

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 2
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the sort by options (with 0 being exit):

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 0
Exiting...
```

- 1 = Sort by last n
- 2 = Sort by Hom
- 3 = Sort by Age

```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 3
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
```

- Also recognize a

```
Sort by la
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

Homework 5

- Assignment - Write an application that will have; using the data structure of your choice, at least 5 Contact objects that will be sorted using Comparators. These are the following requirements:
- A prompt for the s
exit):

```
Sort by lastn
Sort by home
Sort by age
Enter option
Exiting...
```

```
import java.util.Comparator;

public class AgeComparator implements Comparator<Contact> {

    @Override
    public int compare(Contact contact1, Contact contact2) {
        return contact1.getAge().compareTo(contact2.getAge());
    }

}
```

- 1 = Sort by last
- 2 = Sort by Hom
- 3 = Sort by Age

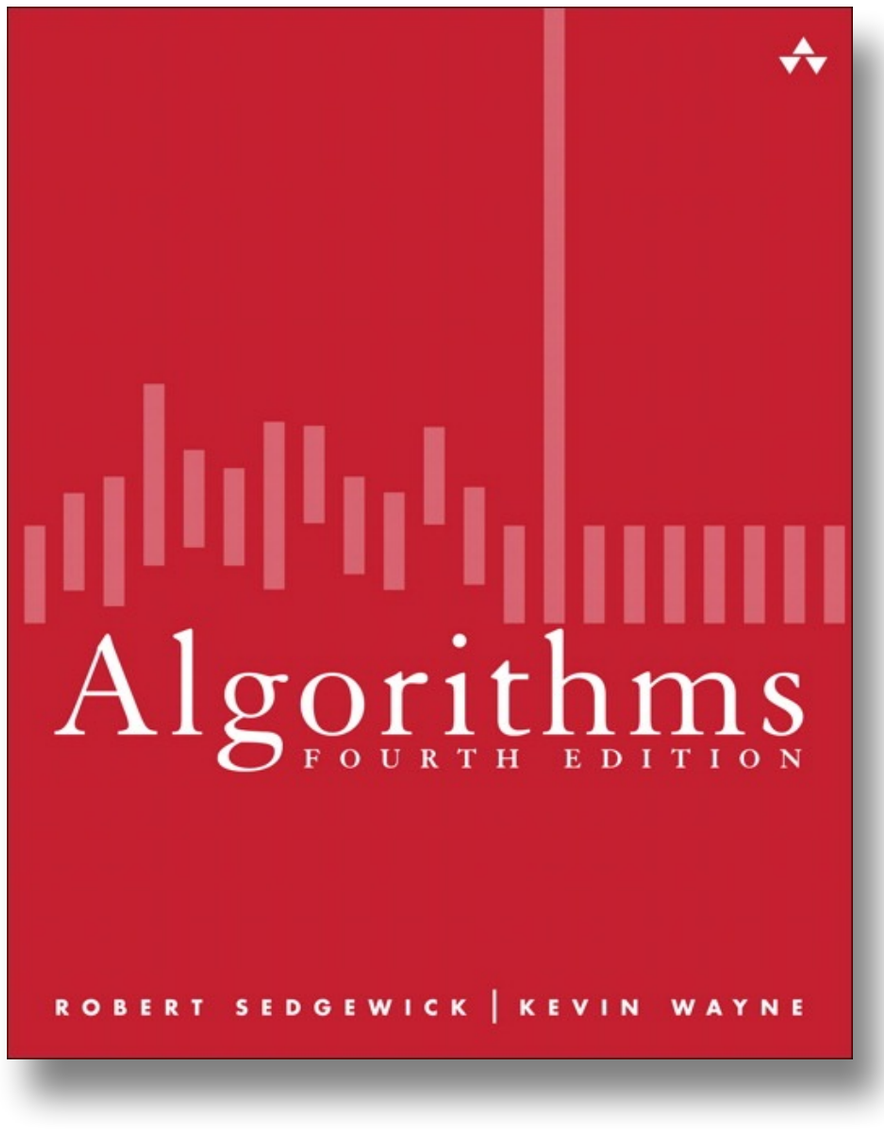
```
Sort by lastname   :[1]
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 3
Contact [firstname=Adam, lastname=Ant, homestate=MI, age=14]
Contact [firstname=Ida, lastname=Know, homestate=FL, age=23]
Contact [firstname=Joe, lastname=Jones, homestate=IL, age=35]
Contact [firstname=Jane, lastname=Doe, homestate=CA, age=41]
Contact [firstname=Bill, lastname=Barnes, homestate=OH, age=62]
```

- Also recognize a

```
Sort by la
Sort by home state :[2]
Sort by age        :[3]
Enter option or 0 to end imput: 4
Invalid entry.
```

Assignment due next Monday at 11:59 PM

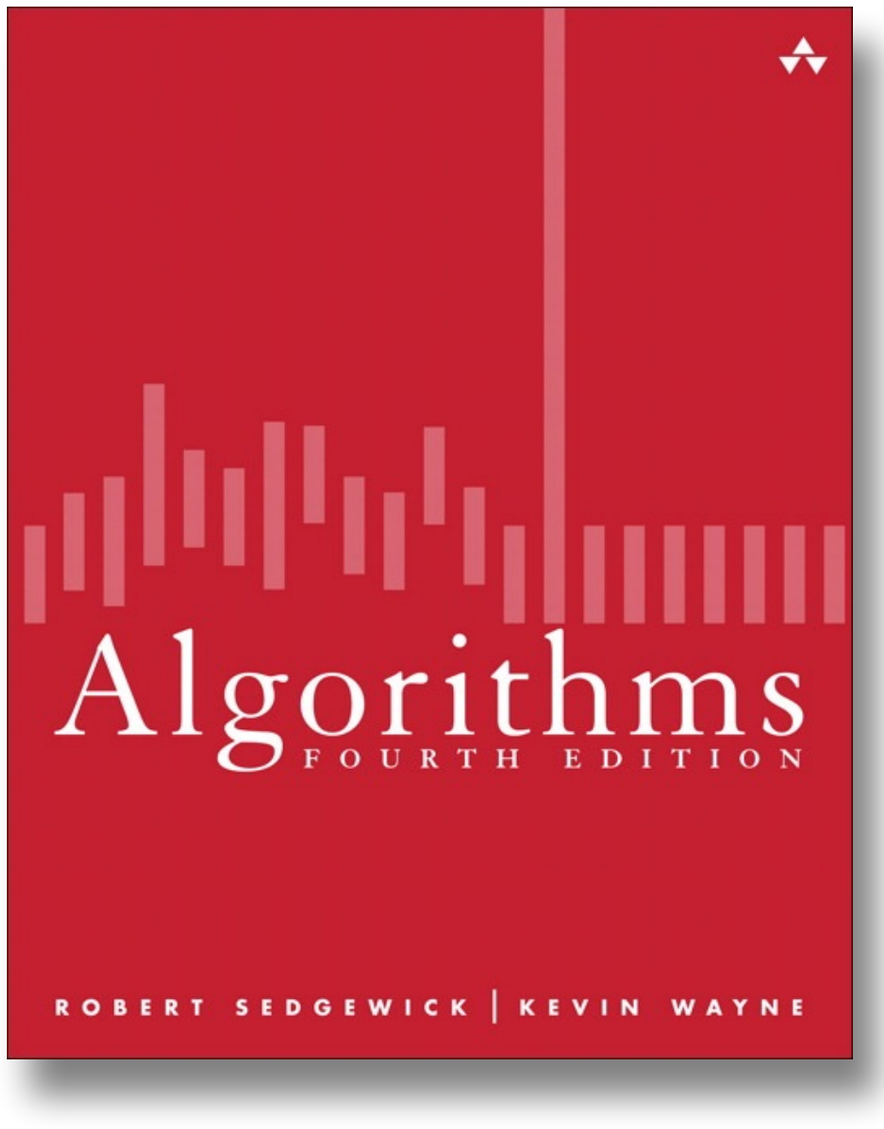
Lecture Overview



3.1 Symbol Tables

- API
- Elementary Implementations
- Ordered Operations

Lecture Overview



3.1 Symbol Tables

- API
- Elementary Implementations
- Ordered Operations

Symbol Tables

- **Insert** a value with specified key.
- Given a key, **search** for the corresponding value.

Ex. DNS lookup.

- Insert domain name with specified IP address.
- Given domain name, find corresponding IP address.

domain name	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60
↑ key	↑ value

Symbol Table Applications

application	purpose of search	key	value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	type and value
routing table	route Internet packets	destination	best route
DNS	find IP address	domain name	IP address
reverse DNS	find domain name	IP address	domain name
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Symbol Tables: Context

Also known as: maps, dictionaries, associative arrays

Language support:.

- External libraries: C, Visual Basic, Standard ML, bash, …
- Built-in libraries: Java, C#, C++, Scala, …
- Built-in to language: Awk, Perl, PHP, Tcl, JavaScript, Python, Ruby, Lua

every array is an associative array

every object is an associative array

table is the only primitive data structure

```
hasNiceSyntaxForAssociativeArrays["Python"] = true
hasNiceSyntaxForAssociativeArrays["Java"]   = false
```

legal Python code

Symbol Tables: Context

Associative array abstraction: Associate one value with each key

public class ST<Key, Value>		
	ST()	*create an empty symbol table*
void	put(Key key, Value val)	*put key-value pair into the table* ← a[key] = val;
Value	get(Key key)	*value paired with key* ← a[key]
boolean	contains(Key key)	*is there a value paired with key?*
void	delete(Key key)	*remove key (and its value) from table*
boolean	isEmpty()	*is the table empty?*
int	size()	*number of key-value pairs in the table*
Iterable<Key>	keys()	*all the keys in the table*

Symbol Tables

- Values are not null. ⟵ ***Java allows null value***
- Method **get()** returns **null** if key not present.
- Method **put()** overwrites old value with new value.

Intended consequences.

- Easy to implement **contains()**

```
public boolean contains(Key key) {
      return get(key) != null;
}
```

- Can implement lazy version of **delete()**

```
public void delete(Key key) {
      put(key, null);
}
```

Keys and Values

Value type: Any generic type

Key type: several natural assumptions

specify *Comparable* in API

- Assume keys are ***Comparable***, use **compareTo()**.
- Assume keys are any generic type, use **equals()** to test equality.
- Assume keys are any generic type, use **equals()** to test equality; use **hashCode()** to scramble key

built-in to Java

Best practices: Use immutable types for symbol table keys.

- Immutable in Java: Integer, Double, String, java.io.File, …
- Mutable in Java: StringBuilder, java.net.URL, arrays, …

Equality Test

All Java classes inherit a method equals()

Java requirements: For any references x, y and z

- Reflexive: x.equals(x) **is** true
- Symmetric: x.equals(y) **iff** y.equals(x)
- Transitive: **if** x.equals(y) **and** y.equals(z), **then** x.equals(z)
- Non-null: x.equals(null) **is** false

equivalence relation

- Default implementation (x == y)
- Customized implementations. Integer, Double, String, java,io.File, …
- User-defined implementations. Some care needed

Implementing equals for user-defined types

Seems easy

```
public       class Date implements Comparable<Date>
{
   private final int month;
   private final int day;
   private final int year;
   ...

   public boolean equals(Date that)
   {
      if (this.day   != that.day  ) return false;
      if (this.month != that.month) return false;
      if (this.year  != that.year ) return false;
      return true;
   }
}
```

check that all significant fields are the same

Implementing equals for user-defined types

Seems easy, but requires some care

typically unsafe to use equals() with inheritance (would violate symmetry)

```
public final class Date implements Comparable<Date>
{
   private final int month;
   private final int day;
   private final int year;
   ...

   public boolean equals(Object y)
   {
      if (y == this) return true;

      if (y == null) return false;

      if (y.getClass() != this.getClass())
         return false;

      Date that = (Date) y;
      if (this.day   != that.day  ) return false;
      if (this.month != that.month) return false;
      if (this.year  != that.year ) return false;
      return true;
   }
}
```

must be Object.
Why? Experts still debate.

optimize for true object equality

check for null

objects must be in the same class
(religion: getClass() vs. instanceof)

cast is guaranteed to succeed

check that all significant fields are the same

Equals Design

"Standard" recipe for user-defined types

- Optimization for reference equality
- Check against **null**
- Check that two objects are of the same type and cast
- Compare each significant field:
  - if field is a primitive type, use ==
  - if field is an object, use equals()
  - if field is an array, apply to each entry

but use Double.compare() with double (or otherwise deal with -0.0 and NaN)

apply rule recursively

can use Arrays.deepEquals(a, b) but not a.equals(b)

Best practices:

e.g., cached Manhattan distance

- No need to use calculated fields that depend on other fields
- Compare fields mostly likely to differ first
- Make compareTo() consistent with equals()

x.equals(y) if and only if (x.compareTo(y) == 0)

ST Test Client for Traces

Build ST by associating value i with i^{th} string from standard input

```
public static void main(String[] args) {
        ST<String, Integer> st = new ST<String, Integer>();
        for(int i=0; !StdIn.isEmpty(); i++) {
                String key = StdIn.readString();
                st.put(key, i);
        }
        for (String s : st.keys())
                StdOut.println(s + " " + st.get(s));
}
```

keys	S	E	A	R	C	H	E	X	A	M	P	L	E
values	0	1	2	3	4	5	6	7	8	9	10	11	12

output

```
A 8
C 4
E 12
H 5
L 11
M 9
P 10
R 3
S 0
X 7
```

ST Test Client for Analysis

Frequency counter: Read a sequence of strings from standard input and print out one that occurs with highest frequency.

```
% more tinyTale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
it was the spring of hope
it was the winter of despair

% java FrequencyCounter 1 < tinyTale.txt
it 10

% java FrequencyCounter 8 < tale.txt
business 122

% java FrequencyCounter 10 < leipzig1M.txt
government 24763
```

tiny example
(60 words, 20 distinct)

real example
(135,635 words, 10,769 distinct)

real example
(21,191,455 words, 534,580 distinct)

Frequency Counter Implementation

```
public class FrequencyCounter
{
   public static void main(String[] args)
   {
      int minlen = Integer.parseInt(args[0]);
      ST<String, Integer> st = new ST<String, Integer>();
      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (word.length() < minlen) continue;
         if (!st.contains(word)) st.put(word, 1);
         else                    st.put(word, st.get(word) + 1);
      }
      String max = "";
      st.put(max, 0);
      for (String word : st.keys())
         if (st.get(word) > st.get(max))
            max = word;
      StdOut.println(max + " " + st.get(max));
   }
}
```

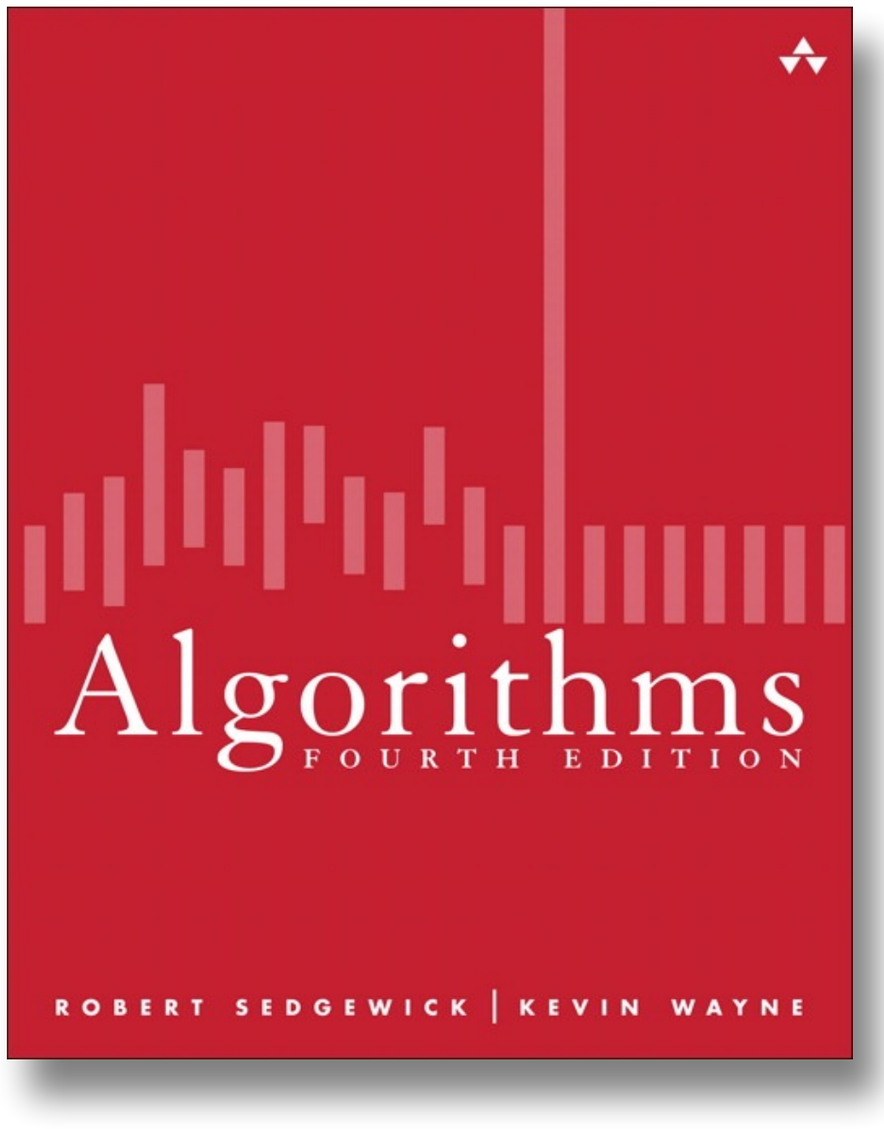
create ST

ignore short strings

read string and update frequency

print a string with max freq

Lecture Overview



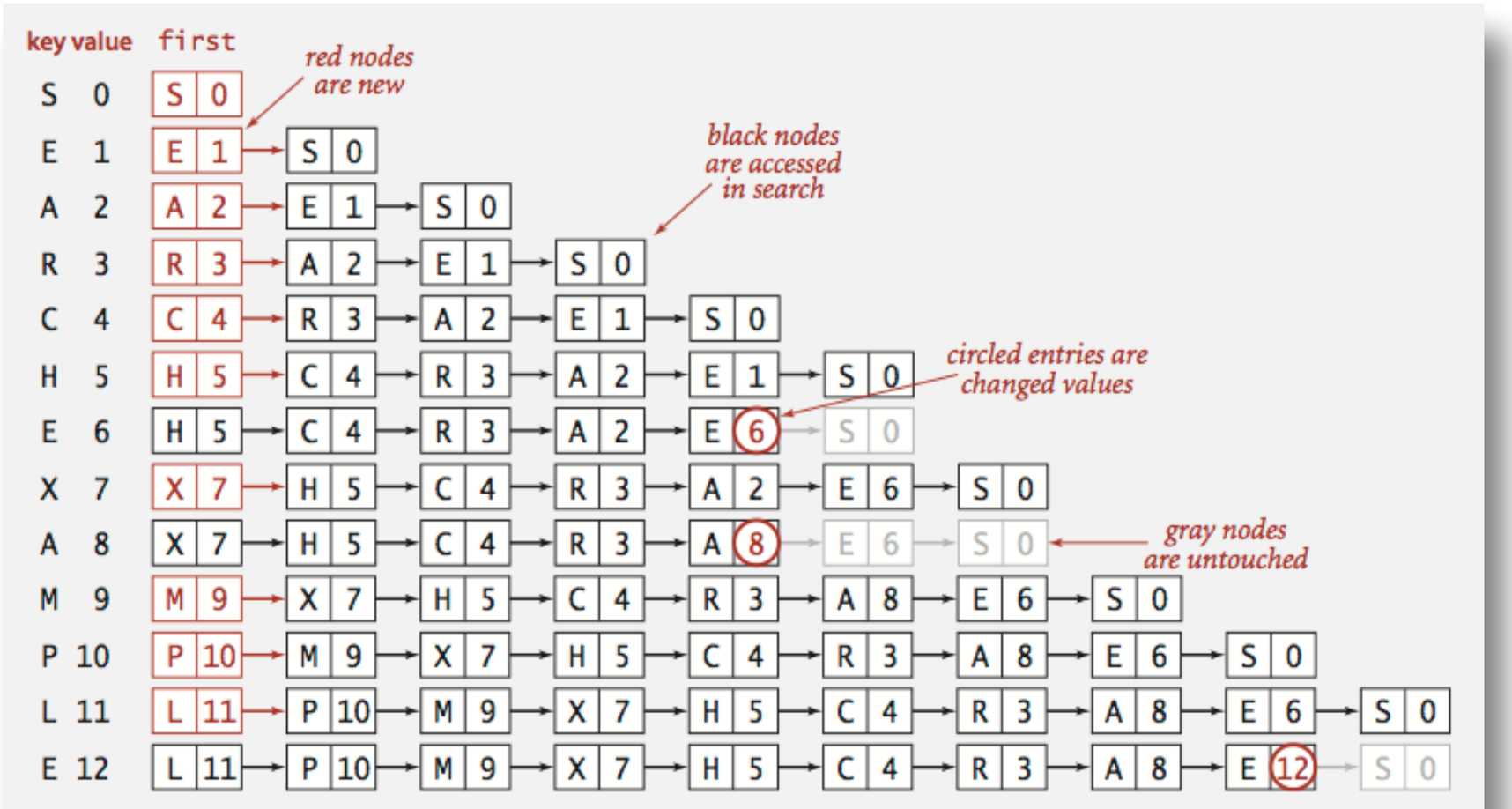
3.1 Symbol Tables

- API
- Elementary Implementations
- Ordered Operations

Sequential Search in a Linked List

Data structure: Maintain an (unordered) linked list of key-value pairs.

Search: Scan through all keys until find a match.
Insert: Scan through all keys until find a match; if no match add to front.



Trace of linked-list ST implementation for standard indexing client

Elementary ST Implementations: Summary

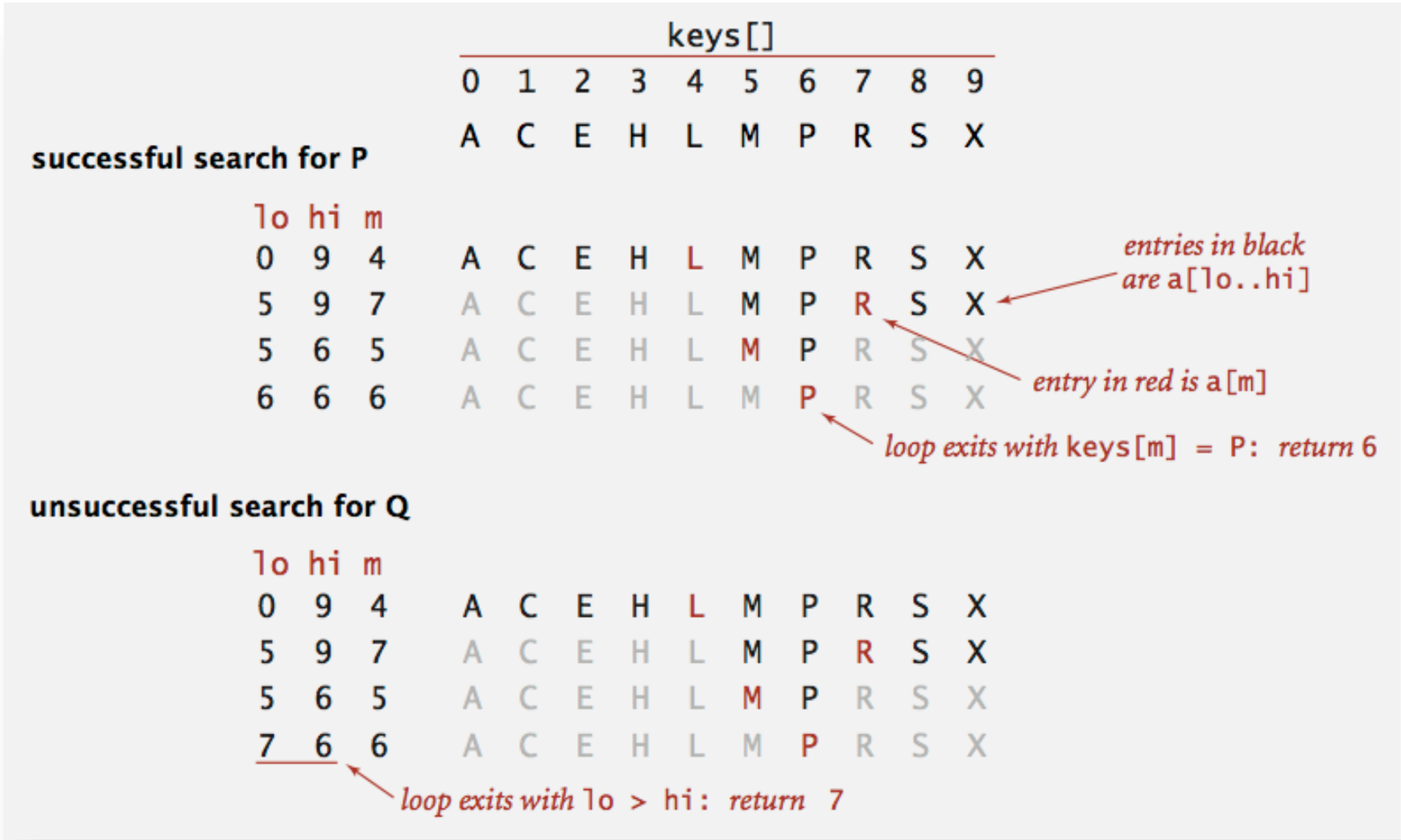
ST implementation	guarantee		average case		key interface
	search	insert	search hit	insert	
sequential search (unordered list)	N	N	$N / 2$	N	`equals()`

Challenge: Efficient implementations of both search and insert.

Binary Search in an Ordered Array

Data structure: Maintain an ordered array of key-value pairs.

Rank helper function: How many keys < k ?



Binary Search: Java Implementation

```
public Value get(Key key)
{
   if (isEmpty()) return null;
   int i = rank(key);
   if (i < N && keys[i].compareTo(key) == 0) return vals[i];
   else return null;
}
```

```
private int rank(Key key)                          number of keys < key
{
   int lo = 0, hi = N-1;
   while (lo <= hi)
   {
       int mid = lo + (hi - lo) / 2;
       int cmp = key.compareTo(keys[mid]);
       if      (cmp  < 0) hi = mid - 1;
       else if (cmp  > 0) lo = mid + 1;
       else if (cmp == 0) return mid;
   }
   return lo;
}
```

Binary Search: Trace of Standard Indexing Client

Problem: To insert, need to shift all greater keys over.

key	value	keys[] 0	1	2	3	4	5	6	7	8	9	N	vals[] 0	1	2	3	4	5	6	7	8	9
S	0	S										1	0									
E	1	E	S									2	1	0								
A	2	A	E	S								3	2	1	0							
R	3	A	E	R	S							4	2	1	3	0						
C	4	A	C	E	R	S						5	2	4	1	3	0					
H	5	A	C	E	H	R	S					6	2	4	1	5	3	0				
E	6	A	C	E	H	R	S					6	2	4	(6)	5	3	0				
X	7	A	C	E	H	R	S	X				7	2	4	6	5	3	0	7			
A	8	A	C	E	H	R	S	X				7	(8)	4	6	5	3	0	7			
M	9	A	C	E	H	M	R	S	X			8	8	4	6	5	9	3	0	7		
P	10	A	C	E	H	M	P	R	S	X		9	8	4	6	5	9	10	3	0	7	
L	11	A	C	E	H	L	M	P	R	S	X	10	8	4	6	5	11	9	10	3	0	7
E	12	A	C	E	H	L	M	P	R	S	X	10	8	4	(12)	5	11	9	10	3	0	7
		A	C	E	H	L	M	P	R	S	X		8	4	12	5	11	9	10	3	0	7

entries in red were inserted

entries in black moved to the right

entries in gray did not move

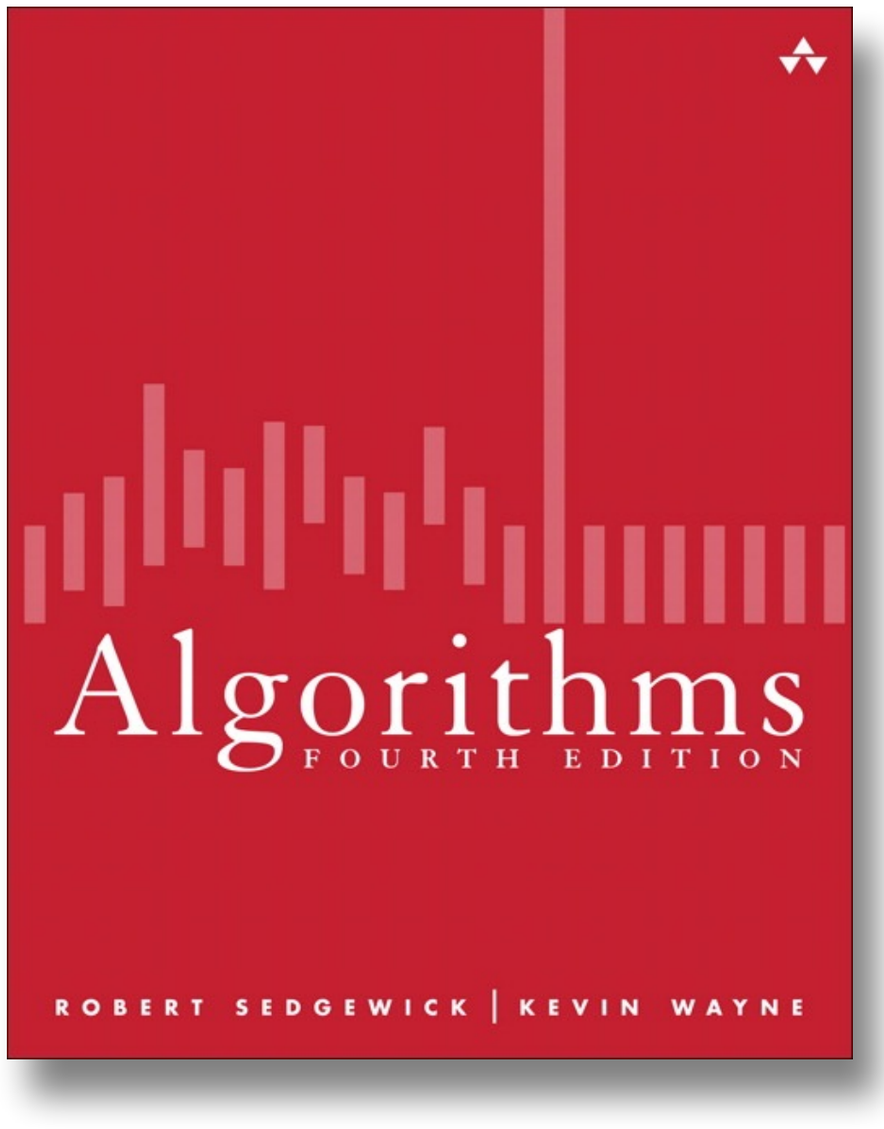
circled entries are changed values

Elementary ST Implementations: Summary

ST implementation	guarantee		average case		key interface
	search	insert	search hit	insert	
sequential search (unordered list)	N	N	$N / 2$	N	`equals()`
binary search (ordered array)	$\log N$	N	$\log N$	$N / 2$	`compareTo()`

Challenge: Efficient implementations of both search and insert.

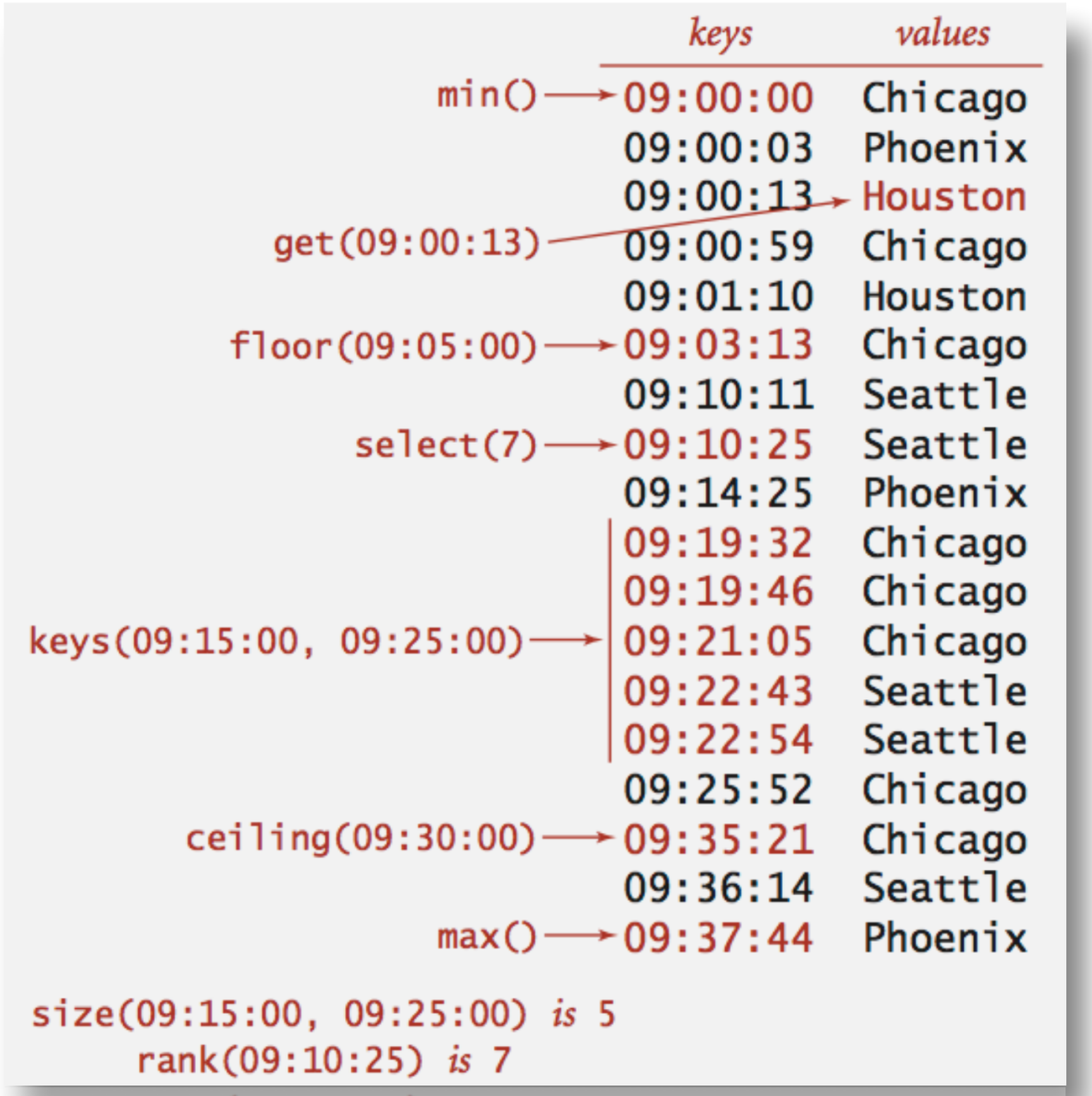
Lecture Overview



3.1 Symbol Tables

- API
- Elementary Implementations
- Ordered Operations

Examples of Ordered Symbol Table API



Ordered Symbol Table API

```
public class ST<Key extends Comparable<Key>, Value>
```

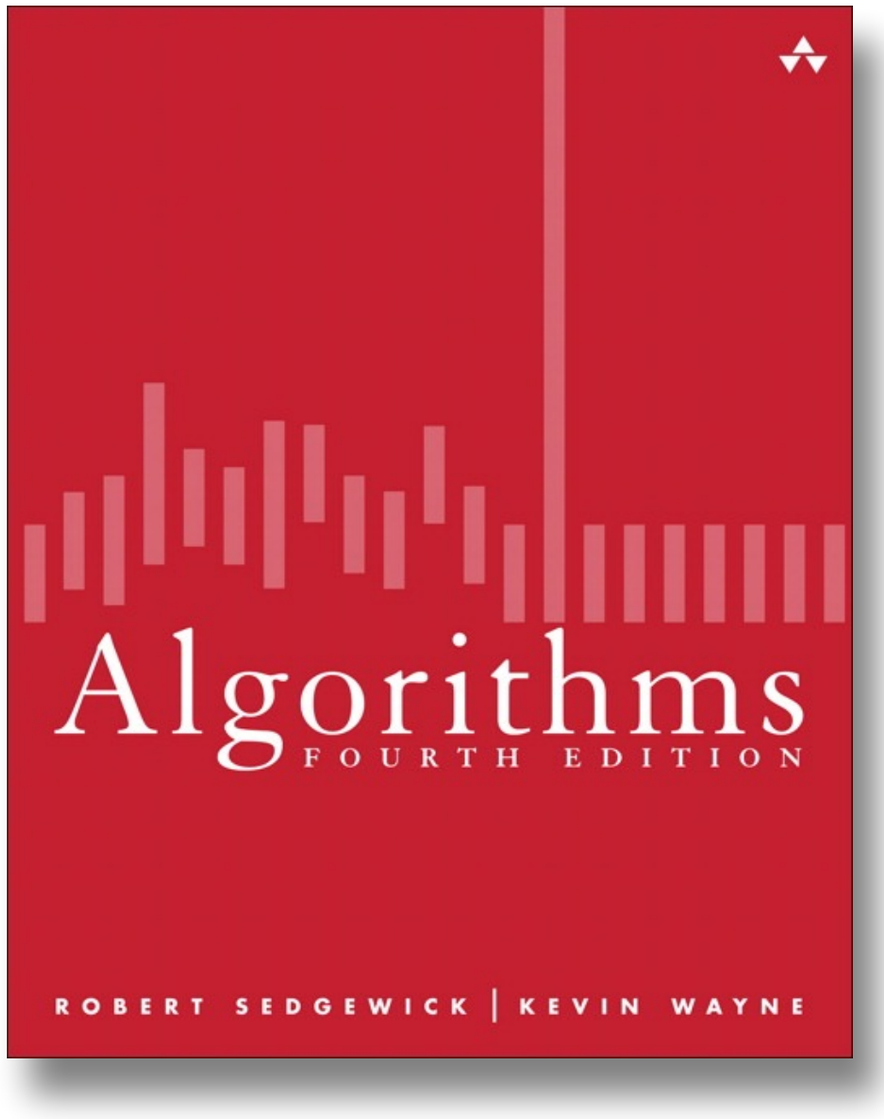
	...	
Key	min()	*smallest key*
Key	max()	*largest key*
Key	floor(Key key)	*largest key less than or equal to key*
Key	ceiling(Key key)	*smallest key greater than or equal to key*
int	rank(Key key)	*number of keys less than key*
Key	select(int k)	*key of rank k*
void	deleteMin()	*delete smallest key*
void	deleteMax()	*delete largest key*
int	size(Key lo, Key hi)	*number of keys between lo and hi*
Iterable<Key>	keys()	*all keys, in sorted order*
Iterable<Key>	keys(Key lo, Key hi)	*keys between lo and hi, in sorted order*

Binary Search: Ordered Symbol Table Operations Summary

	sequential search	binary search
search	N	$\log N$
insert / delete	N	N
min / max	N	1
floor / ceiling	N	$\log N$
rank	N	$\log N$
select	N	1
ordered iteration	$N \log N$	N

order of growth of the running time for ordered symbol table operations

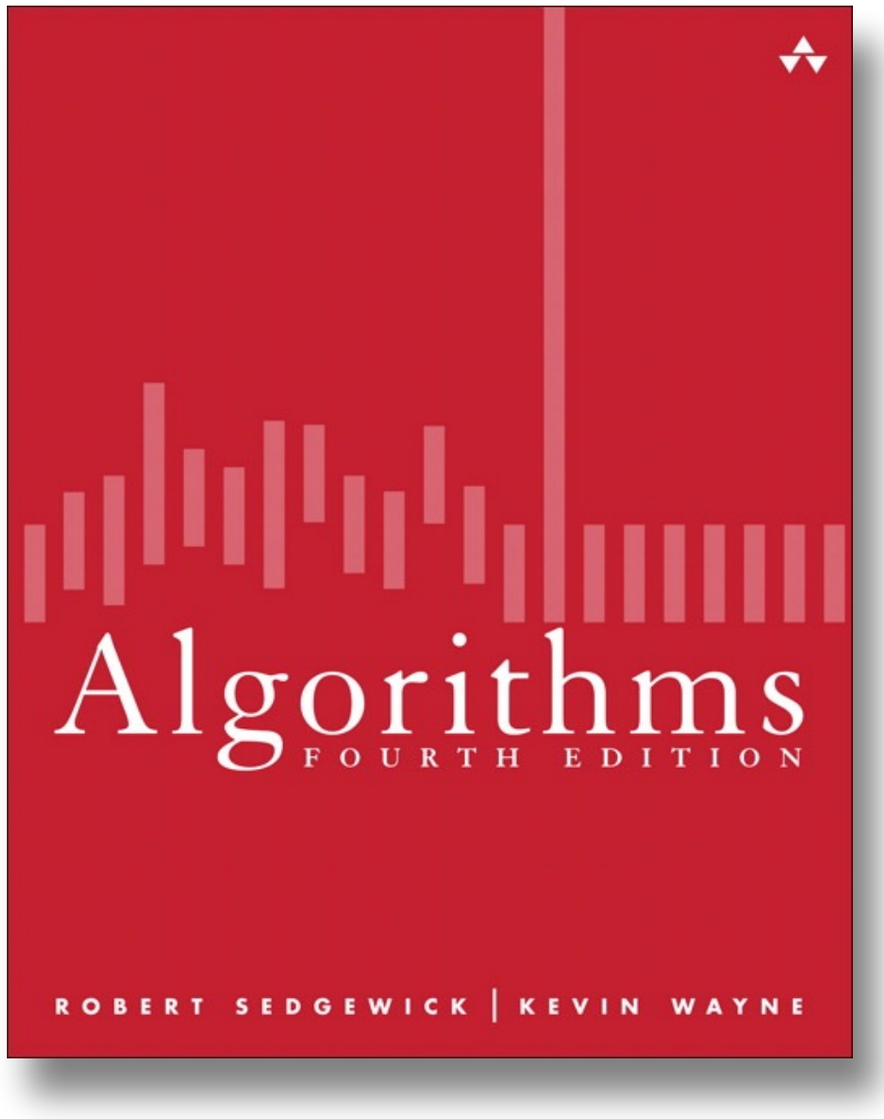
Lecture Overview



3.5 Symbol Table Applications

- Sets
- Dictionary Clients
- Indexing Clients
- Sparse Vectors

Lecture Overview



3.5 Symbol Table Applications

- Sets
- Dictionary Clients
- Indexing Clients
- Sparse Vectors

Set API

Mathematical set: A collection of distinct keys.

public class SET<Key extends Comparable<Key>>	
SET()	*create an empty set*
void add(Key key)	*add the key to the set*
boolean contains(Key key)	*is the key in the set?*
void remove(Key key)	*remove the key from the set*
int size()	*return the number of keys in the set*
Iterator<Key> iterator()	*iterator through keys in the set*

Question: How to implement ?

Exception Filter

Read in a list of words from one file.
Print out all words from standard input that are { in, not in } the list.

```
% more list.txt
was it the of

% java WhiteList list.txt < tinyTale.txt
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of

% java BlackList list.txt < tinyTale.txt
best times worst times
age wisdom age foolishness
epoch belief epoch incredulity
season light season darkness
spring hope winter despair
```

list of exceptional words

Exception Filter Applications

Read in a list of words from one file.
Print out all words from standard input that are { in, not in } the list.

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards

Exception Filter: Java Implementation

Read in a list of words from one file.
Print out all words from standard input that are in the list.

```
public class WhiteList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();

      In in = new In(args[0]);
      while (!in.isEmpty())
         set.add(in.readString());

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (set.contains(word))
            StdOut.println(word);
      }
   }
}
```

create empty set of strings

read in whitelist

print words in list

Exception Filter: Java Implementation

Read in a list of words from one file.
Print out all words from standard input that are **not** in the list.

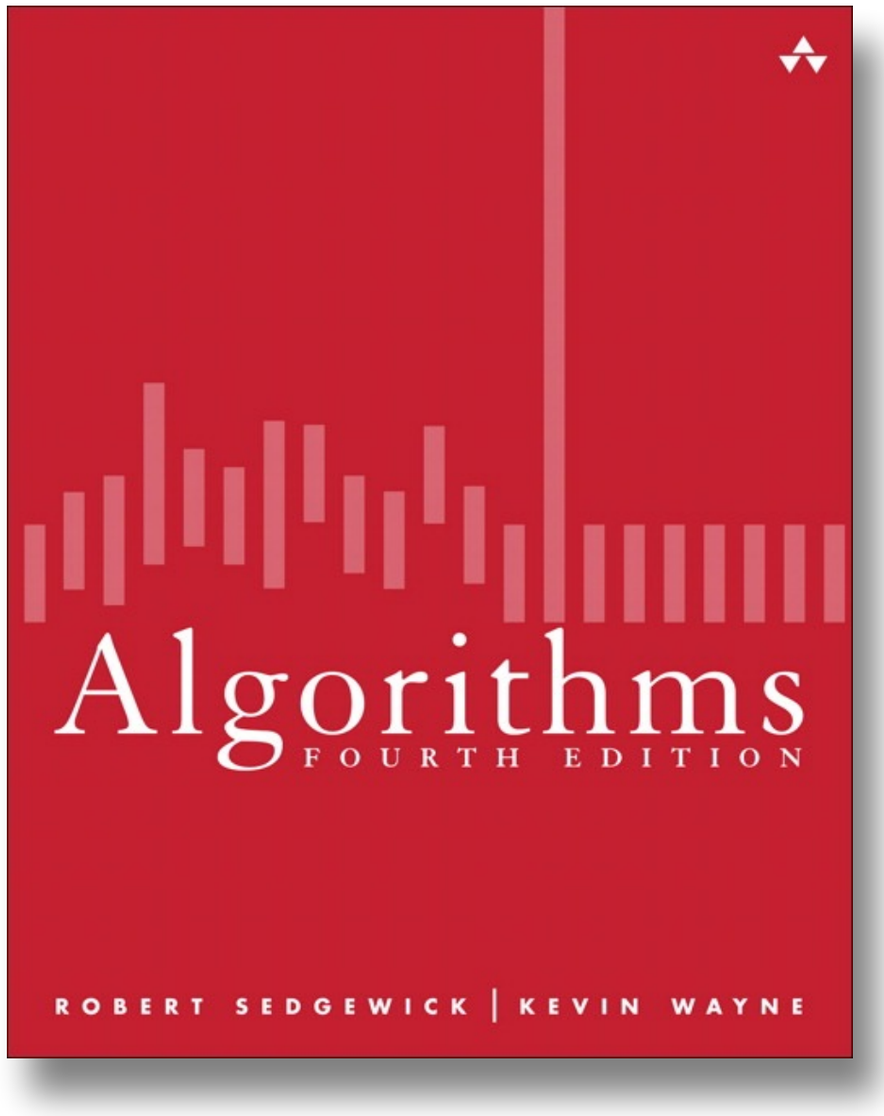
```
public class BlackList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();

      In in = new In(args[0]);
      while (!in.isEmpty())
         set.add(in.readString());

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (!set.contains(word))
            StdOut.println(word);
      }
   }
}
```

create empty set of strings

read in whitelist

print words not in list

Lecture Overview



3.5 Symbol Table Applications

- Sets
- Dictionary Clients
- Indexing Clients
- Sparse Vectors

Dictionary Lookup

Command-line arguments

- A comma-separated value (CSV) file
- Key field
- Value field

Ex. 1 - DNS lookup

domain name is key IP is value

```
% java LookupCSV ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found
```

domain name is key URL is value

```
% java LookupCSV ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
```

```
% more ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
www.yale.edu,130.132.51.8
www.econ.yale.edu,128.36.236.74
www.cs.yale.edu,128.36.229.30
espn.com,199.181.135.201
yahoo.com,66.94.234.13
msn.com,207.68.172.246
google.com,64.233.167.99
baidu.com,202.108.22.33
yahoo.co.jp,202.93.91.141
sina.com.cn,202.108.33.32
ebay.com,66.135.192.87
adobe.com,192.150.18.60
163.com,220.181.29.154
passport.net,65.54.179.226
tom.com,61.135.158.237
nate.com,203.226.253.11
cnn.com,64.236.16.20
daum.net,211.115.77.211
blogger.com,66.102.15.100
fastclick.com,205.180.86.4
wikipedia.org,66.230.200.100
rakuten.co.jp,202.72.51.22
...
```

Dictionary Lookup

Command-line arguments

- A comma-separated value (CSV) file
- Key field
- Value field

Ex. 2 - Amino acids

codon is key name is value

```
% java LookupCSV amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

```
% more amino.csv
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
...
```

Dictionary Lookup

Command-line arguments

- A comma-separated value (CSV) file
- Key field
- Value field

Ex. 3 - Class list

```
% java LookupCSV classlist.csv 4 1
eberl
Ethan
nwebb
Natalie

% java LookupCSV classlist.csv 4 3
dpan
P01
```

first name is value (login is key)

section is value (login is key)

```
% more classlist.csv
13,Berl,Ethan Michael,P01,eberl
12,Cao,Phillips Minghua,P01,pcao
11,Chehoud,Christel,P01,cchehoud
10,Douglas,Malia Morioka,P01,malia
12,Haddock,Sara Lynn,P01,shaddock
12,Hantman,Nicole Samantha,P01,nhantman
11,Hesterberg,Adam Classen,P01,ahesterb
13,Hwang,Roland Lee,P01,rhwang
13,Hyde,Gregory Thomas,P01,ghyde
13,Kim,Hyunmoon,P01,hktwo
12,Korac,Damjan,P01,dkorac
11,MacDonald,Graham David,P01,gmacdona
10,Michal,Brian Thomas,P01,bmichal
12,Nam,Seung Hyeon,P01,seungnam
11,Nastasescu,Maria Monica,P01,mnastase
11,Pan,Di,P01,dpan
12,Partridge,Brenton Alan,P01,bpartrid
13,Rilee,Alexander,P01,arilee
13,Roopakalu,Ajay,P01,aroopaka
11,Sheng,Ben C,P01,bsheng
12,Webb,Natalie Sue,P01,nwebb
⋮
```

Dictionary Lookup: Java Implementation

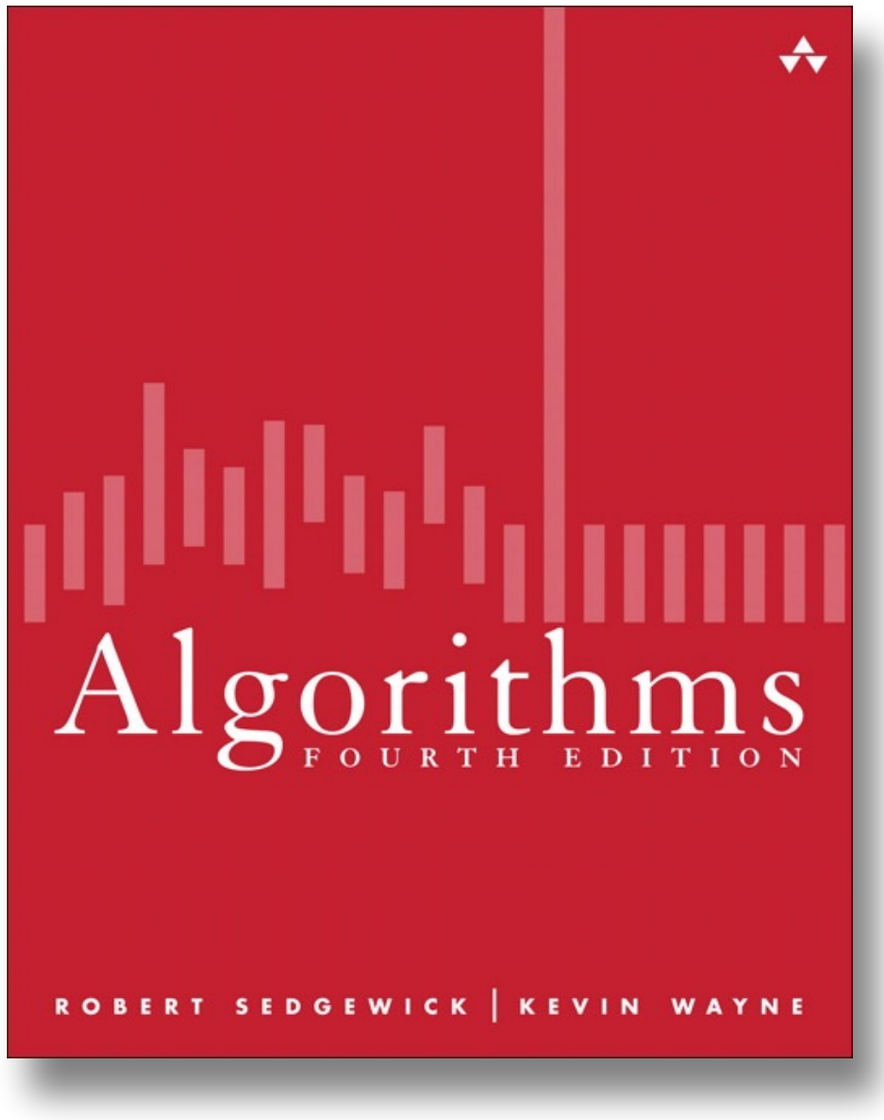
```
public class LookupCSV
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      int keyField = Integer.parseInt(args[1]);
      int valField = Integer.parseInt(args[2]);

      ST<String, String> st = new ST<String, String>();
      while (!in.isEmpty())
      {
         String line = in.readLine();
         String[] tokens = line.split(",");
         String key = tokens[keyField];
         String val = tokens[valField];
         st.put(key, val);
      }

      while (!StdIn.isEmpty())
      {
         String s = StdIn.readString();
         if (!st.contains(s)) StdOut.println("Not found");
         else                 StdOut.println(st.get(s));
      }
   }
}
```

process input file

build symbol table

process lookups with standard I/O

Lecture Overview

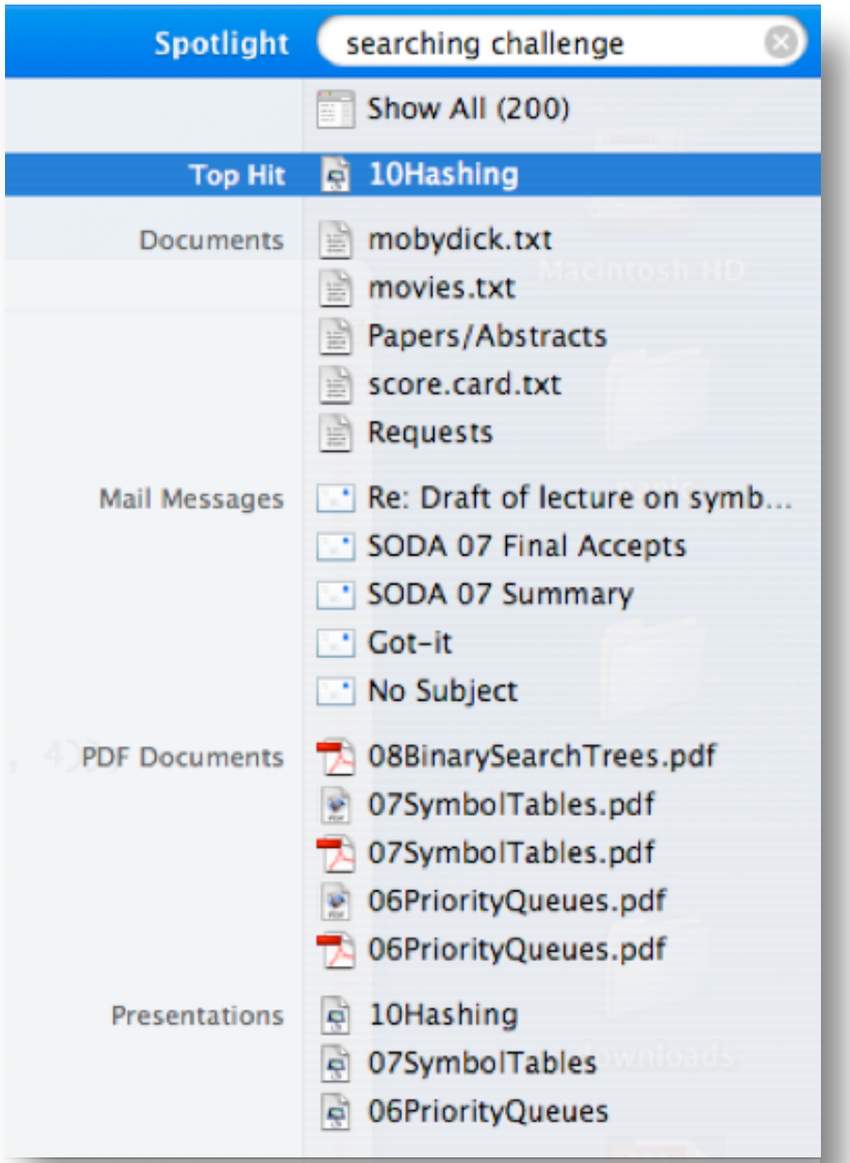


3.5 Symbol Table Applications

- Sets
- Dictionary Clients
- Indexing Clients
- Sparse Vectors

File Indexing

Goal: Index a PC (or the web).



File Indexing

Goal: Given a list of files, create an index so that you can efficiently find all files containing a given query string.

```
% ls *.txt
aesop.txt magna.txt moby.txt
sawyer.txt tale.txt

% java FileIndex *.txt

freedom
magna.txt moby.txt tale.txt

whale
moby.txt

lamb
sawyer.txt aesop.txt
```

```
% ls *.java
BlackList.java Concordance.java
DeDup.java FileIndex.java ST.java
SET.java WhiteList.java

% java FileIndex *.java

import
FileIndex.java SET.java ST.java

Comparator
null
```

Solution: Key = query string; value = set of files containing that string..

File Indexing

```
import java.io.File;
public class FileIndex
{
   public static void main(String[] args)
   {
      ST<String, SET<File>> st = new ST<String, SET<File>>();

      for (String filename : args) {
         File file = new File(filename);
         In in = new In(file);
         while (!in.isEmpty())
         {
            String key = in.readString();
            if (!st.contains(key))
               st.put(word, new SET<File>());
            SET<File> set = st.get(key);
            set.add(file);
         }
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         StdOut.println(st.get(query));
      }
   }
}
```

symbol table

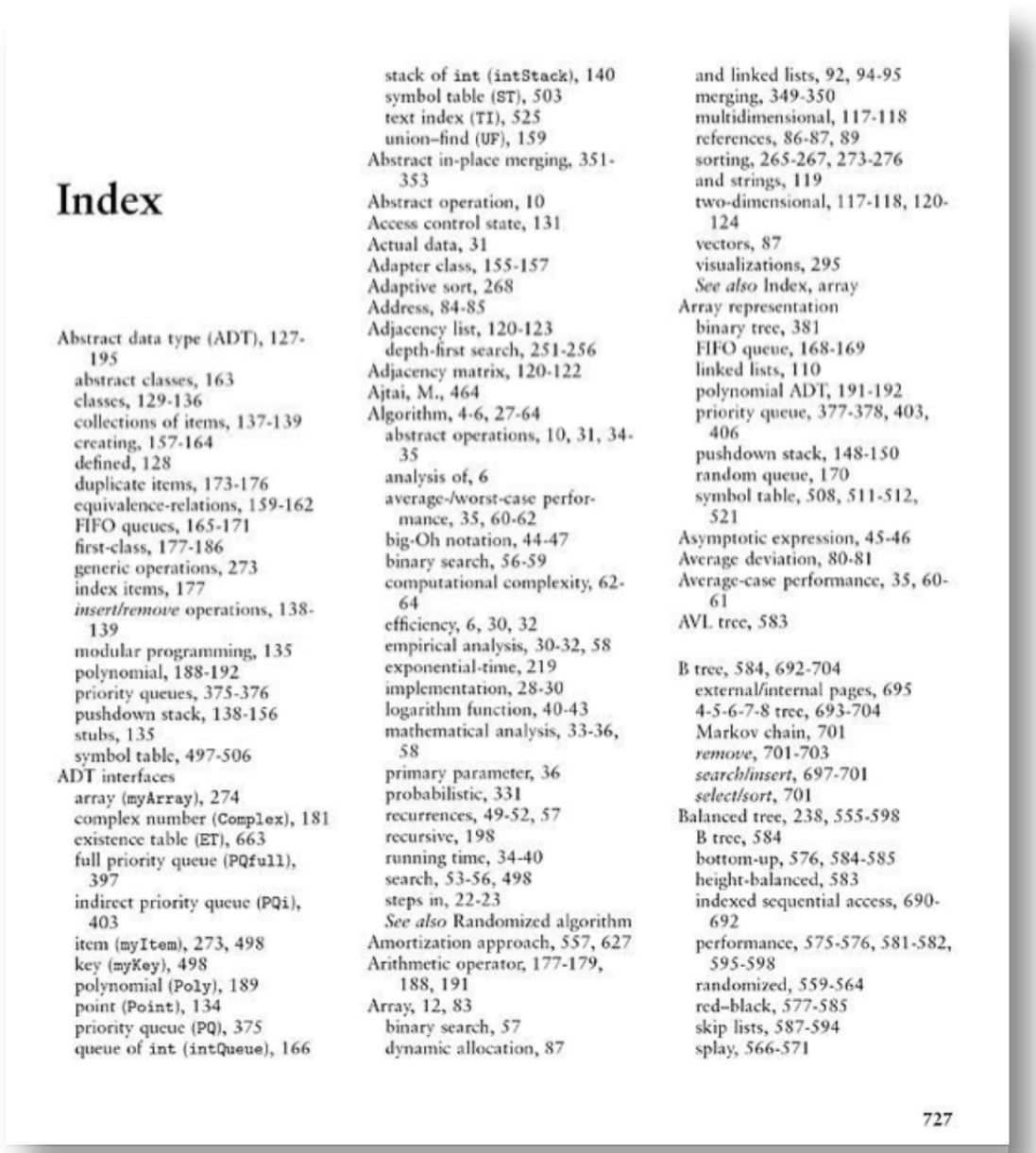
list of file names from command line

for each word in file, add file to corresponding set

process queries

File Indexing

Goal: Index a e-book.



Index

Concordance

Goal: Preprocess a text corpus to support concordance queries: given a word, find all occurrences with their immediate contexts.

```
% java Concordance tale.txt
cities
tongues of the two *cities* that were blended in

majesty
their turnkeys and the *majesty* of the law fired
me treason against the *majesty* of the people in
  of his most gracious *majesty* king george the third

princeton
no matches
```

Solution: Key = query string; value = set of indices containing that string.

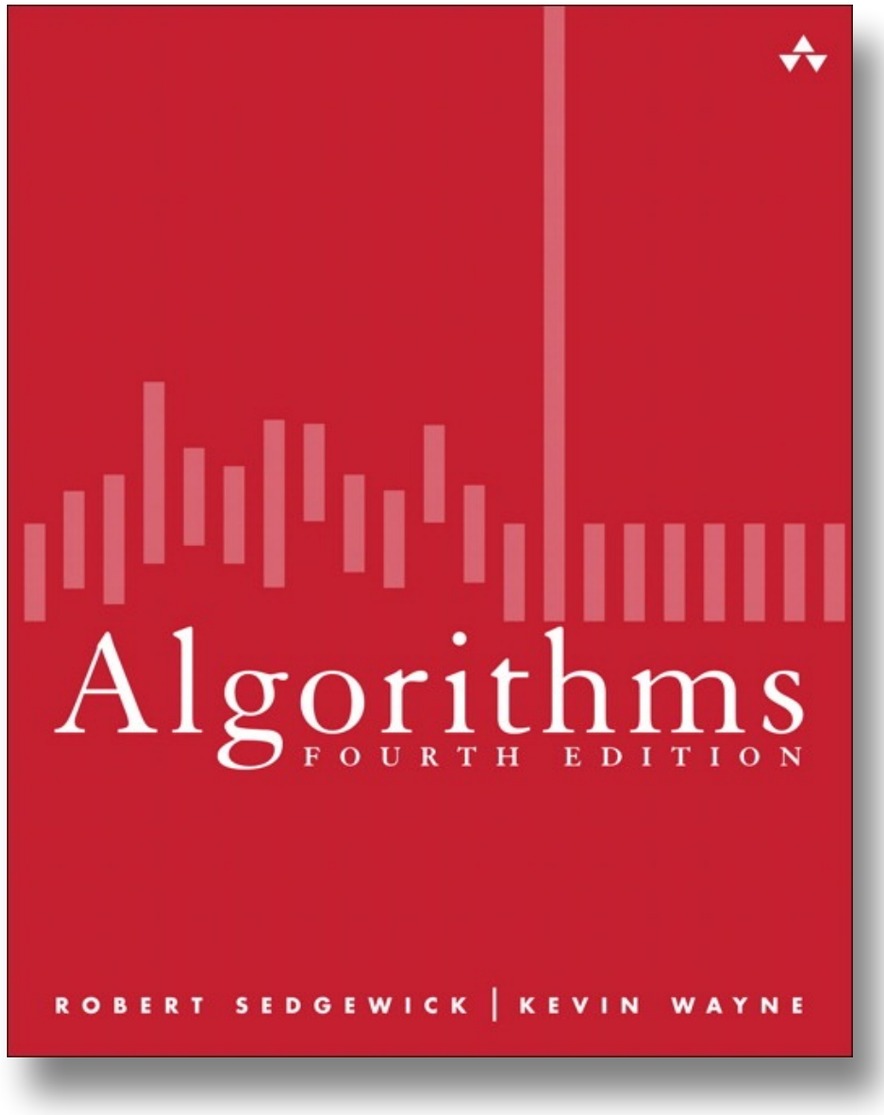
Concordance

```
public class Concordance
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      String[] words = in.readAllStrings();
      ST<String, SET<Integer>> st = new ST<String, SET<Integer>>();
      for (int i = 0; i < words.length; i++)
      {
         String s = words[i];
         if (!st.contains(s))
            st.put(s, new SET<Integer>());
         SET<Integer> set = st.get(s);
         set.add(i);
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         SET<Integer> set = st.get(query);
         for (int k : set)
             // print words[k-4] to words[k+4]
      }
   }
}
```

read text and build index

process queries and print concordances

Lecture Overview



3.5 Symbol Table Applications

- Sets
- Dictionary Clients
- Indexing Clients
- Sparse Vectors

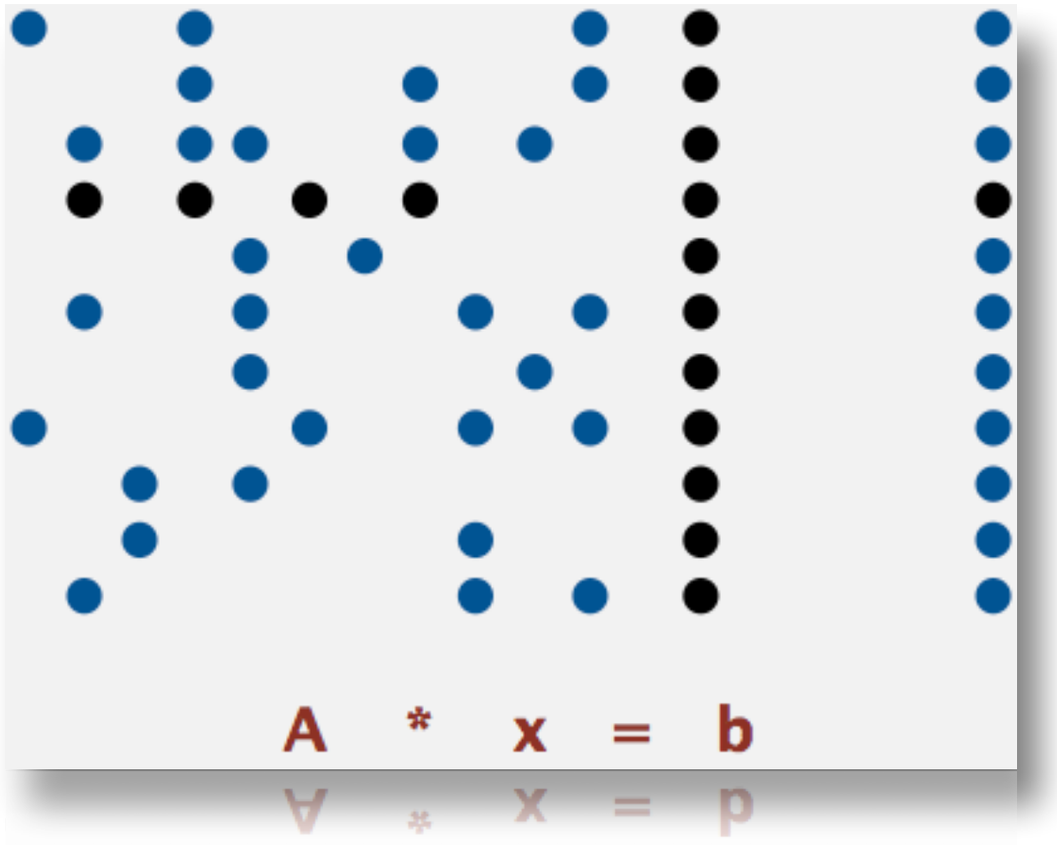
Matrix-Vector Multiplication (Standard Implementation)

```
...
double[][] a = new double[N][N];
double[] x = new double[N];
double[] b = new double[N];
...
// initialize a[][] and x[]
...
for (int i = 0; i < N; i++)
{
   sum = 0.0;
   for (int j = 0; j < N; j++)
      sum += a[i][j]*x[j];
   b[i] = sum;
}
```

nested loops
(N^2 running time)

Sparse Matrix-Vector Multiplication

Problem: Sparse matrix-vector multiplication.

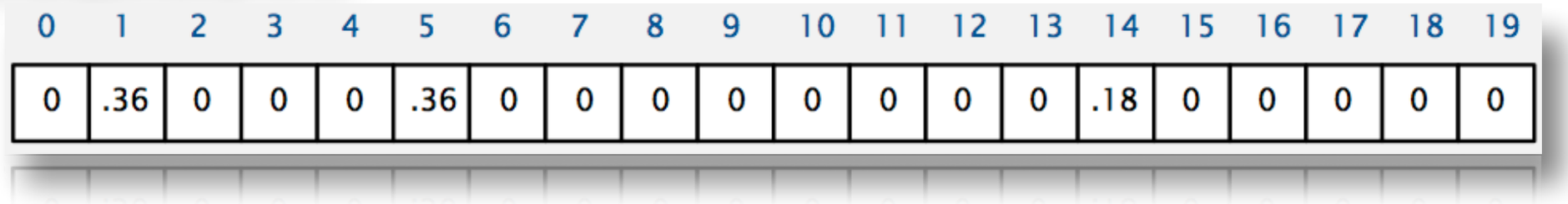


Assumptions: Matrix dimension is 10,000; average nonzeros per row ~ 10.

Vector Representations

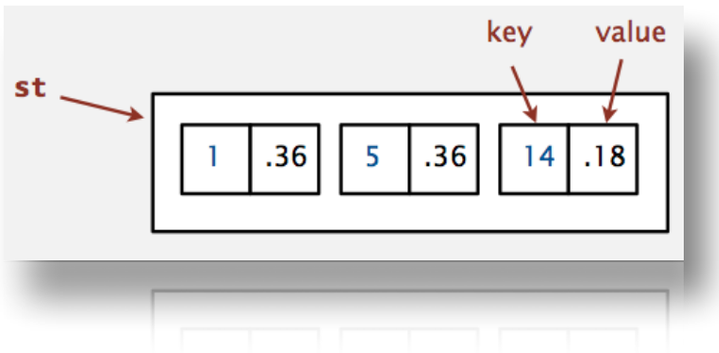
1d array (standard) representation.

- Constant time access to elements.
- Space proportional to N.



Symbol table representation.

- Key = index, value = entry.
- Efficient iterator.
- Space proportional to number of nonzeros.



Sparse Vector Data Type

```
public class SparseVector
{
   private HashST<Integer, Double> v;

   public SparseVector()
   {  v = new HashST<Integer, Double>();  }

   public void put(int i, double x)
   {  v.put(i, x);  }

   public double get(int i)
   {
      if (!v.contains(i)) return 0.0;
      else return v.get(i);
   }

   public Iterable<Integer> indices()
   {  return v.keys();  }

   public double dot(double[] that)
   {
       double sum = 0.0;
       for (int i : indices())
           sum += that[i]*this.get(i);
       return sum;
   }
}
```

HashST because order not important

empty ST represents all 0s vector

a[i] = value

return a[i]

iterate through indices of nonzero entries

dot product is constant time for sparse vectors

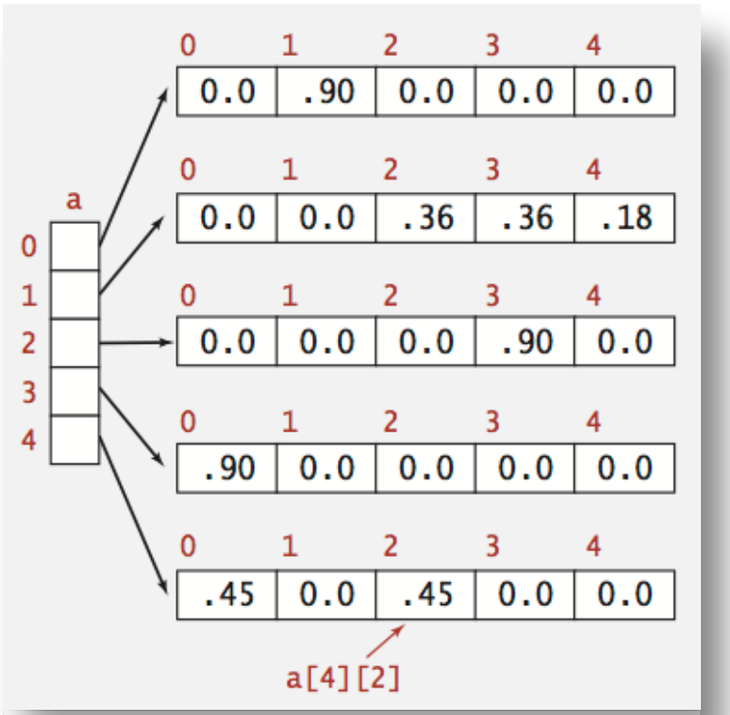
Matrix Representations

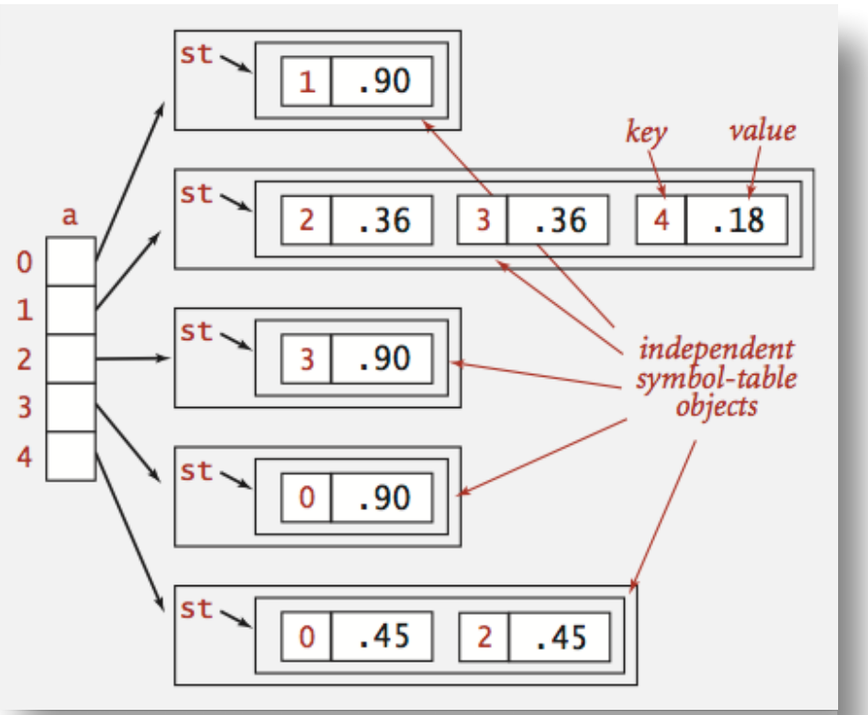
2D array (standard) matrix representation: Each row of matrix is an array.

- Constant time access to elements.
- Space proportional to N2.

Sparse matrix representation: Each row of matrix is a sparse vector.

- Efficient access to elements.
- Space proportional to number of nonzeros (plus N).



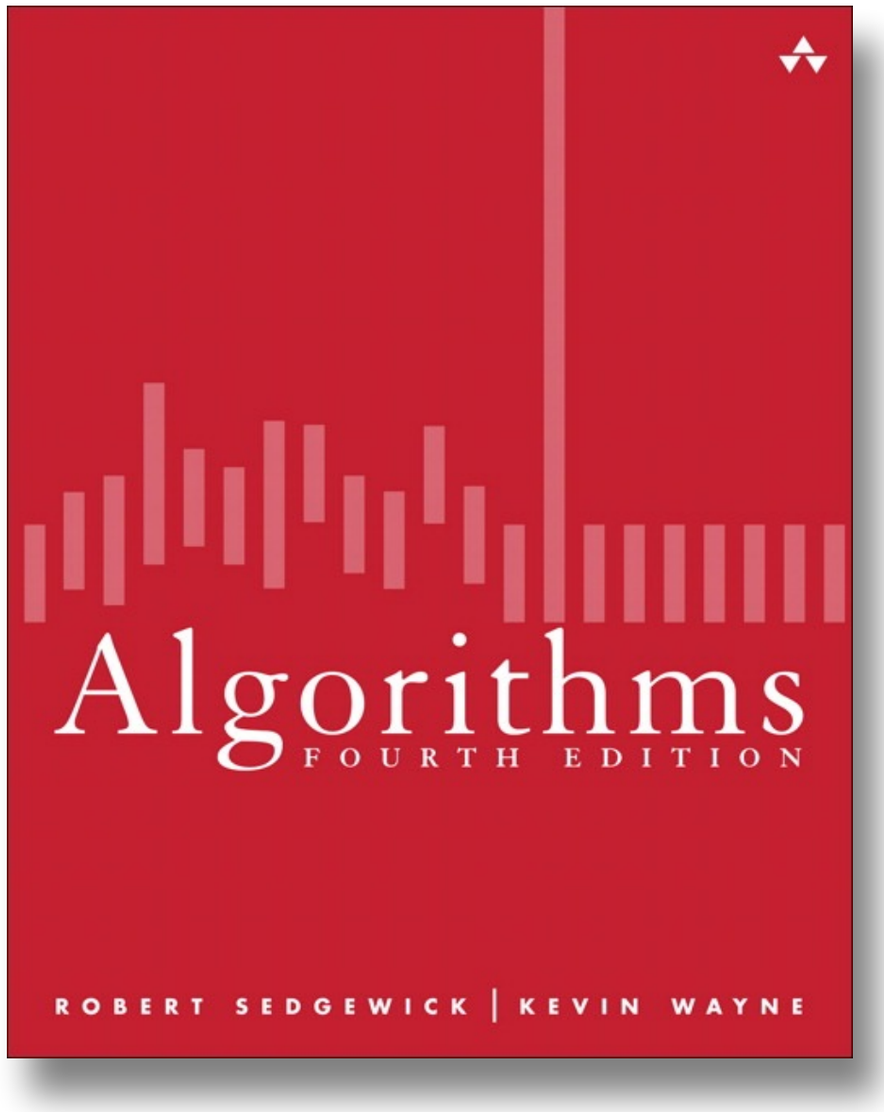


Sparse Matrix-Vector Multiplication

```
..
SparseVector[] a = new SparseVector[N];
double[] x = new double[N];
double[] b = new double[N];
...
// Initialize a[] and x[]
...
for (int i = 0; i < N; i++)
   b[i] = a[i].dot(x);
```

linear running time
for sparse matrix

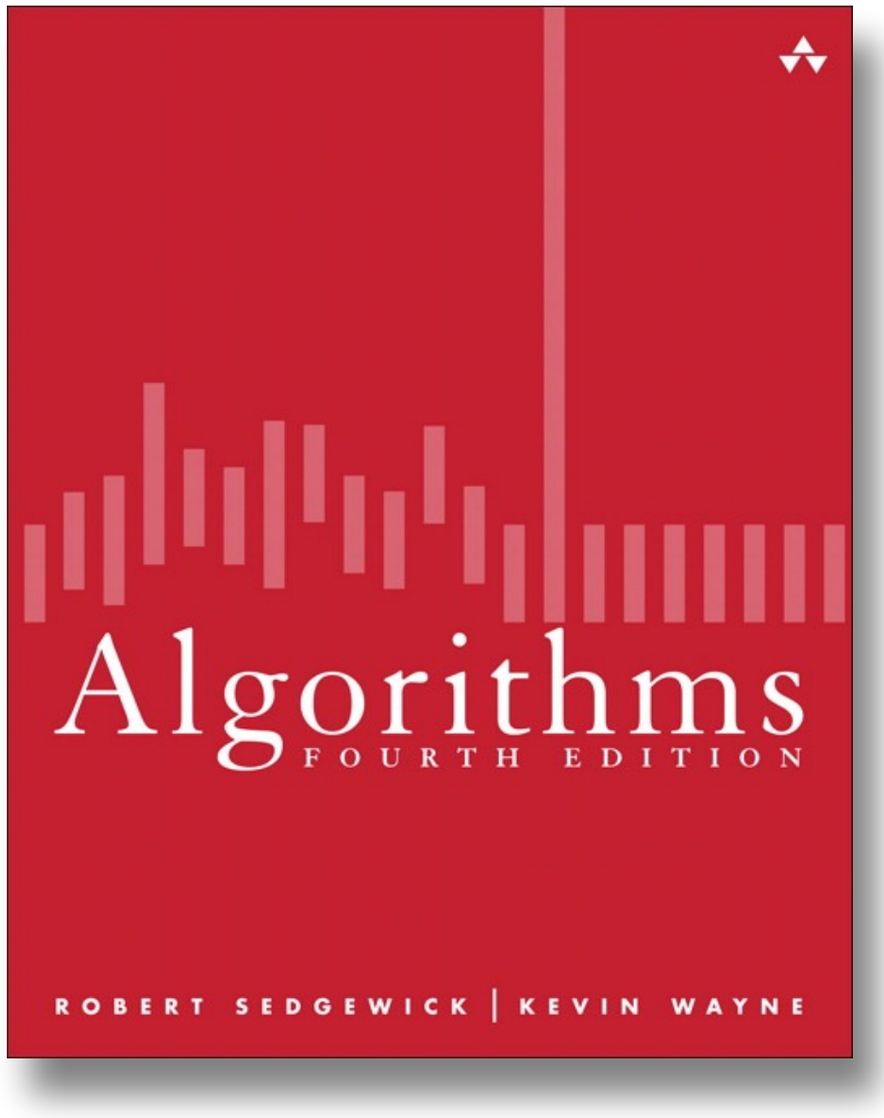
Lecture Overview



3.2 Binary Search Tree

- BSTs
- Ordered operations
- Deletion

Lecture Overview



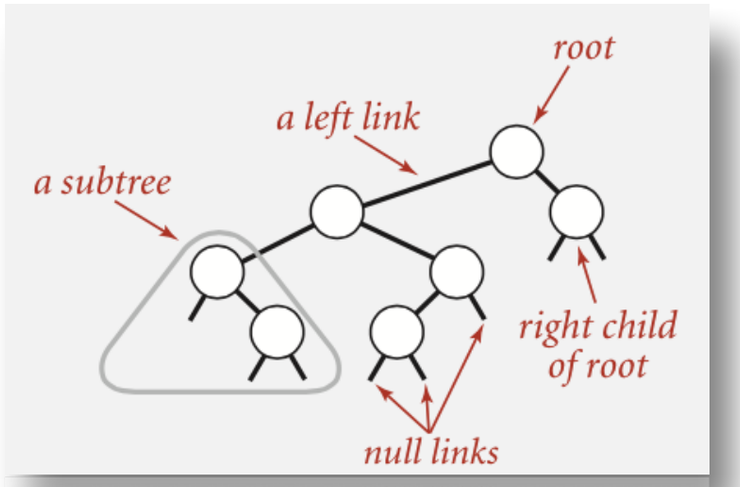
3.2 Binary Search Tree

- BSTs
- Ordered operations
- Deletion

Binary Search Trees

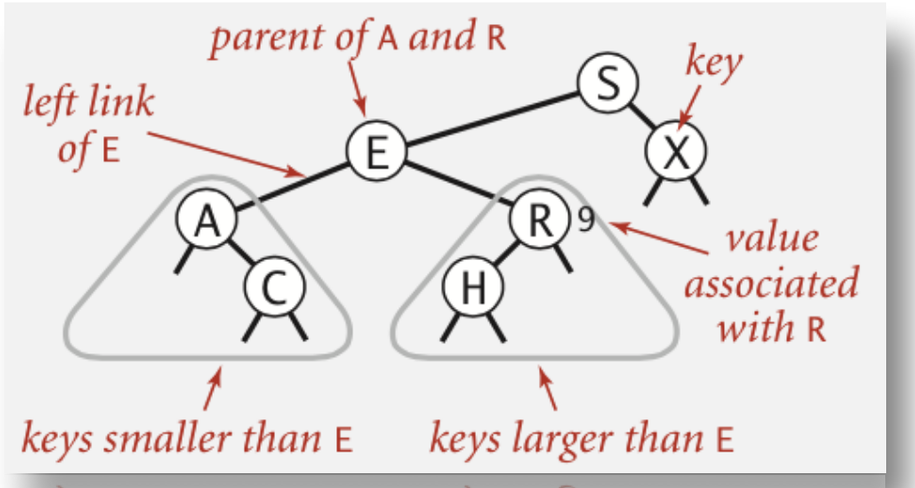
Definition: A BST is a binary tree in symmetric order.

- Empty.
- Two disjoint binary trees (left and right).



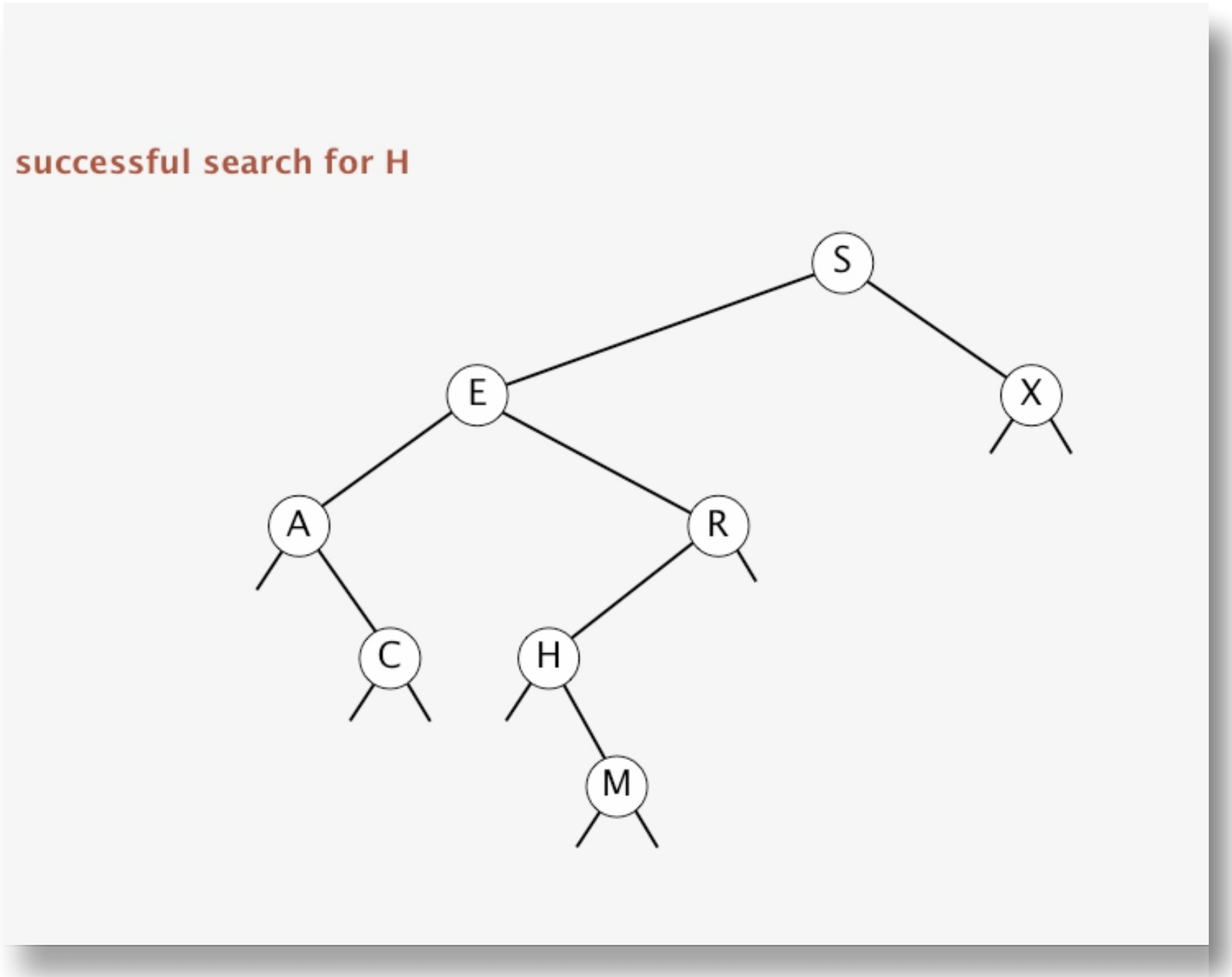
Symmetric order. Each node has a key, and every node's key is.

- Larger than all keys in its left subtree.
- Smaller than all keys in its right subtree.



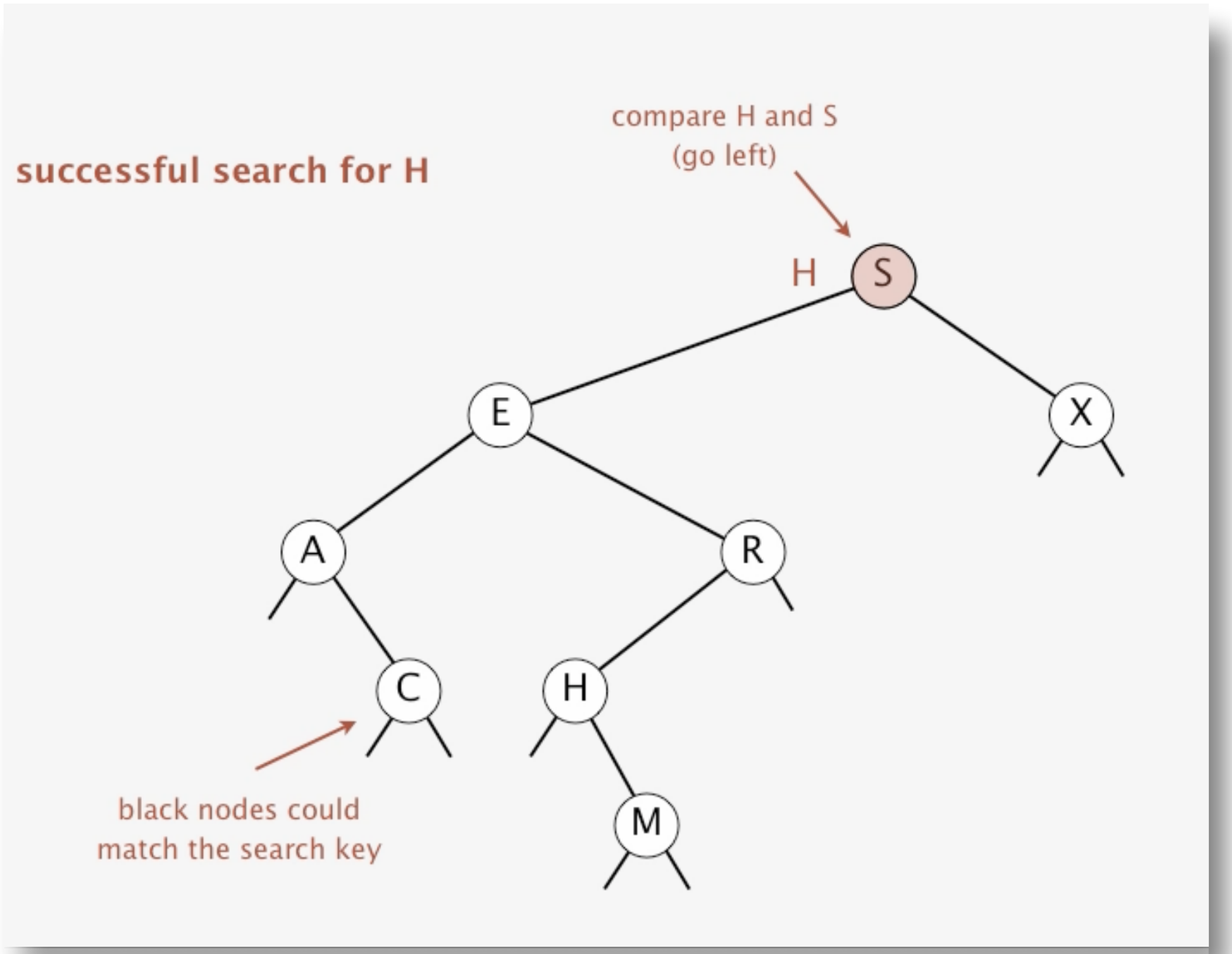
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



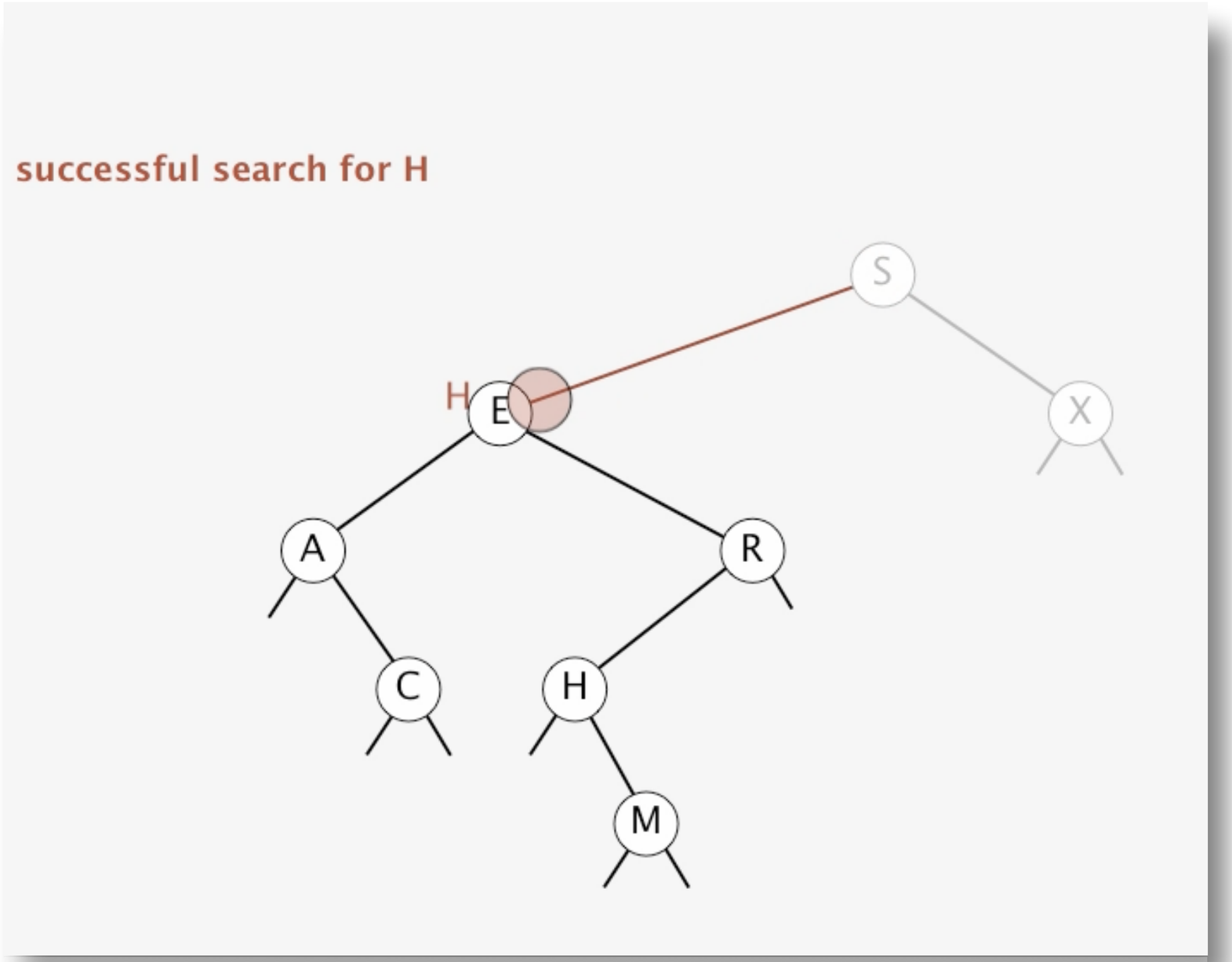
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



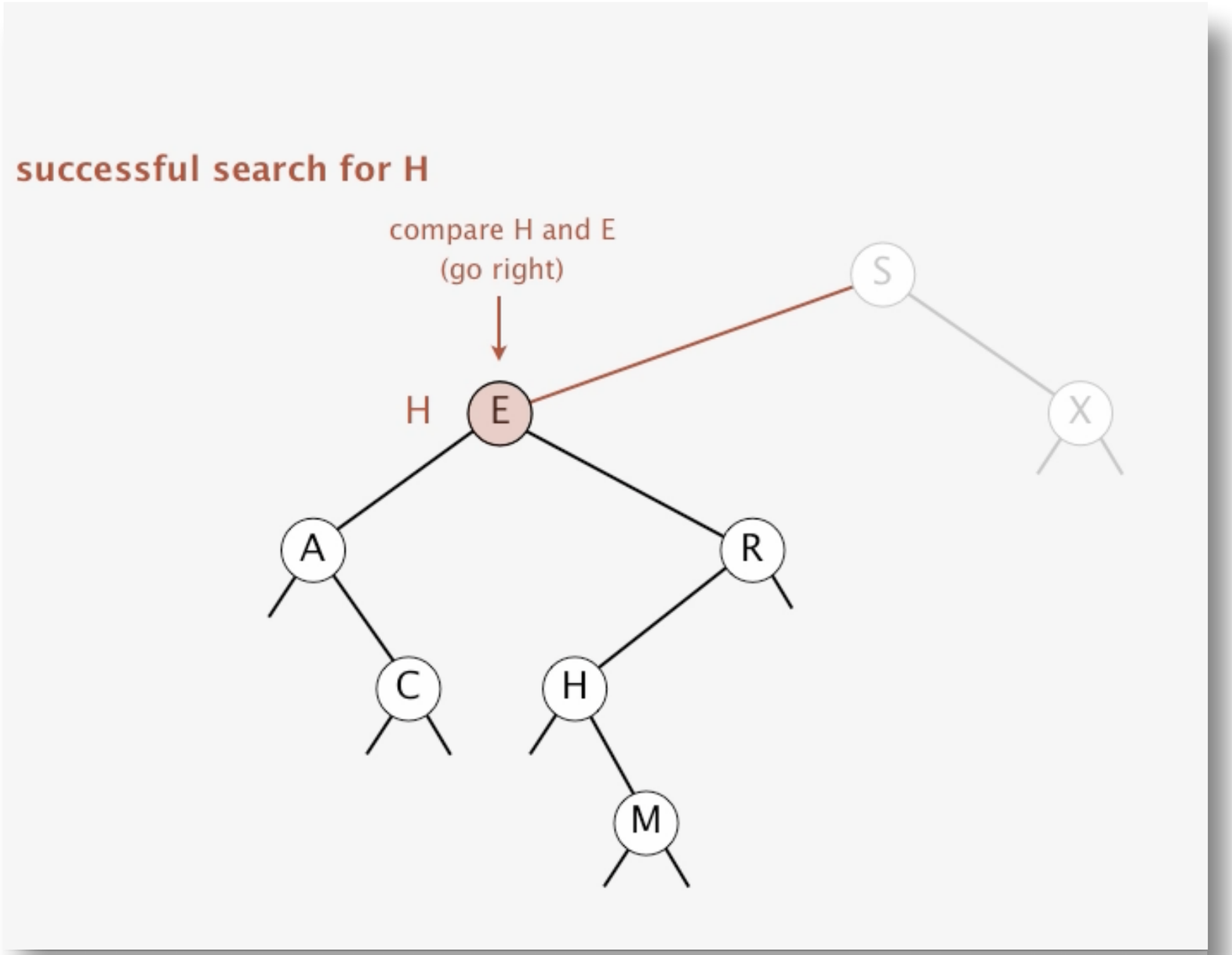
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



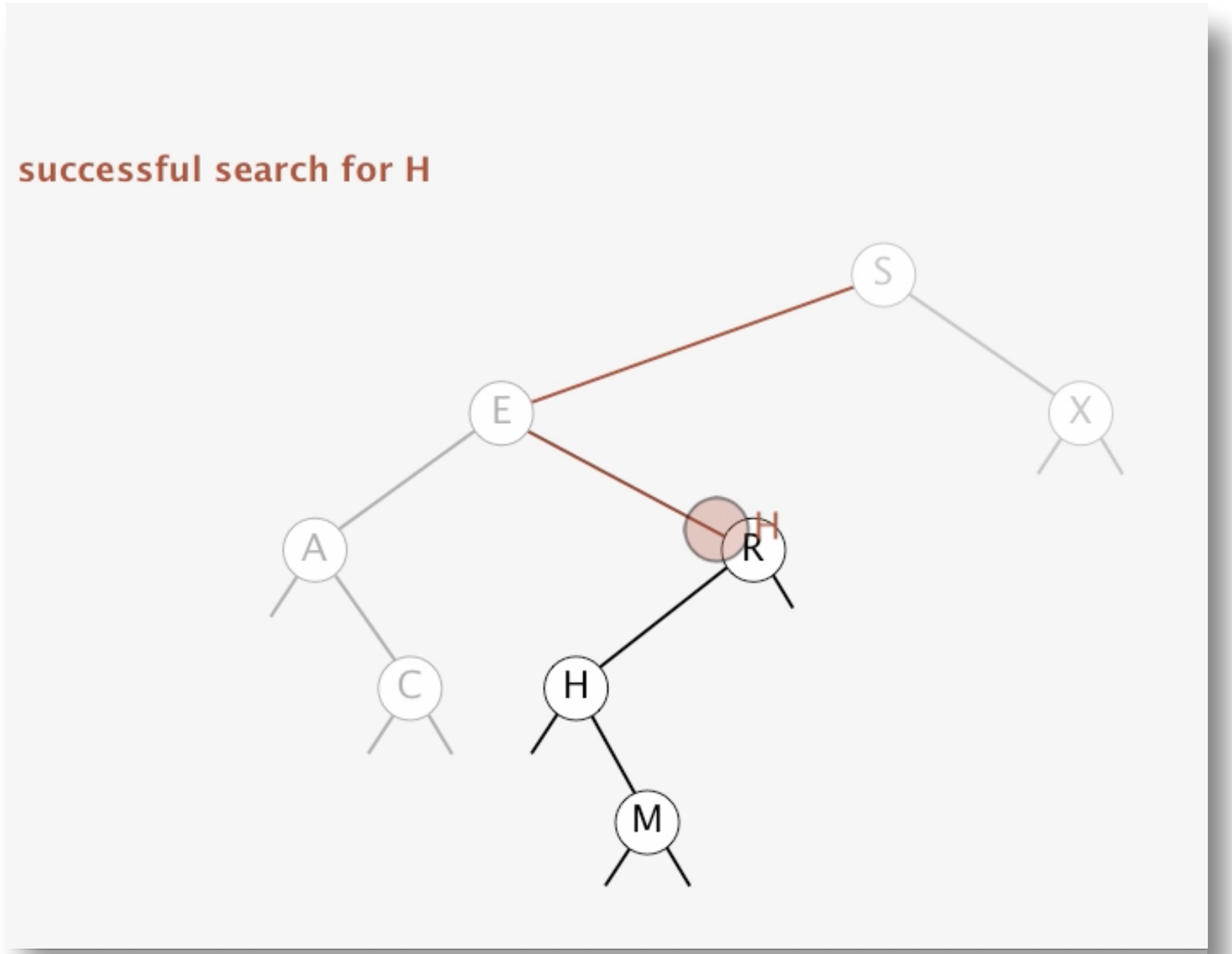
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



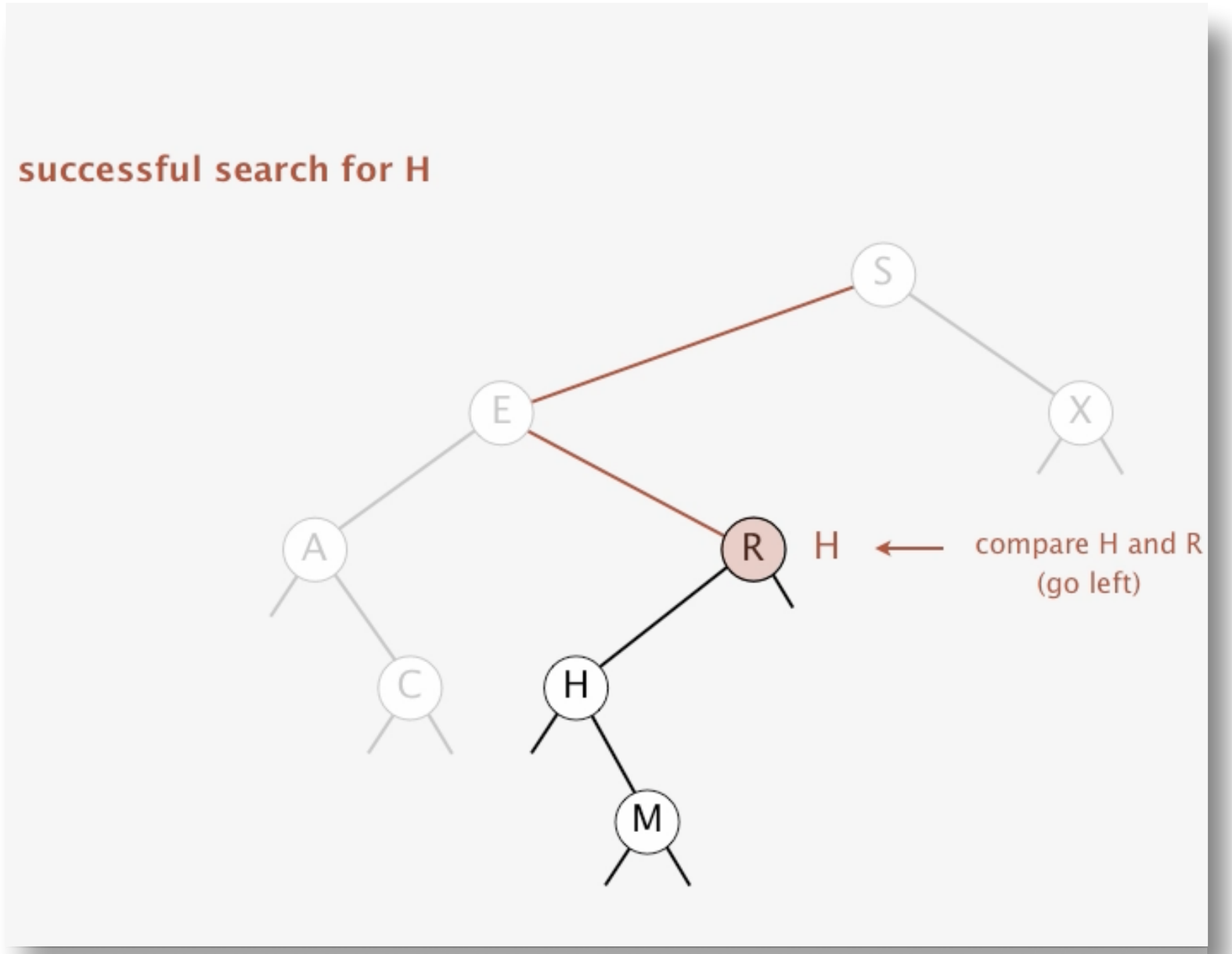
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



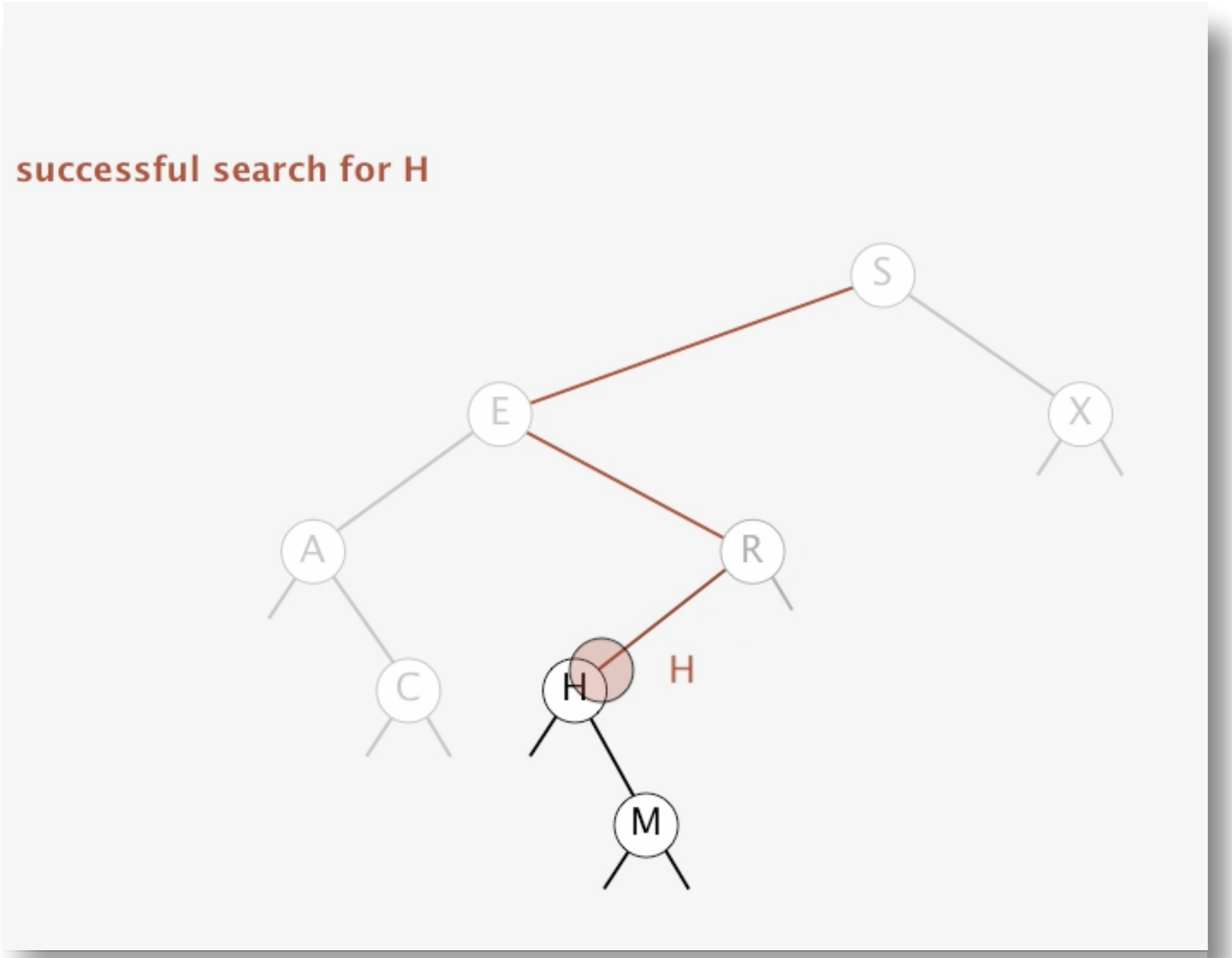
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



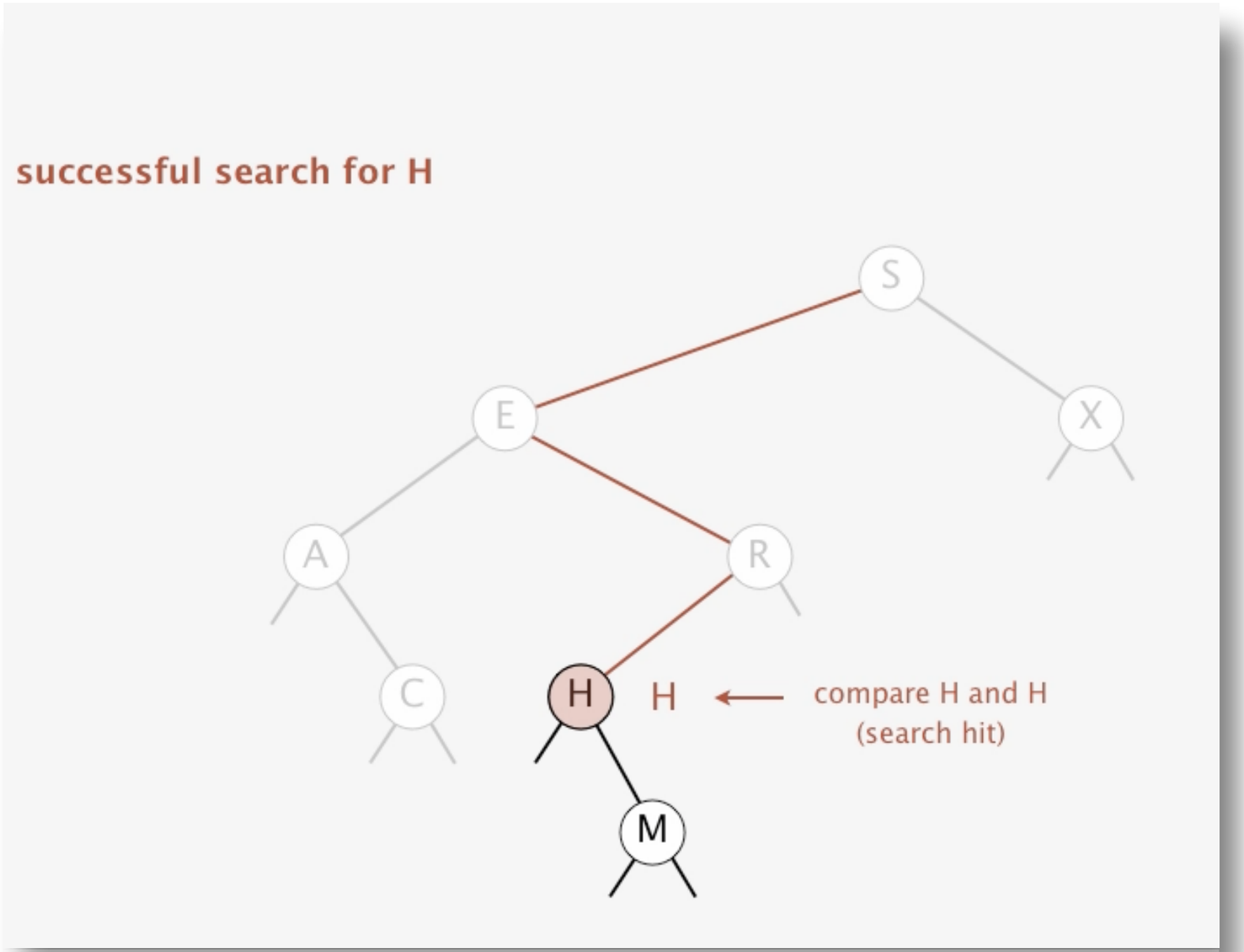
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



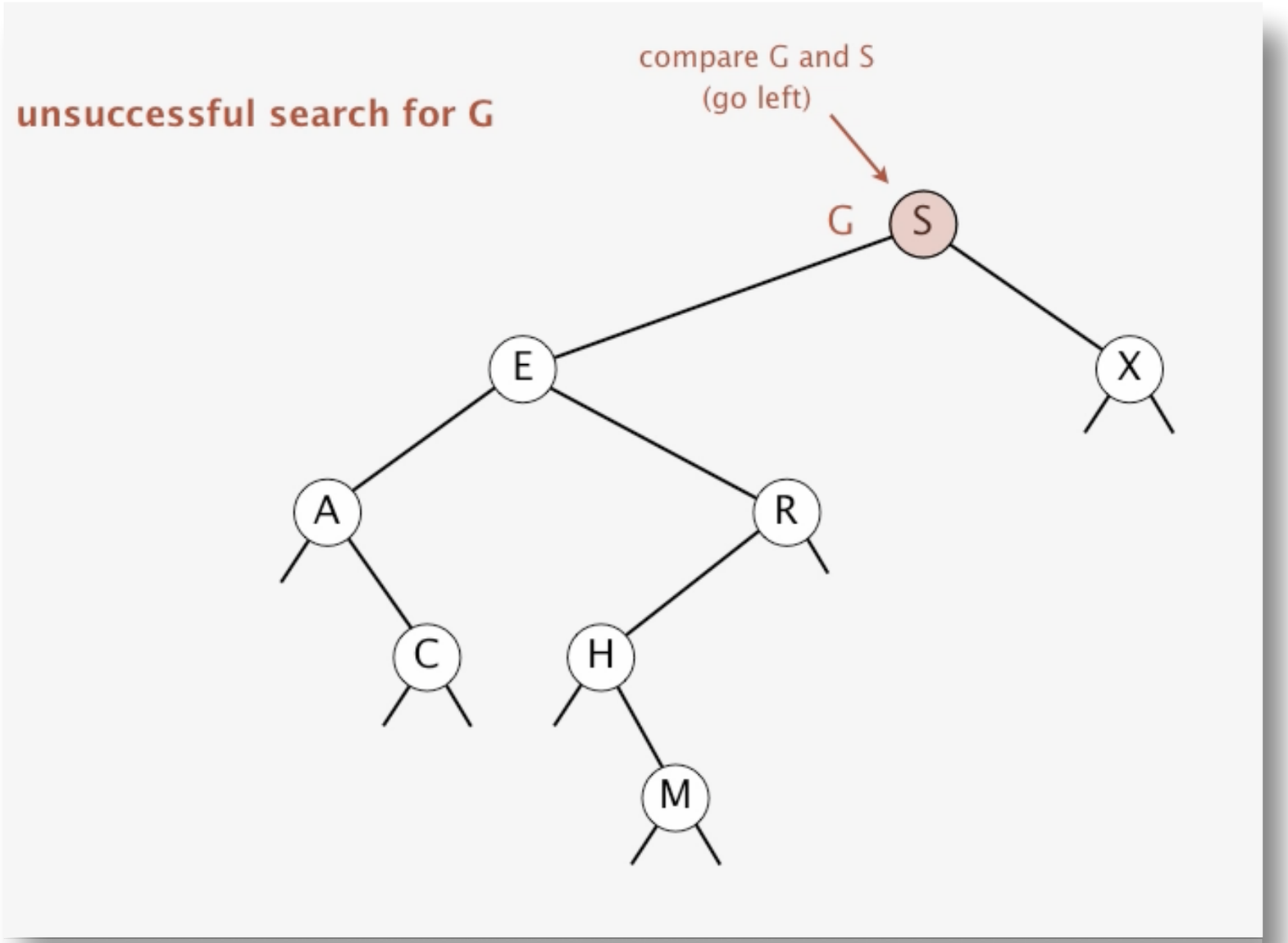
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



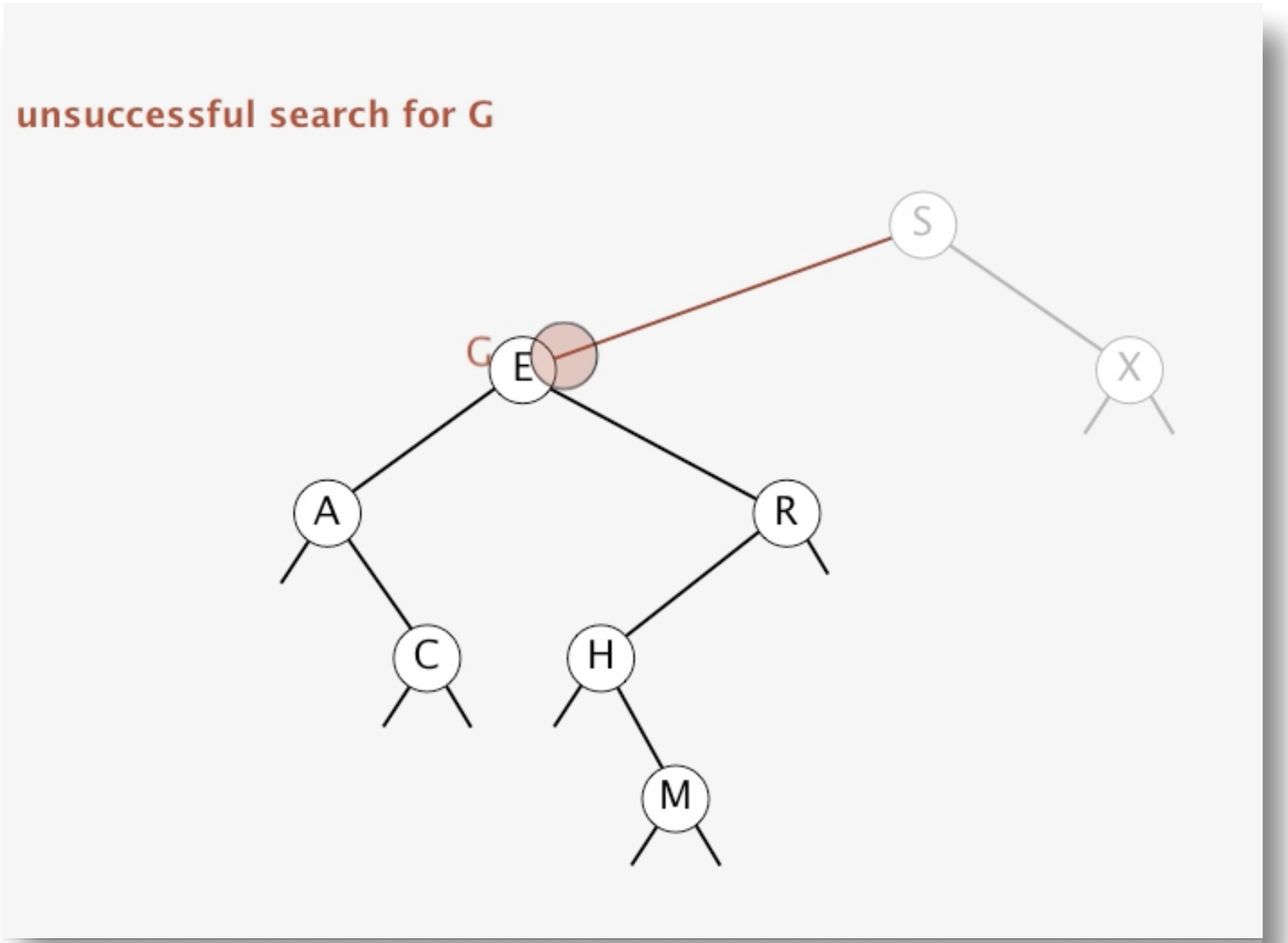
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



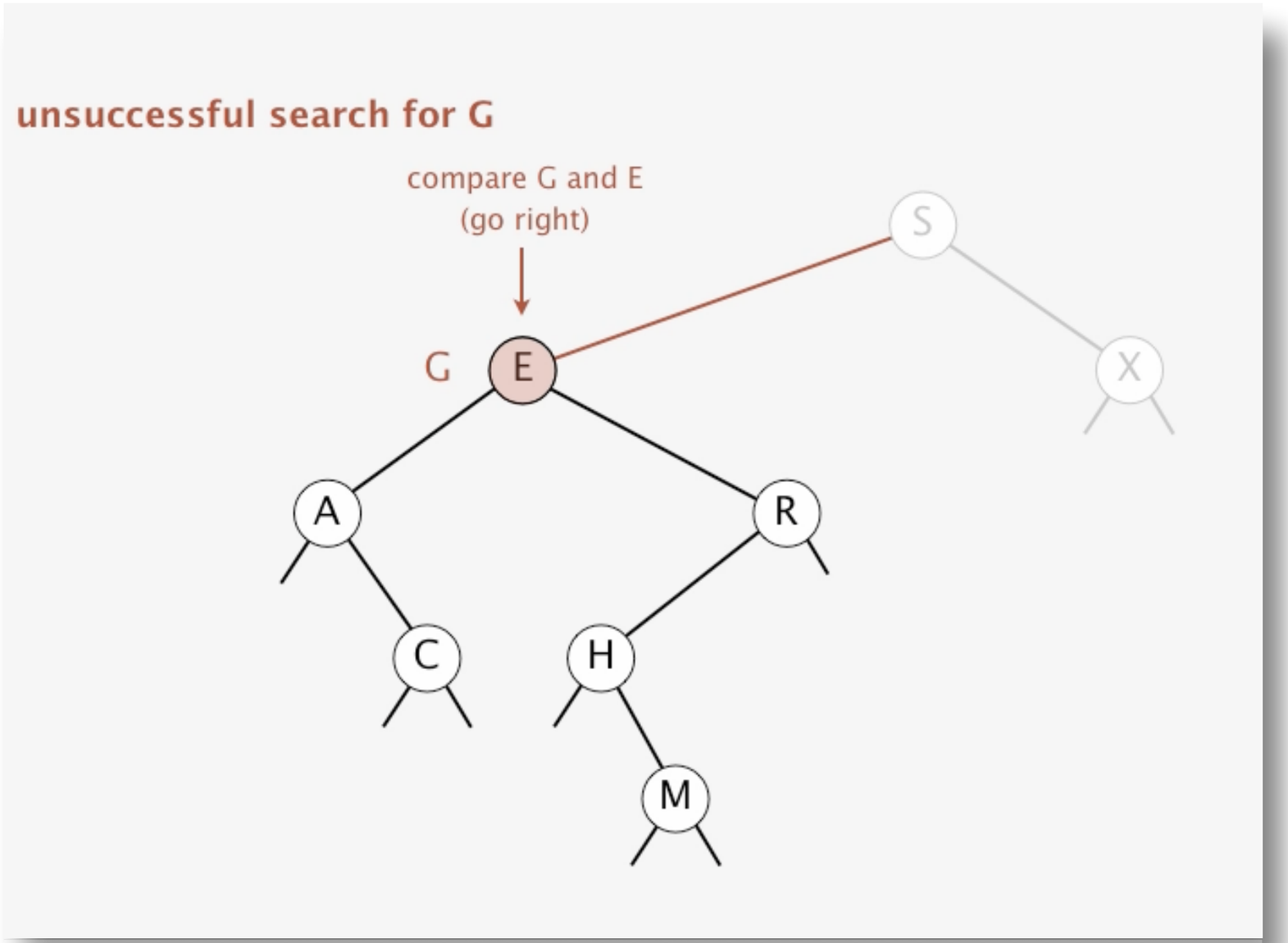
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



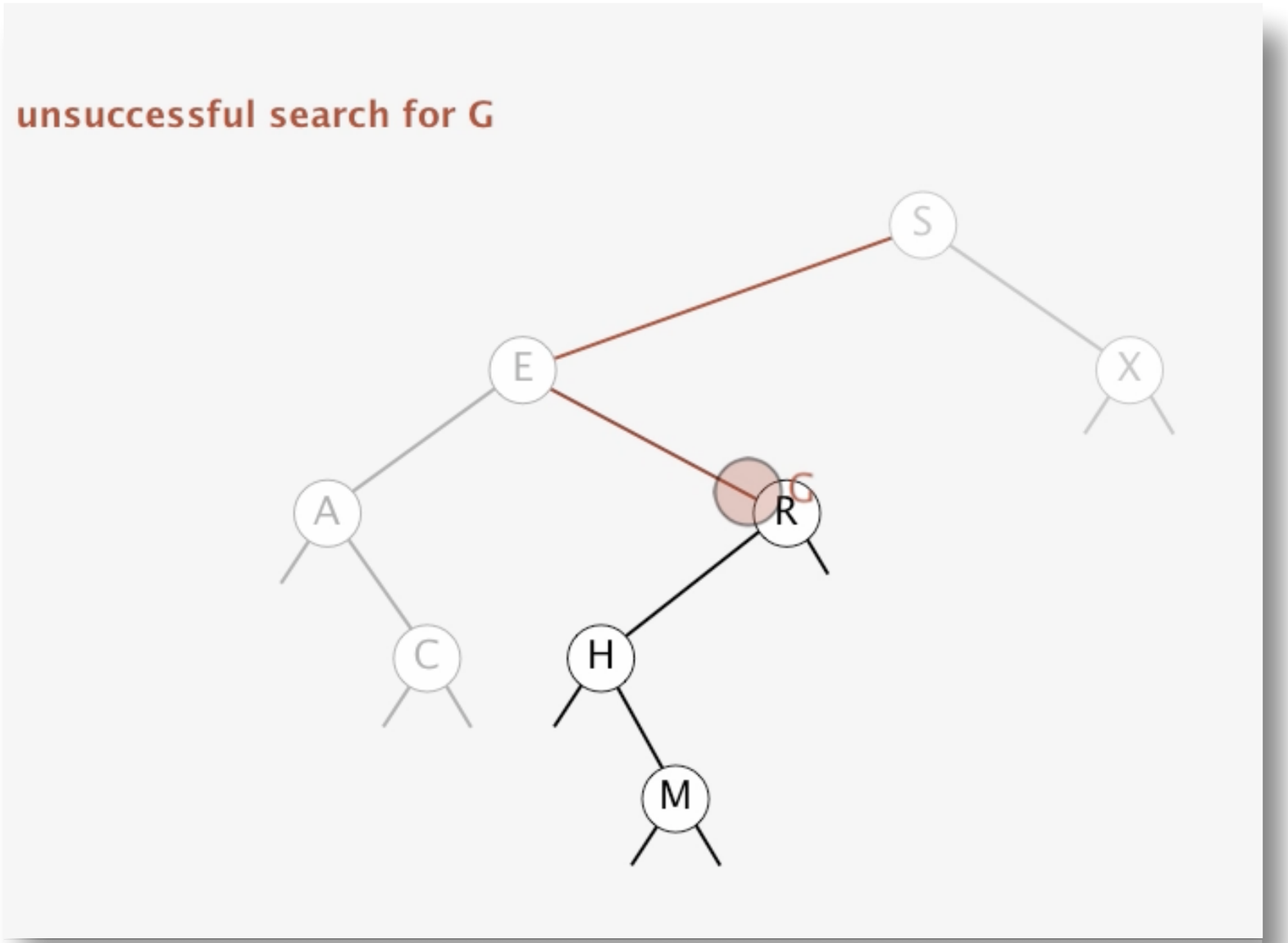
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



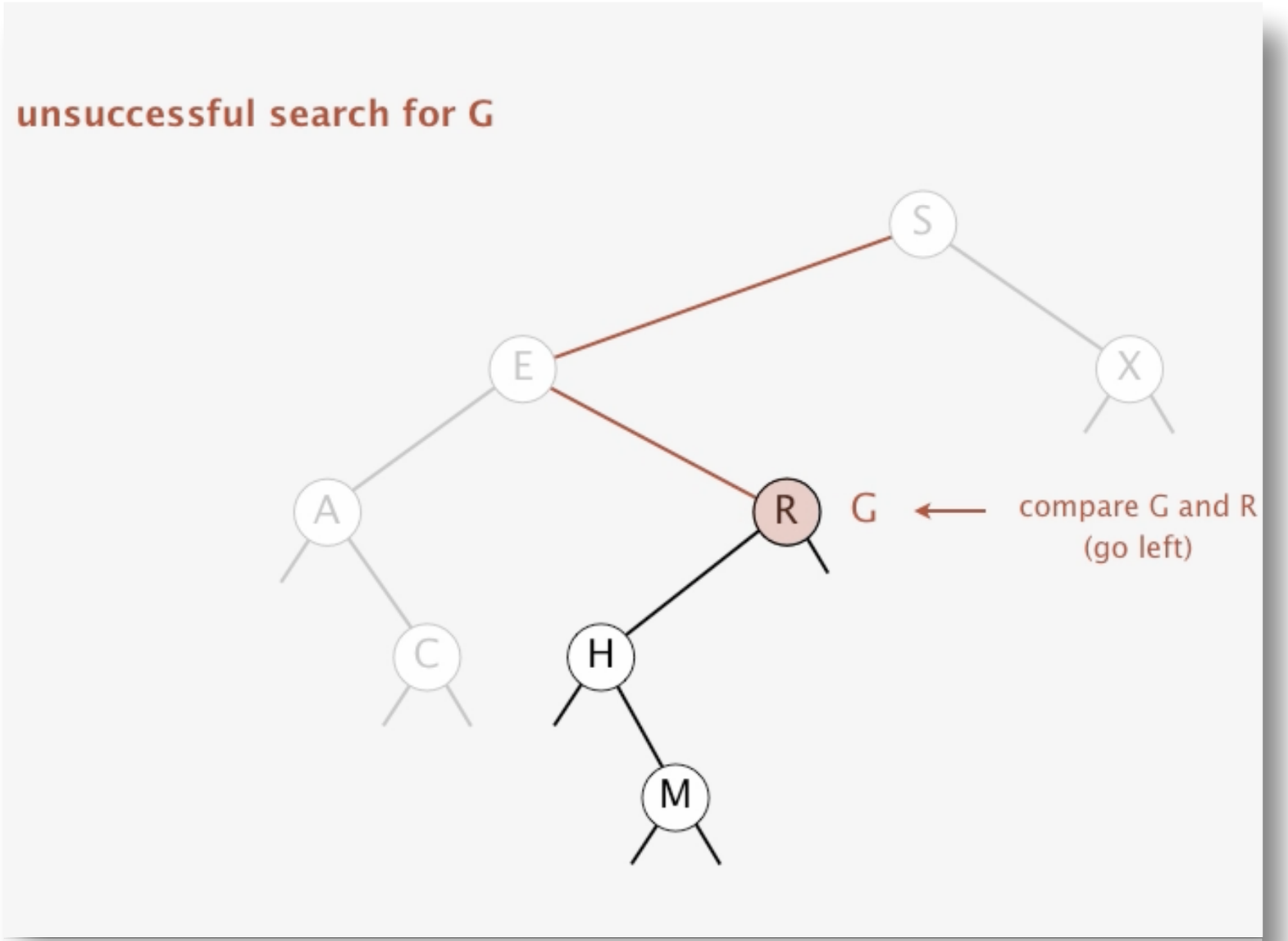
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



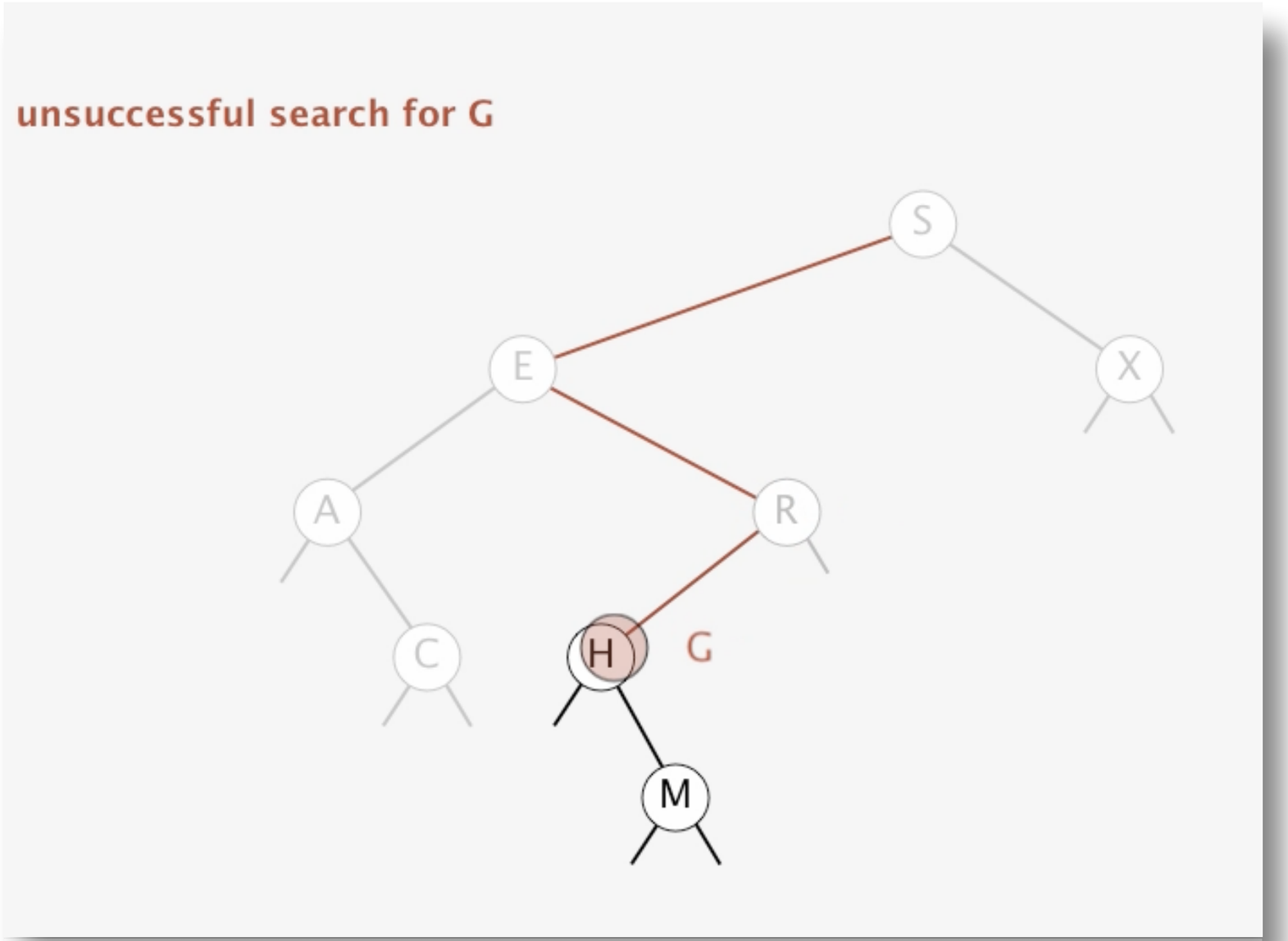
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



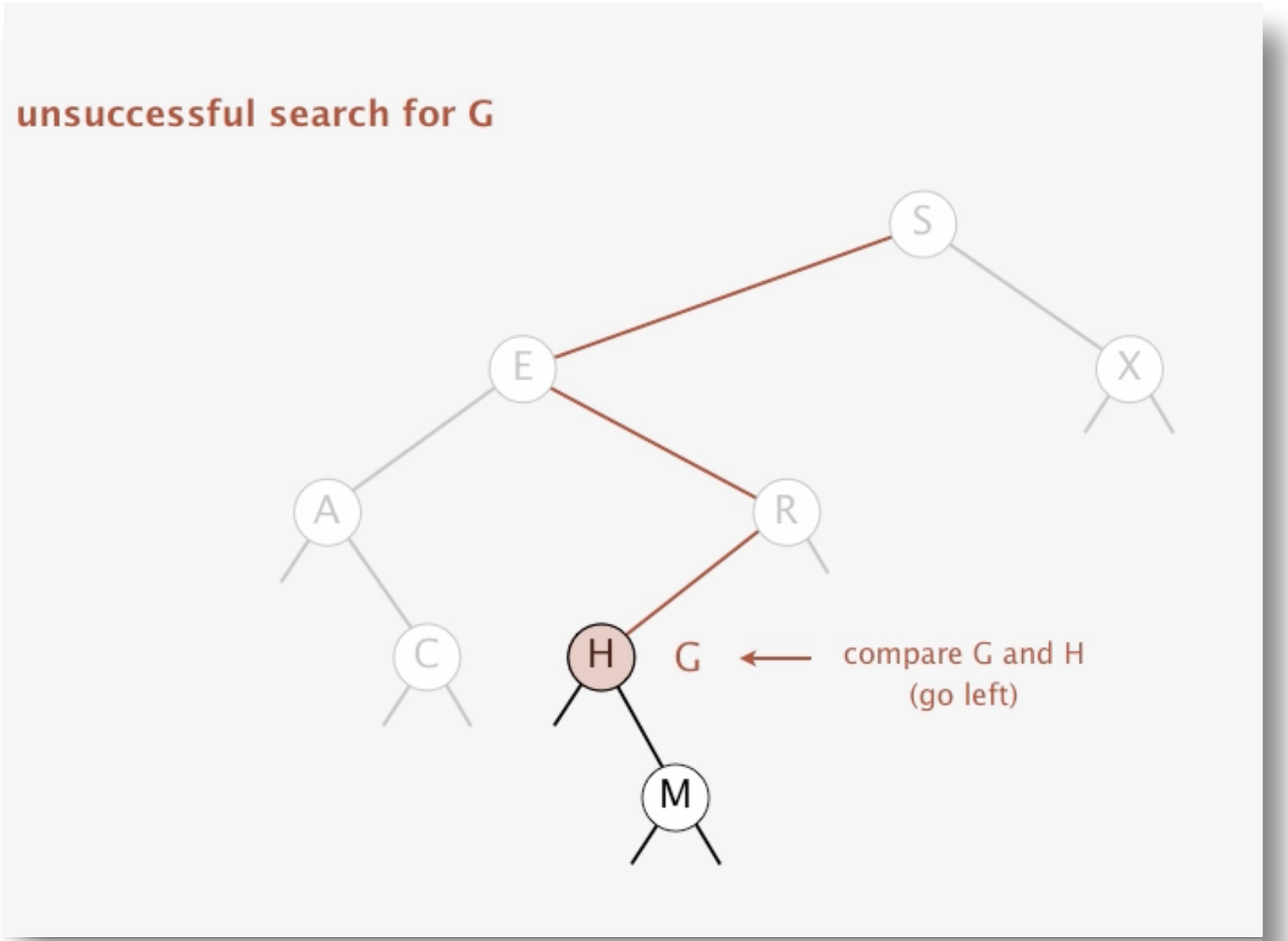
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



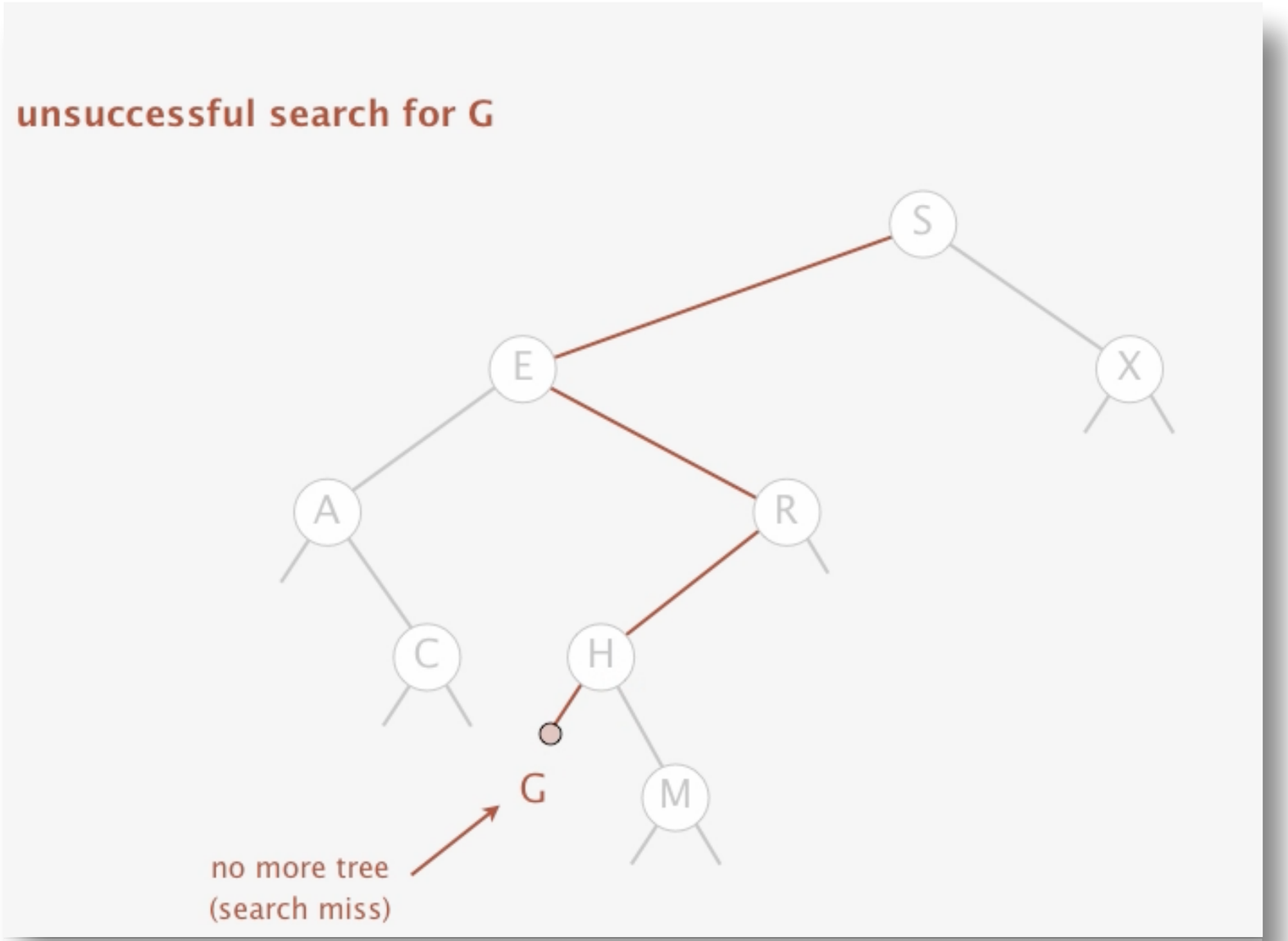
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



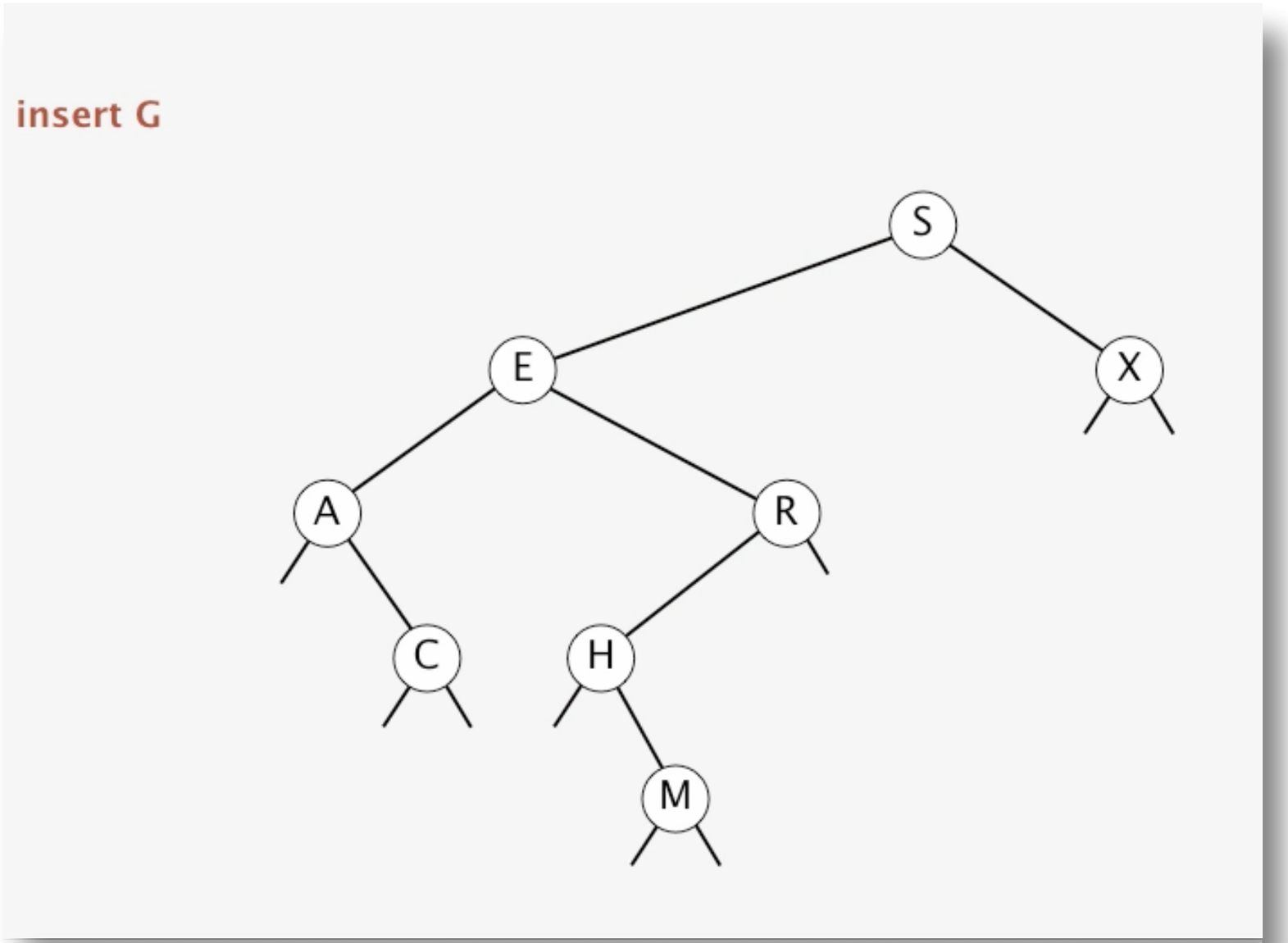
Binary Search Tree Demo

Search: If less, go left; if greater, go right; if equal, search hit.



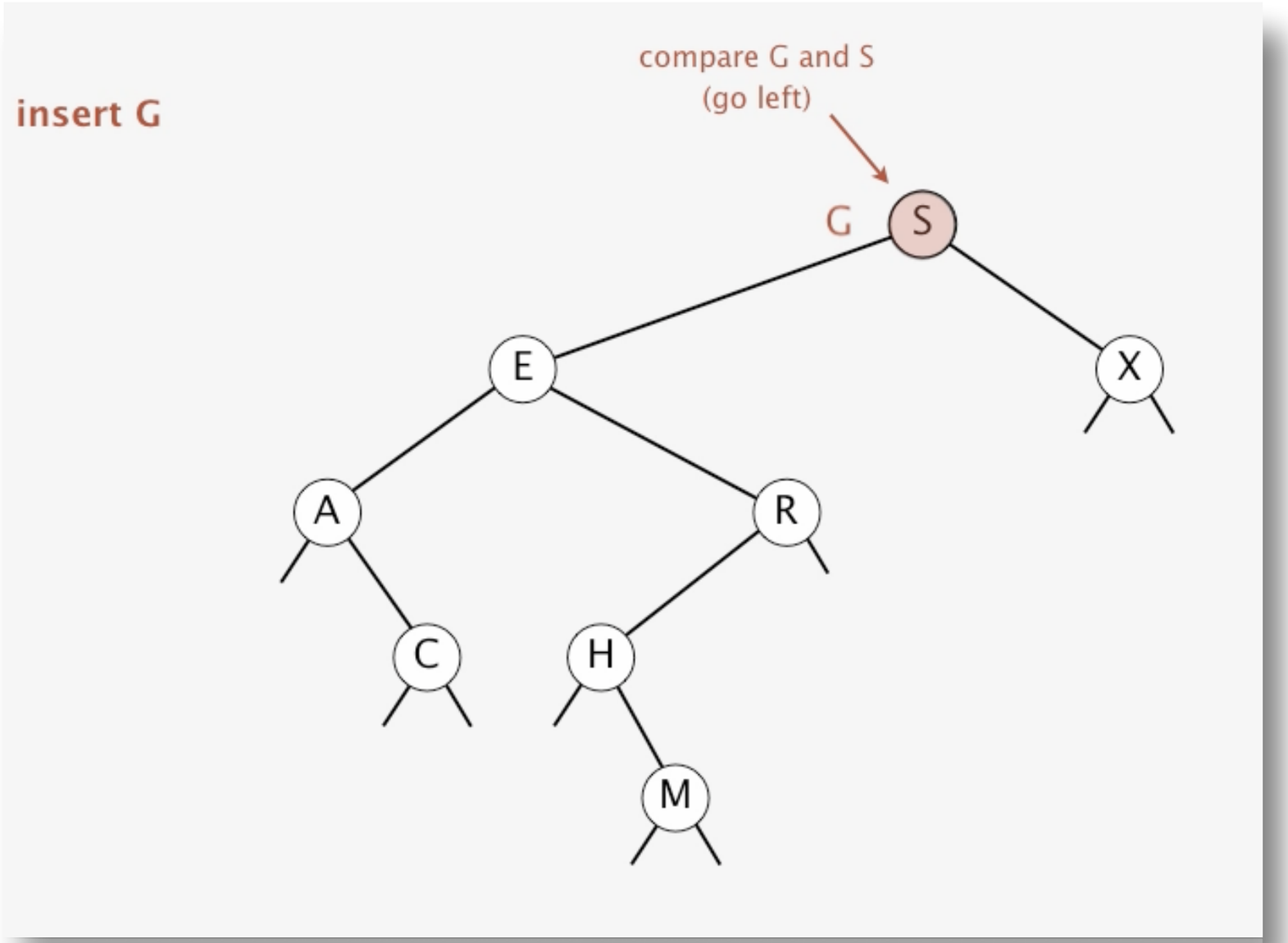
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



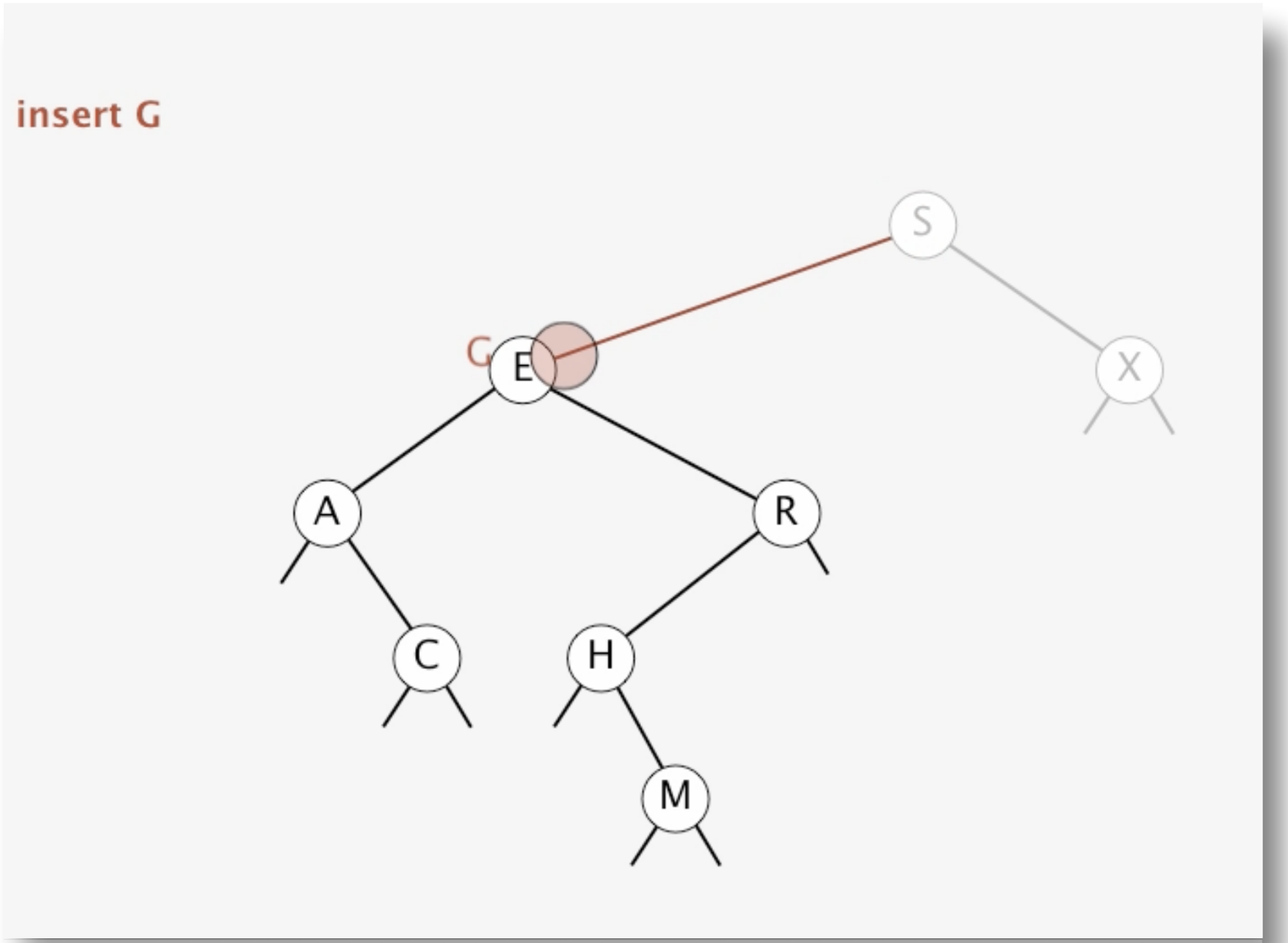
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



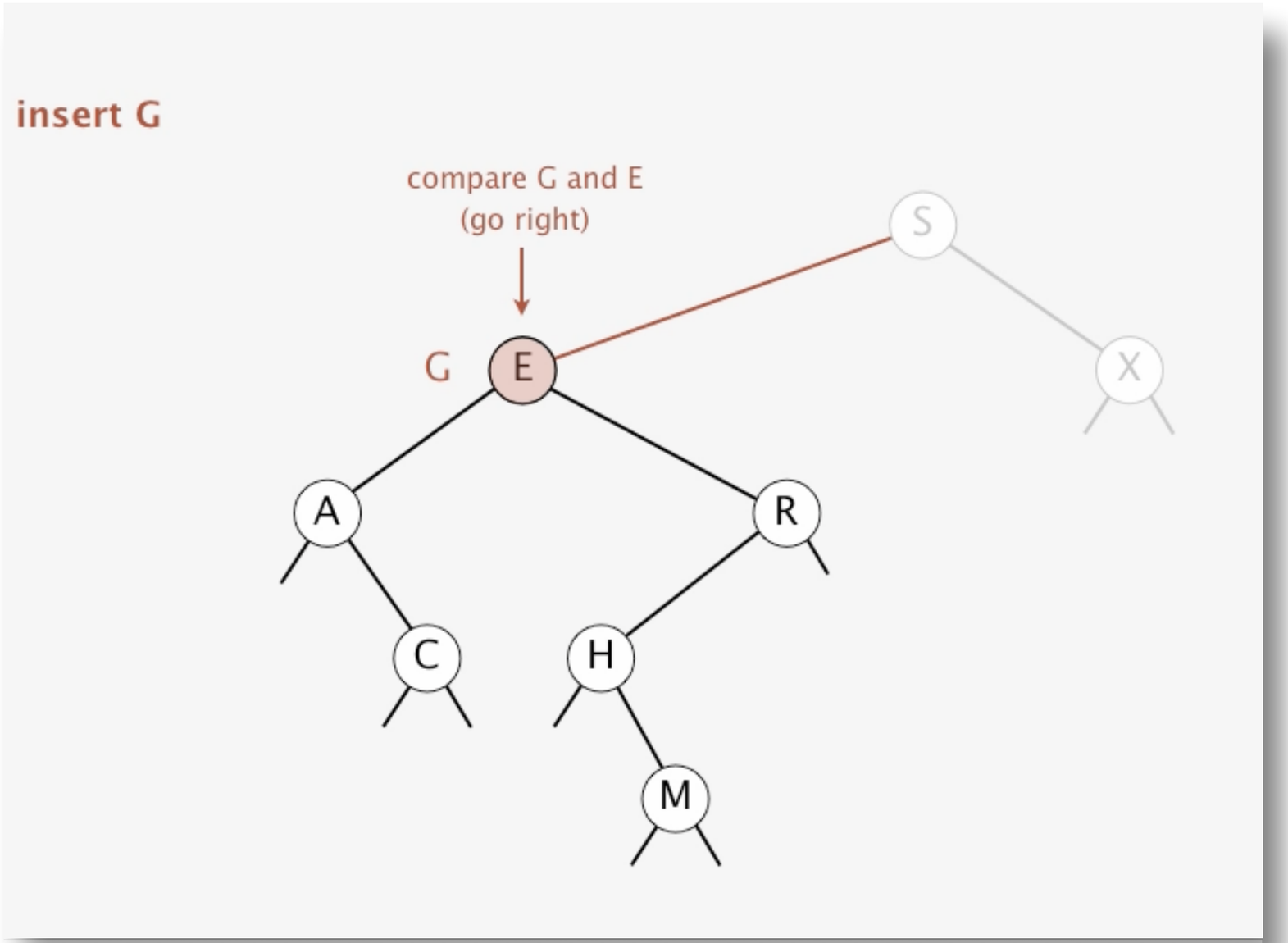
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



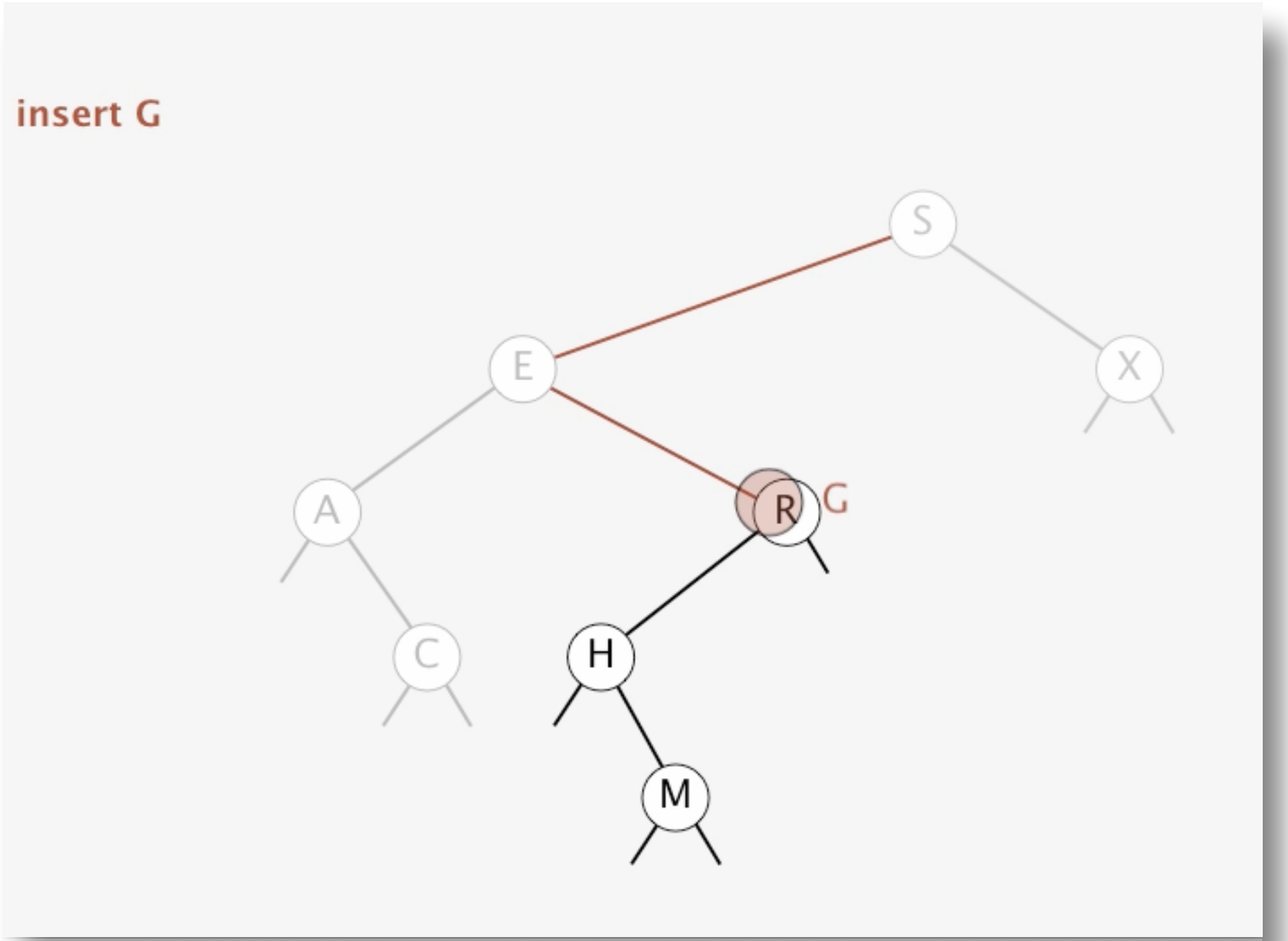
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



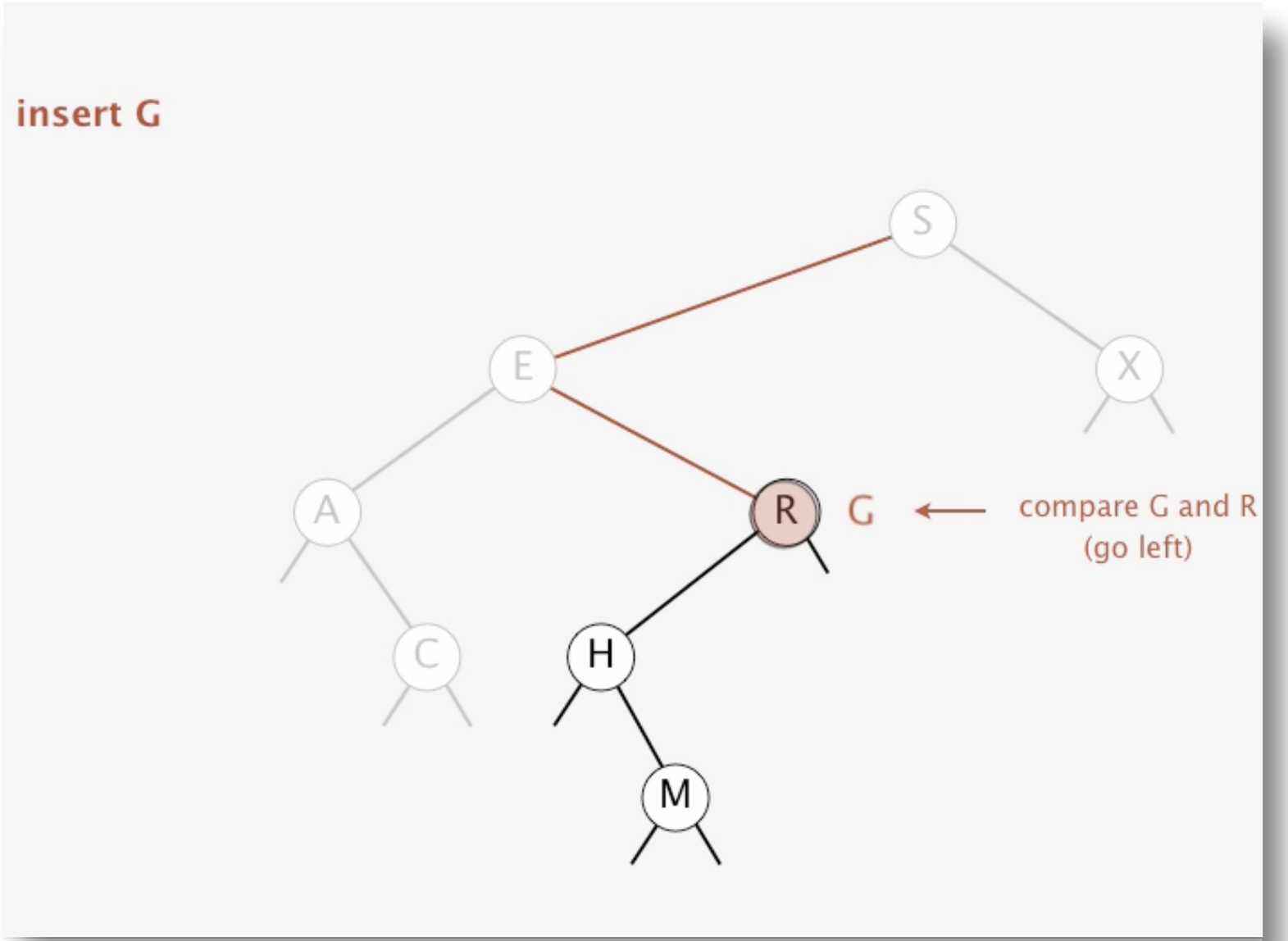
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



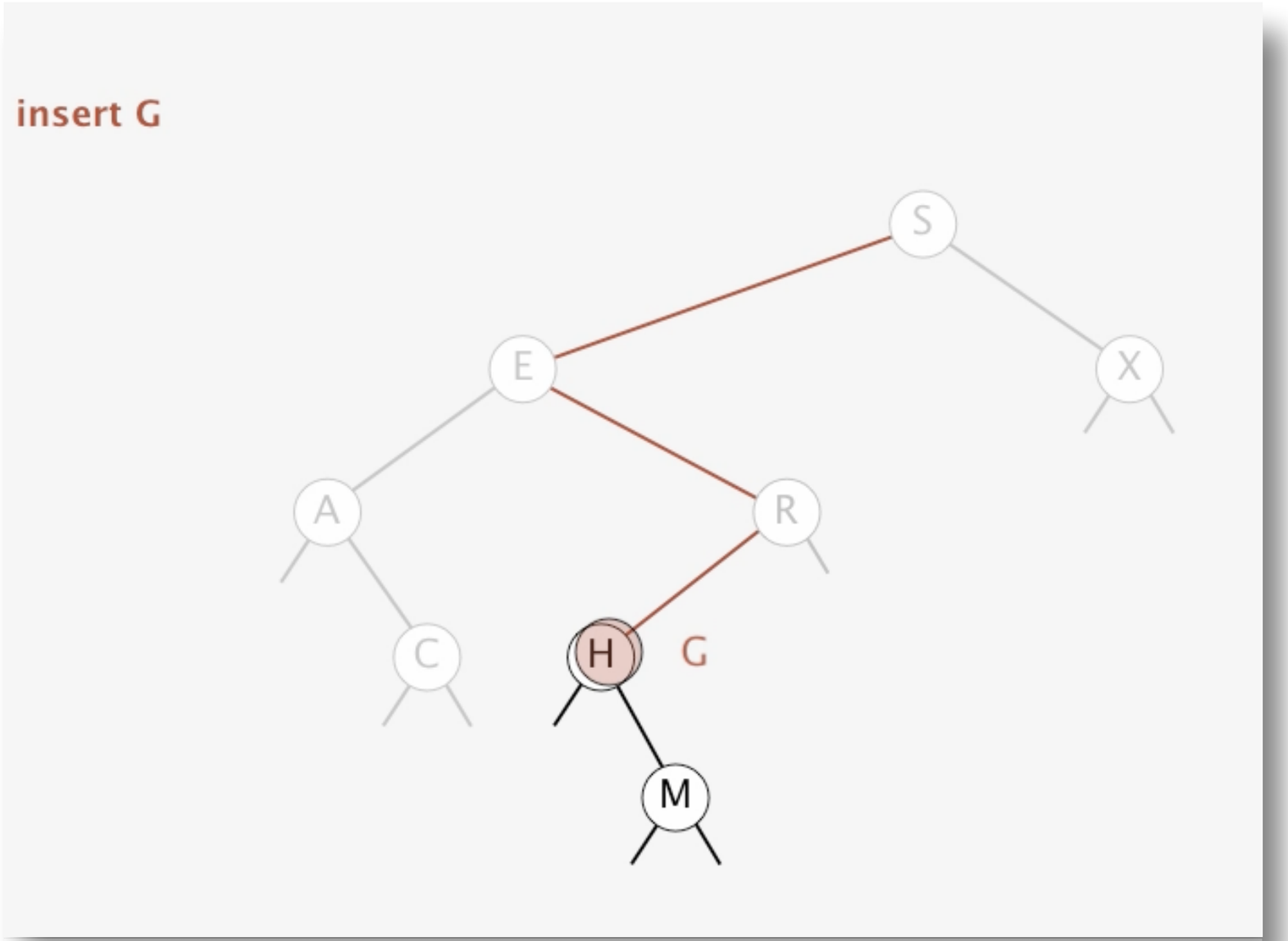
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



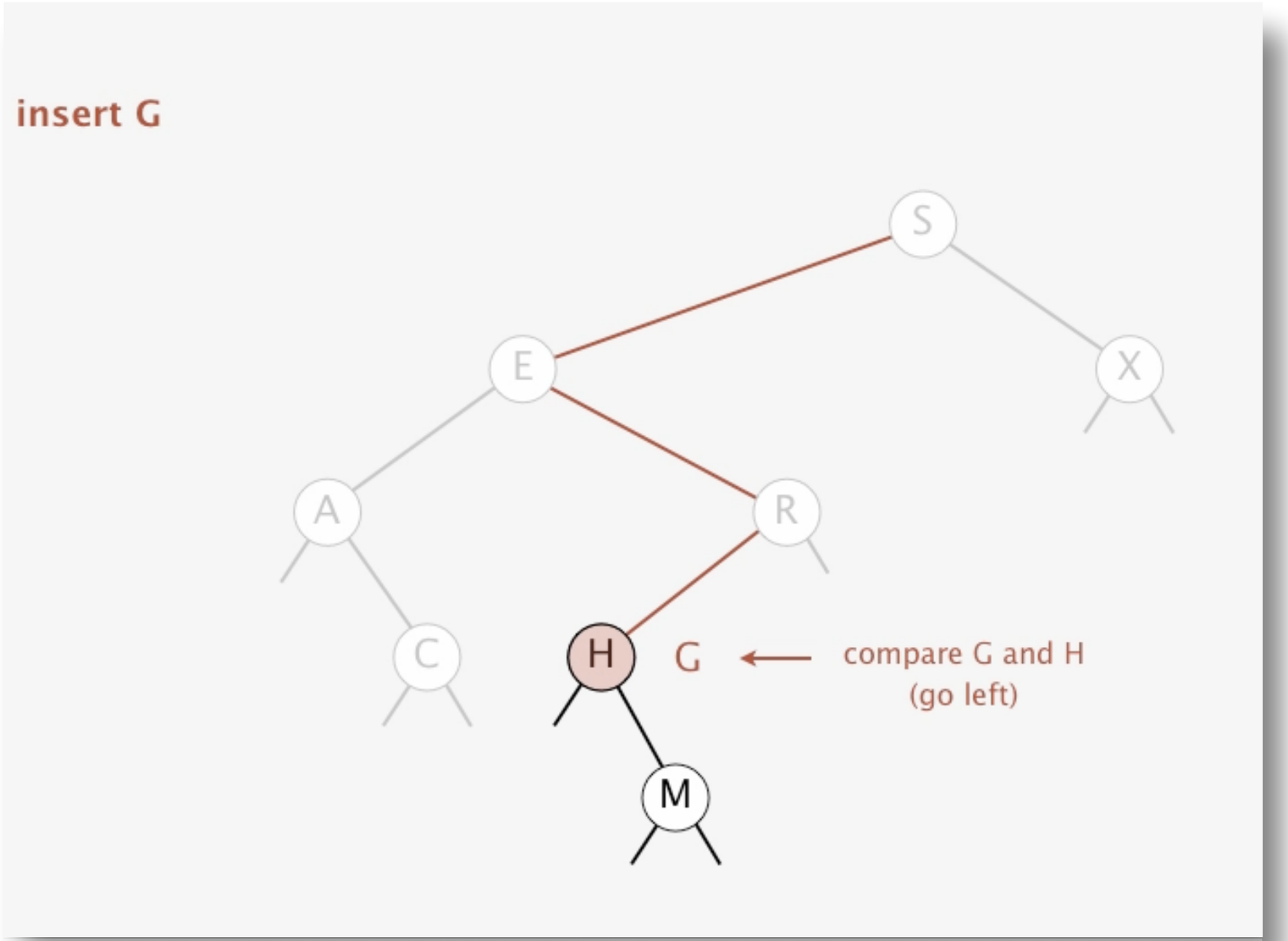
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



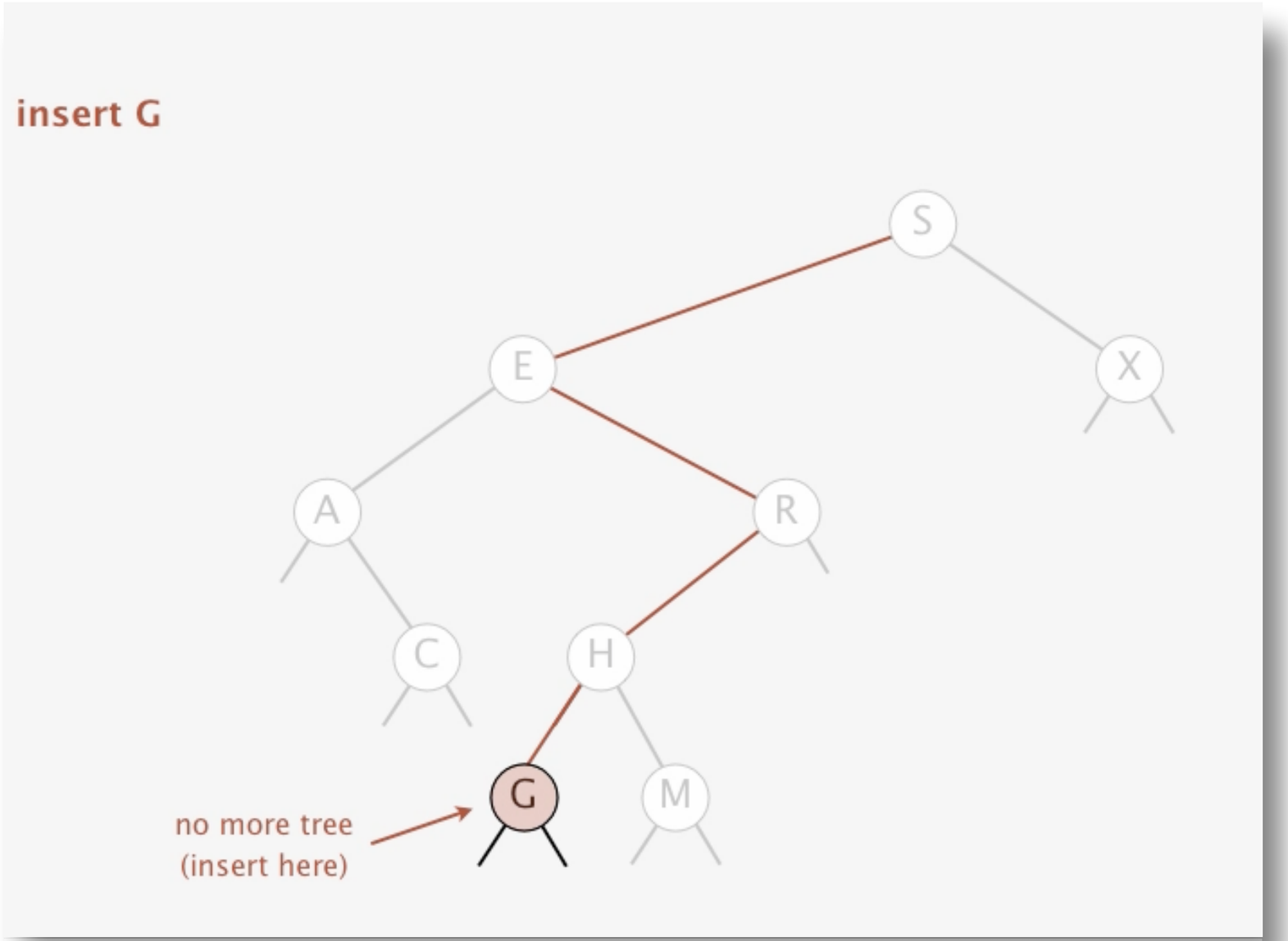
Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.



Binary Search Tree Demo

Insert: If less, go left; if greater, go right; if null, insert.

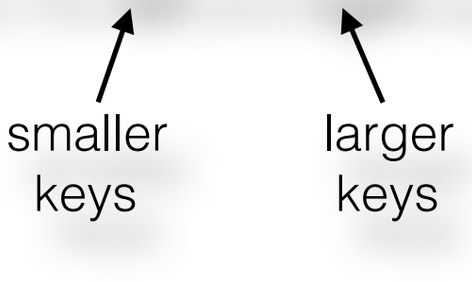


Binary Search Tree Representation in Java

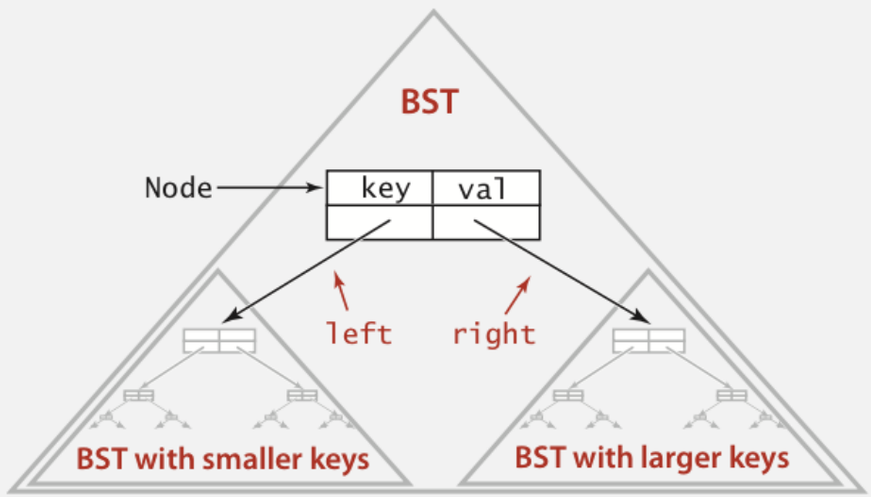
Java Definition: A BST is a reference to a root Node.

A Node is composed of four fields

- A **Key** and a **Value**.
- A reference to the **left** and **right** subtree




```
private class Node
{
   private Key key;
   private Value val;
   private Node left, right;
   public Node(Key key, Value val)
   {
      this.key = key;
      this.val = val;
   }
}
```



Binary search tree

Key and Value are generic types; Key is Comparable

Binary Search Tree Implementation (Skeleton)

```
public class BST<Key extends Comparable<Key>, Value>
{
    private Node root;

   private class Node
   {  /* see previous slide */  }

   public void put(Key key, Value val)
   {  /* see next slides */  }

   public Value get(Key key)
   {  /* see next slides */  }

   public void delete(Key key)
   {  /* see next slides */  }

   public Iterable<Key> iterator()
   {  /* see next slides */  }

}
```

root of BST

Binary Search Tree Search Implementation

Get: Return value corresponding to given key, or null if no such key.

```
public Value get(Key key)
{
   Node x = root;
   while (x != null)
   {
      int cmp = key.compareTo(x.key);
      if      (cmp  < 0) x = x.left;
      else if (cmp  > 0) x = x.right;
      else if (cmp == 0) return x.val;
   }
   return null;
}
```

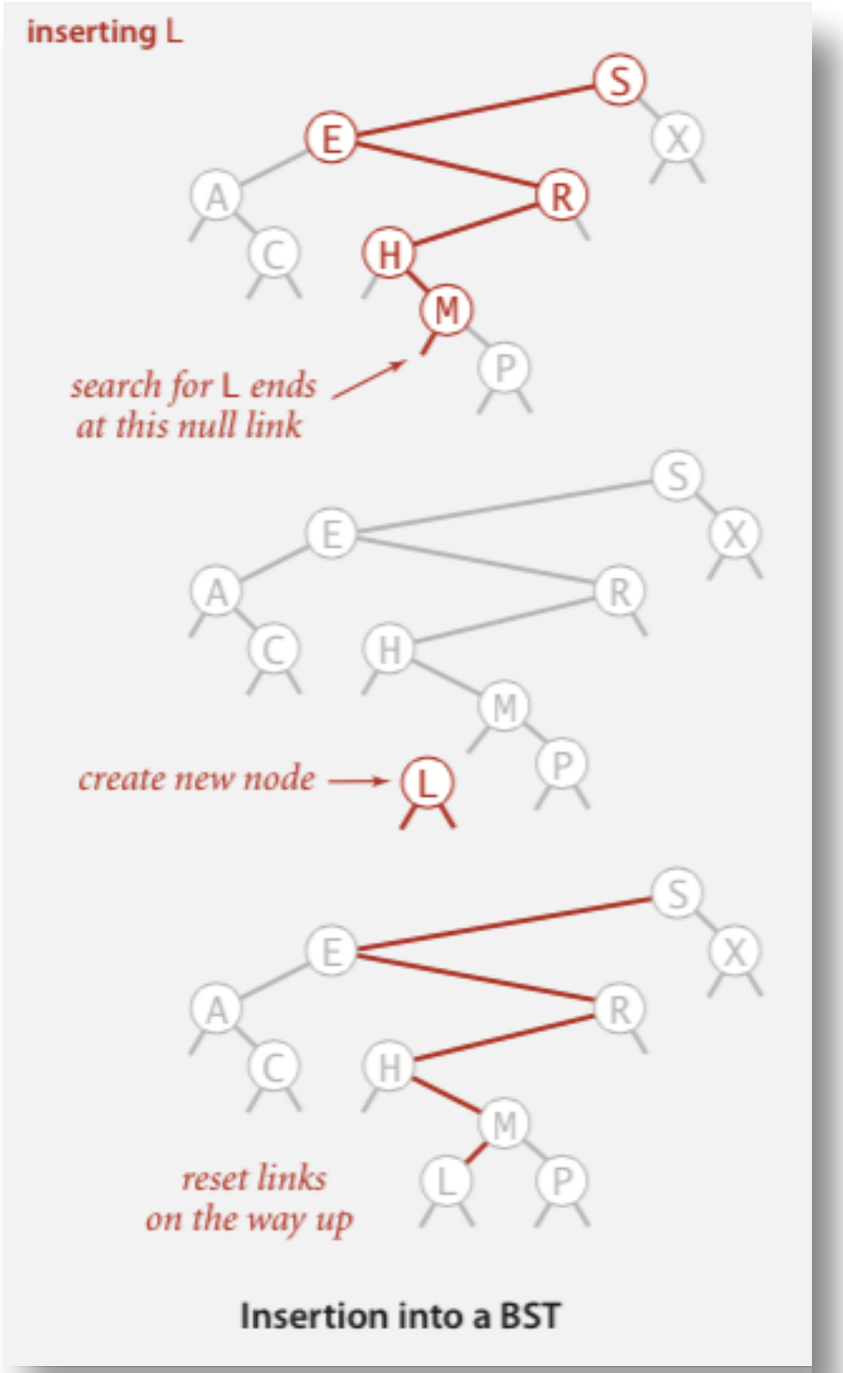
Cost: Number of compares is equal to 1 + depth of node.

Binary Search Tree Representation in Java

Put: Associate value with key.

Search for key, then two cases

- Key in tree ⇒ reset value.
- Key not in tree ⇒ add new node.



Insertion into a BST

Binary Search Tree Search Implementation

Get: Return value corresponding to given key, or null if no such key.

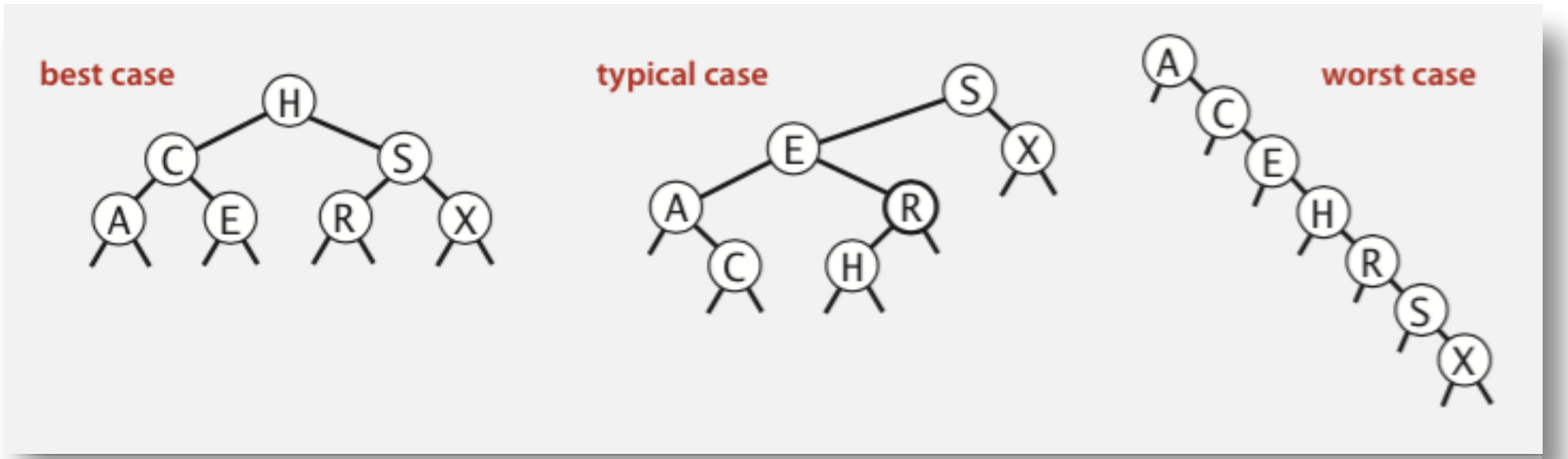
```
public void put(Key key, Value val)
{  root = put(root, key, val);  }

private Node put(Node x, Key key, Value val)
{
   if (x == null) return new Node(key, val);
   int cmp = key.compareTo(x.key);
   if      (cmp  < 0)
      x.left  = put(x.left,  key, val);
   else if (cmp  > 0)
      x.right = put(x.right, key, val);
   else if (cmp == 0)
      x.val = val;
   return x;
}
```

concise, but tricky, recursive code; read carefully!

Cost: Number of compares is equal to 1 + depth of node.

Tree Shape

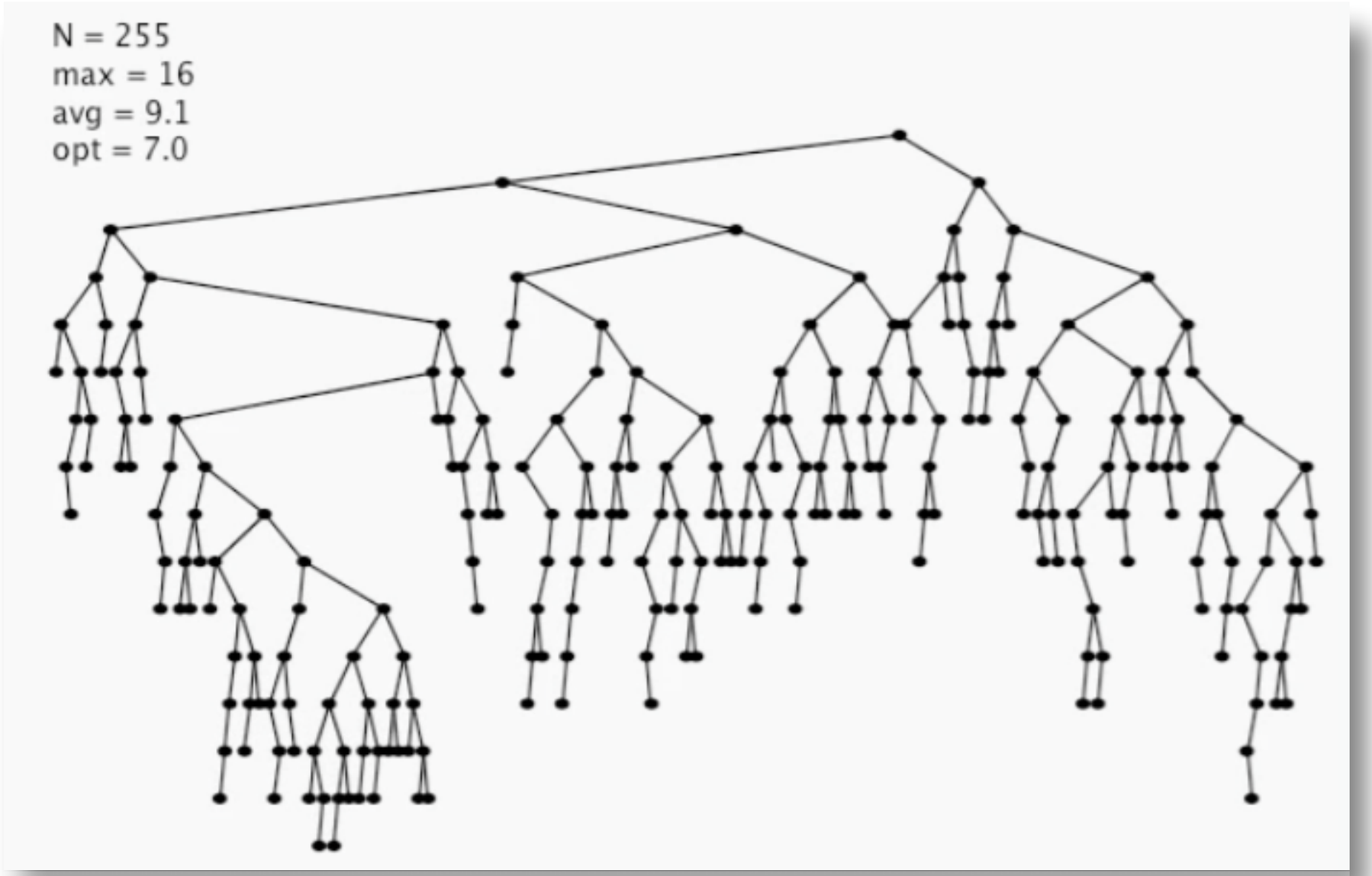
Many BSTs correspond to same set of keys
Number of compares for search/insert is equal to 1 + depth of node



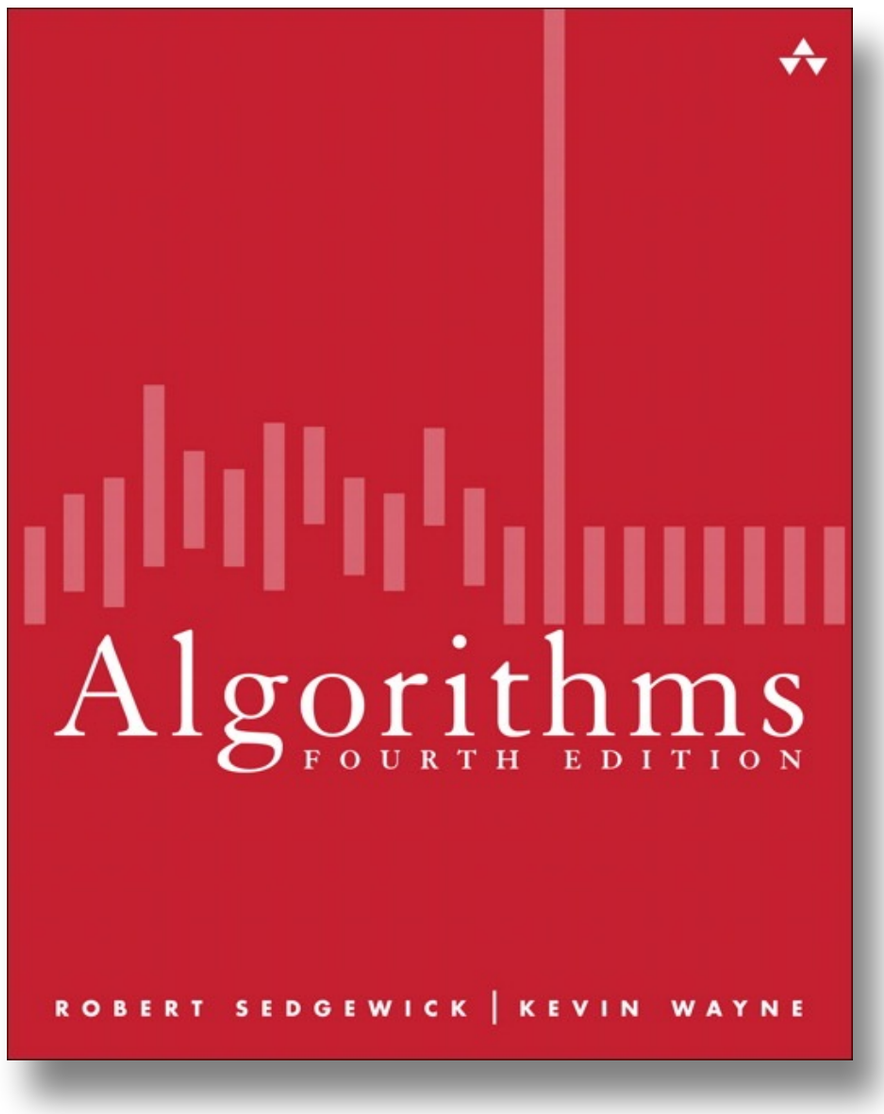
Bottom Line: Tree shape depends on order of insertion.

BST Insertion: Random Order Visualization

Ex: Insert keys in random order.



Lecture Overview

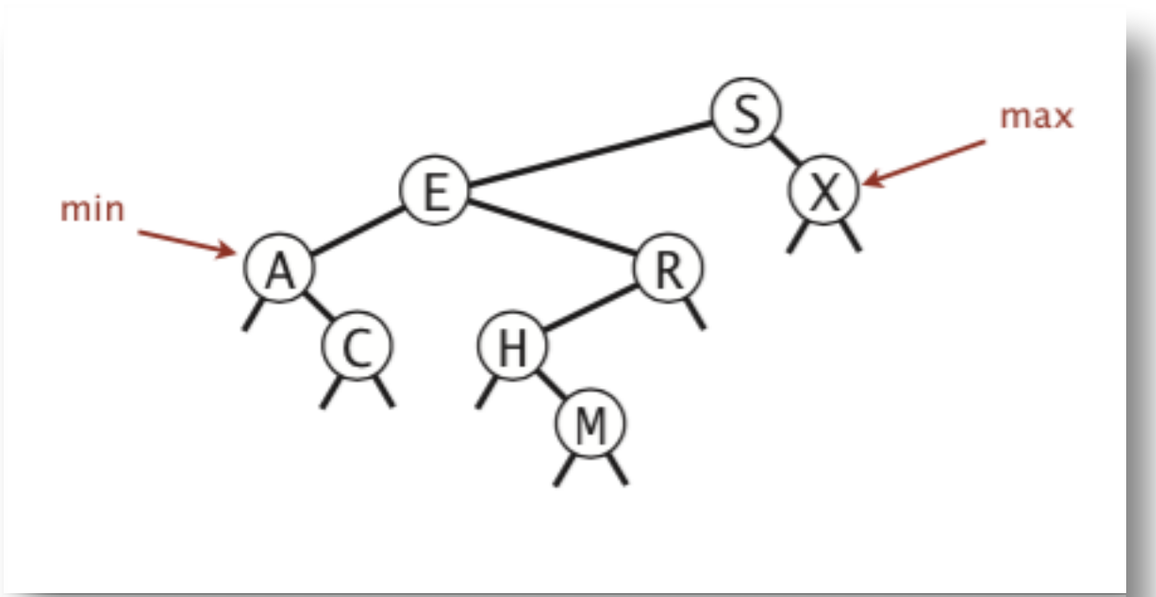


3.2 Binary Search Tree

- BSTs
- Ordered operations
- Deletion

Minimum and Maximum

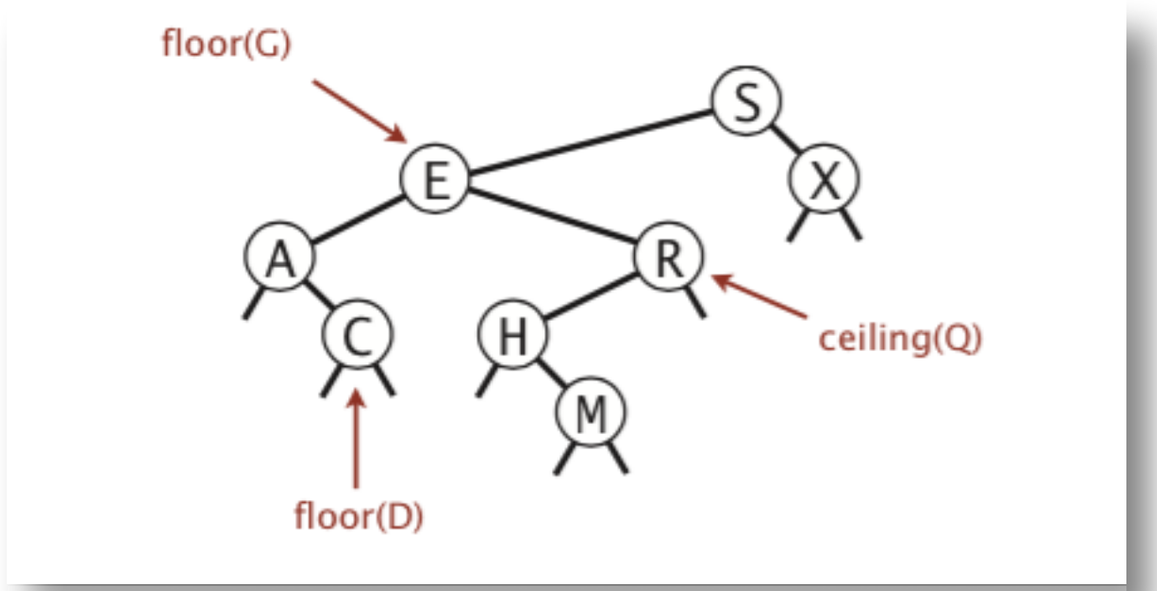
Minimum: Smallest key in table.
Maximum: Largest key in table.



Q: How to find the min / max ?

Floor and Ceiling

Floor: Largest key ≤ a given key.
Ceiling: Smallest key ≥ a given key.



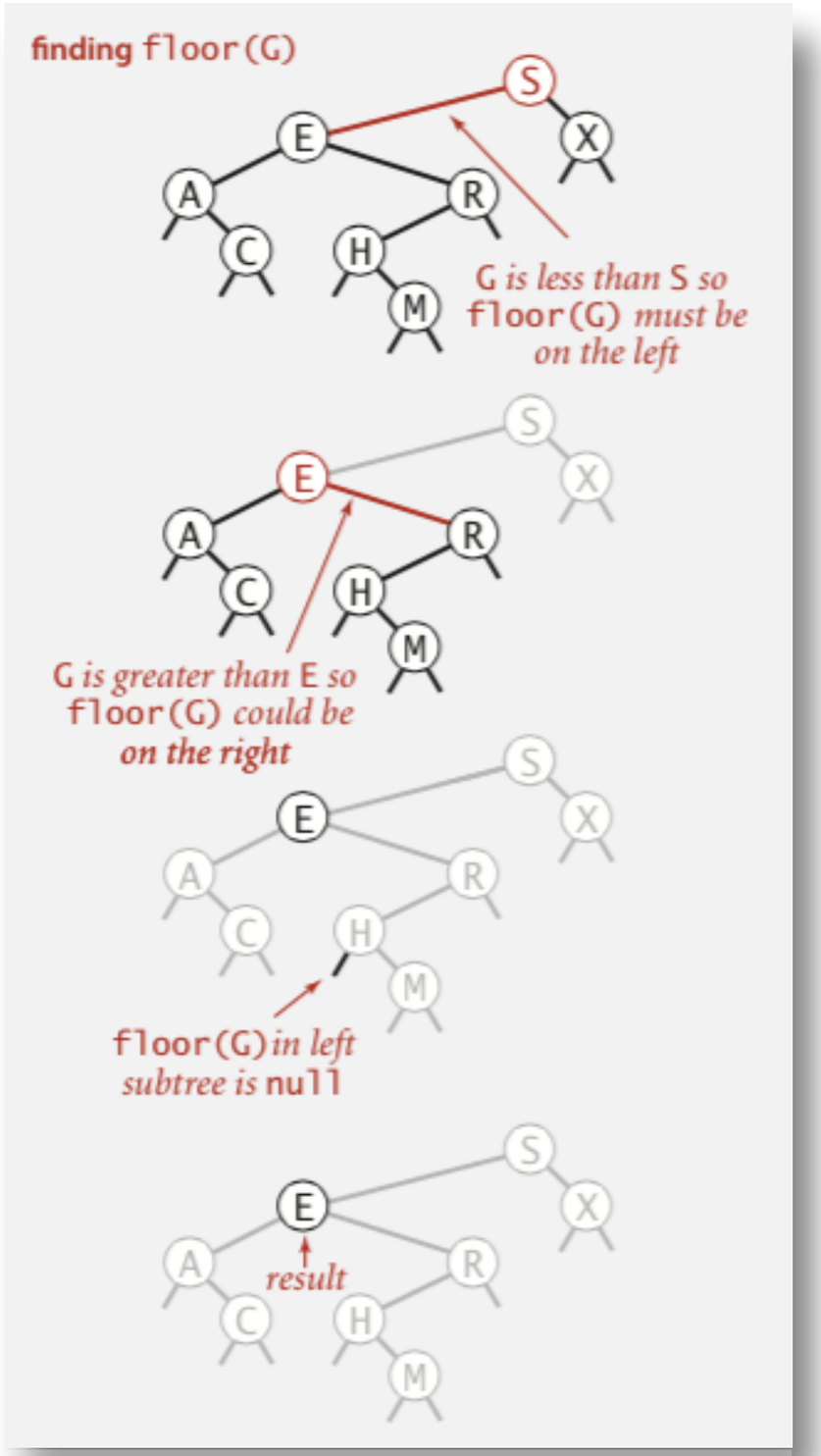
Q: How to find the floor / ceiling ?

Computing the Floor

Case 1: [k equals the key in the node] The floor of k is k.

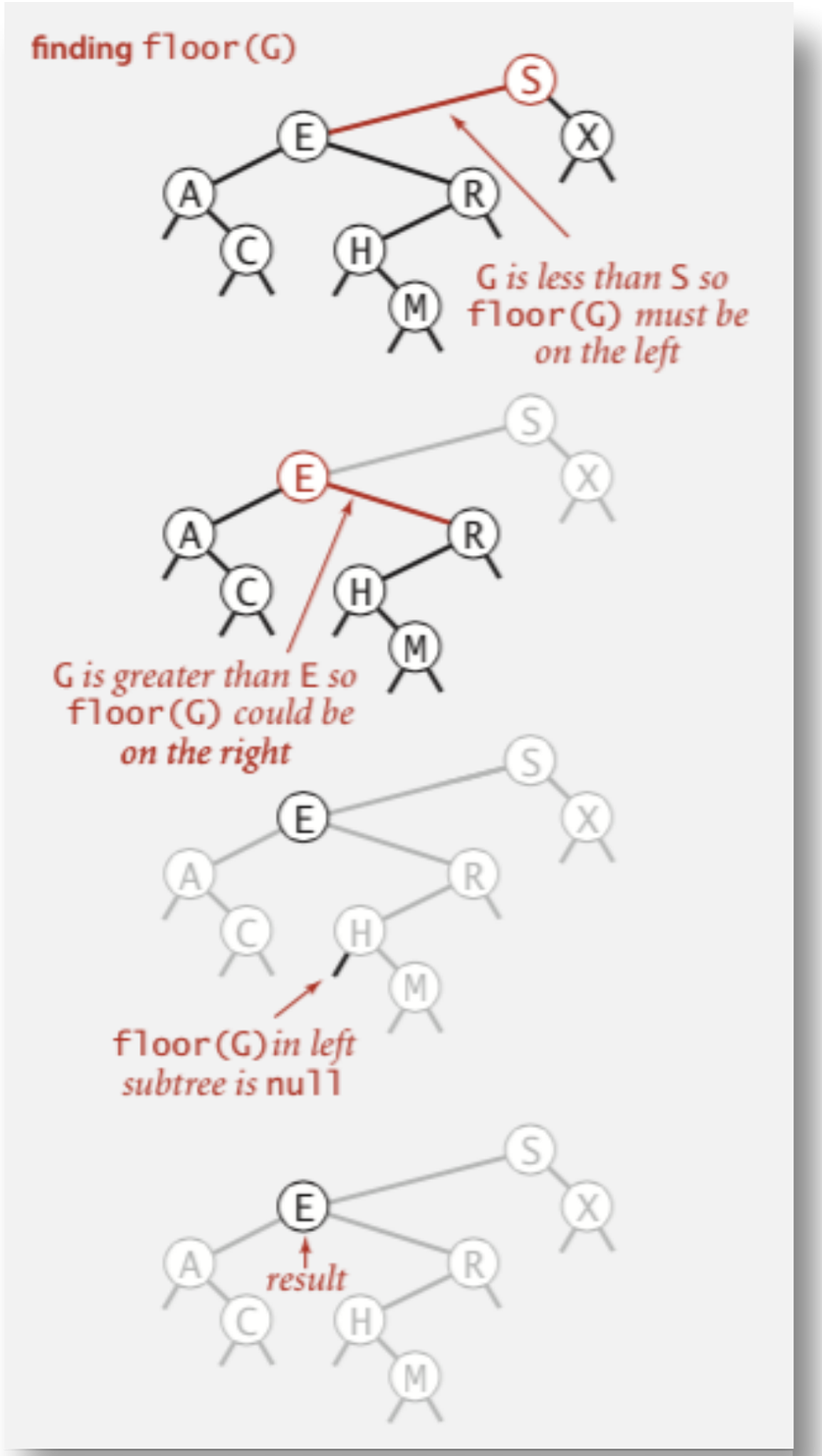
Case 2: [k is less than the key in the node] The floor of k is in the left subtree.

Case 3: [k is greater than the key in the node] The floor of k is in the right subtree
(if there is any key ≤ k in right subtree); otherwise it is the key in the node..



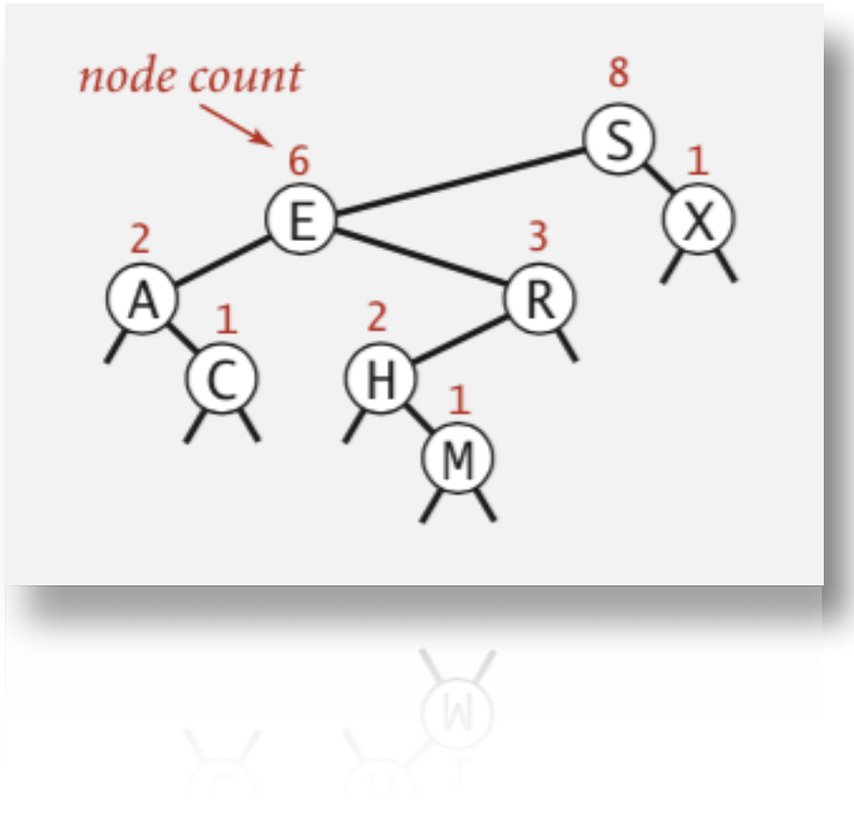
Computing the Floor

```
public Key floor(Key key)
{
   Node x = floor(root, key);
   if (x == null) return null;
   return x.key;
}
private Node floor(Node x, Key key)
{
   if (x == null) return null;
   int cmp = key.compareTo(x.key);

   if (cmp == 0) return x;

   if (cmp < 0)  return floor(x.left, key);

   Node t = floor(x.right, key);
   if (t != null) return t;
   else           return x;

}
```



Rank and Select

Q: How to implement rank() and select() efficiently ?



A: In each node, we store the number of nodes in the subtree rooted at that node; to implement size(), return the count at the root

BST Implementation: Subtree Counts

```
private class Node
{
   private Key key;
   private Value val;
   private Node left;
   private Node right;
   private int count;
}
```

number of nodes in subtree

```
public int size()
{  return size(root);  }

private int size(Node x)
{
   if (x == null) return 0;
   return x.count;
}
```

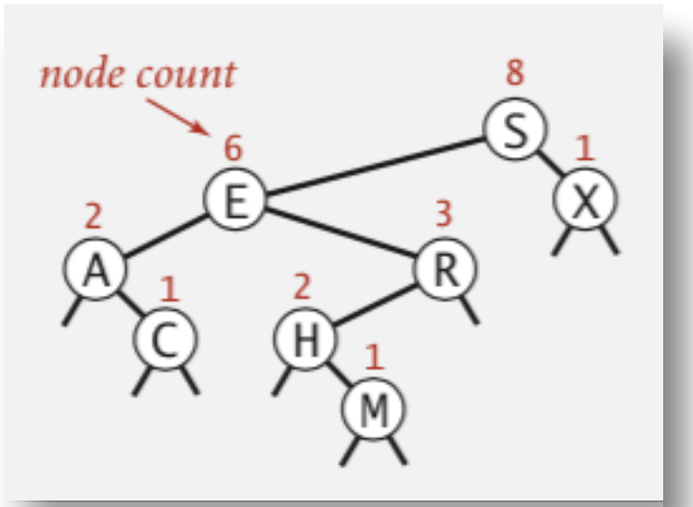
ok to call when x is null

```
private Node put(Node x, Key key, Value val)
{
   if (x == null) return new Node(key, val, 1);
   int cmp = key.compareTo(x.key);
   if      (cmp  < 0) x.left  = put(x.left,  key, val);
   else if (cmp  > 0) x.right = put(x.right, key, val);
   else if (cmp == 0) x.val = val;
   x.count = 1 + size(x.left) + size(x.right);
   return x;
}
```

initialize subtree count to 1

Rank

Rank: How many keys < k ?



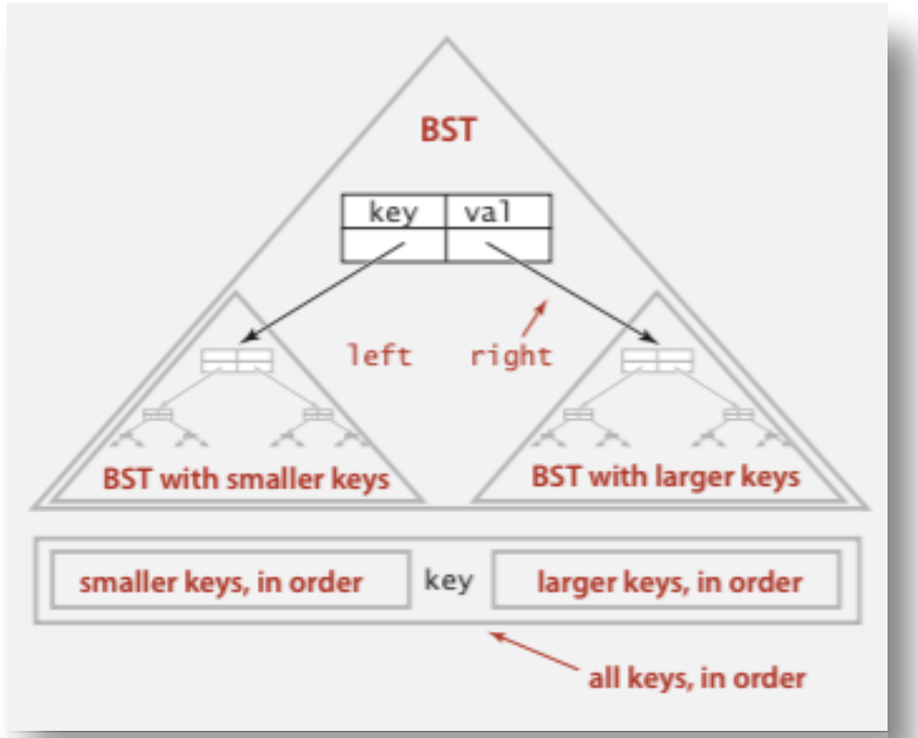
Easy recursive algorithm (3 cases!)

```
public int rank(Key key)
{  return rank(key, root);  }

private int rank(Key key, Node x)
{
   if (x == null) return 0;
   int cmp = key.compareTo(x.key);
   if      (cmp  < 0) return rank(key, x.left);
   else if (cmp  > 0) return 1 + size(x.left) + rank(key, x.right);
   else if (cmp == 0) return size(x.left);
}
```

Inorder Traversal

- Traverse left subtree.
- Enqueue key.
- Traverse right subtree.

```
public Iterable<Key> keys()
{
    Queue<Key> q = new Queue<Key>();
    inorder(root, q);
    return q;
}

private void inorder(Node x, Queue<Key> q)
{
   if (x == null) return;
   inorder(x.left, q);
   q.enqueue(x.key);
   inorder(x.right, q);
}
```



Property: Inorder traversal of a BST yields keys in ascending order

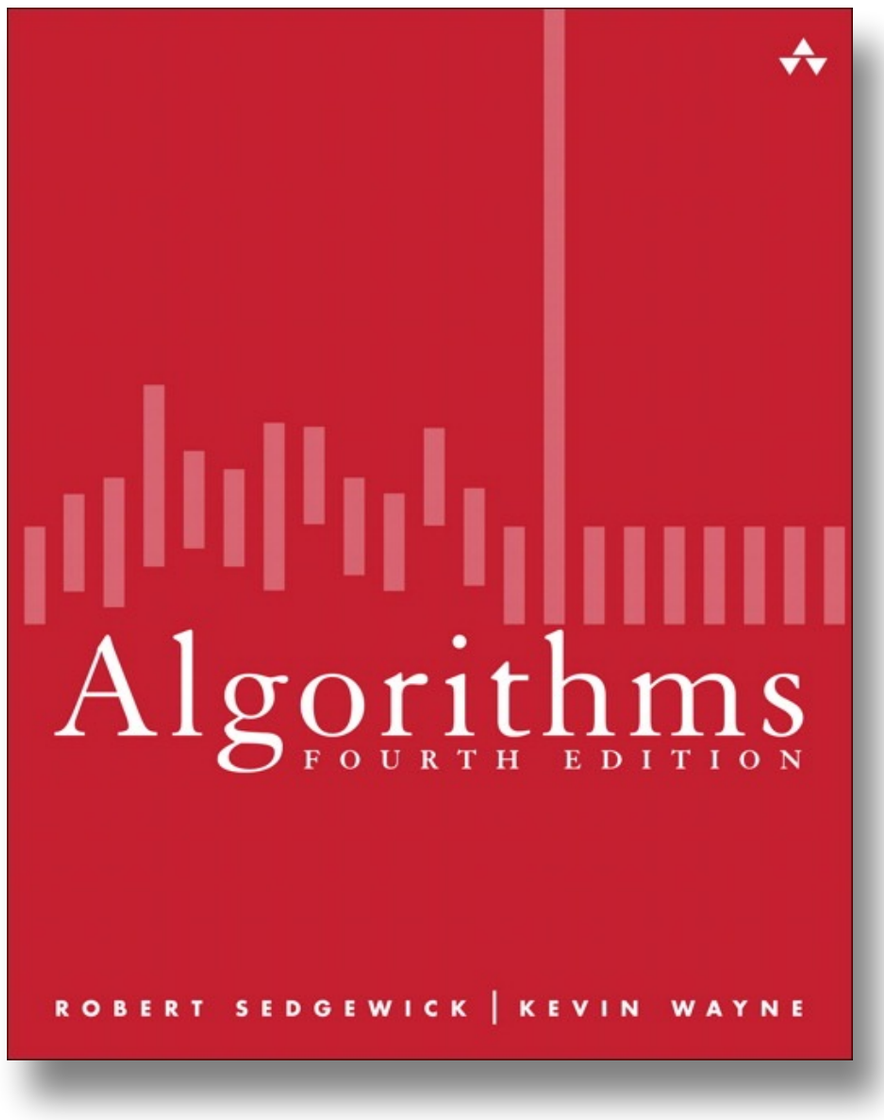
BST: Ordered Symbol Table Operations Summary

	sequential search	binary search	BST
search	N	$\lg N$	h
insert	N	N	h
min / max	N	1	h
floor / ceiling	N	$\lg N$	h
rank	N	$\lg N$	h
select	N	1	h
ordered iteration	$N \log N$	N	N

h = height of BST
(proportional to log N
if keys inserted in random order)

order of growth of running time of ordered symbol table operations

Lecture Overview



3.2 Binary Search Tree

- BSTs
- Ordered operations
- Deletion

Search Tree Implementations: Summary

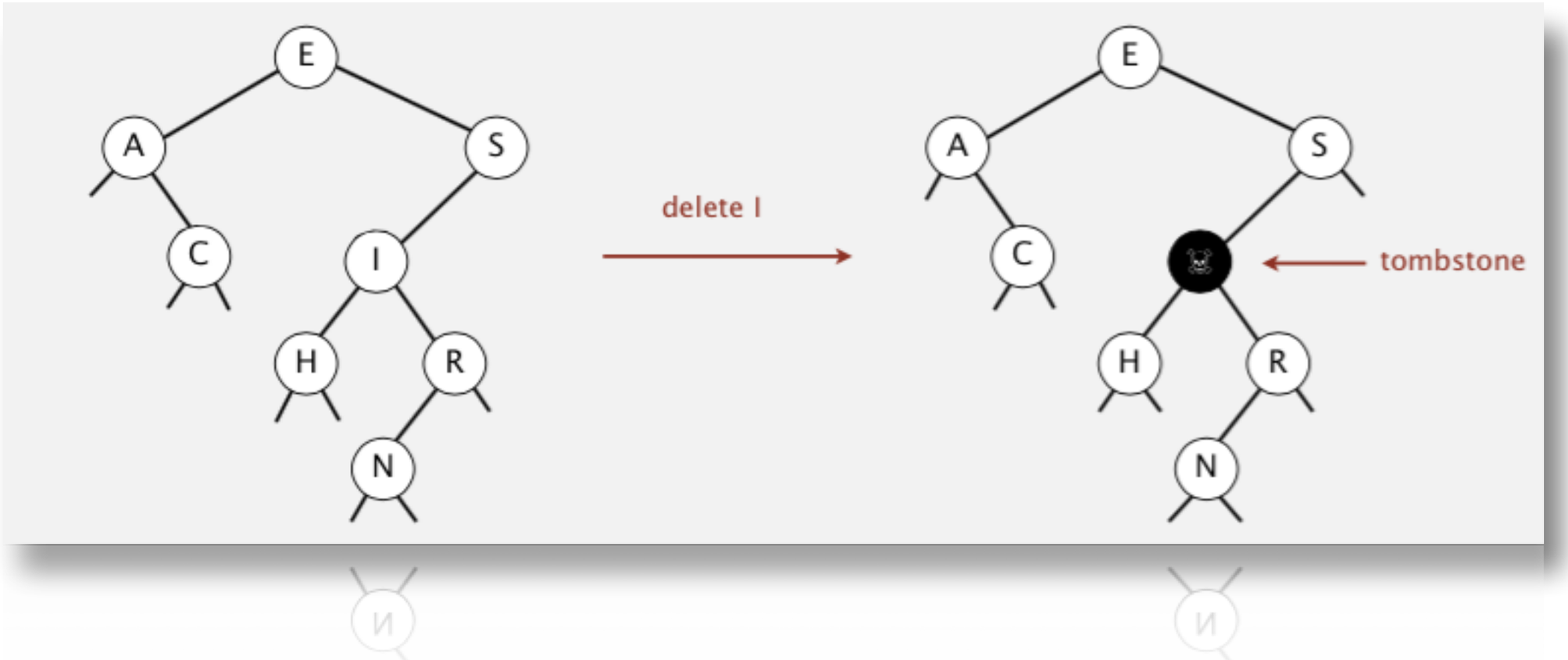
implementation	guarantee			average case			ordered ops?	operations on keys
	search	insert	delete	search hit	insert	delete		
sequential search (linked list)	N	N	N	$\frac{1}{2} N$	N	$\frac{1}{2} N$		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2} N$	$\frac{1}{2} N$	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	? ? ?	✔	`compareTo()`

Next: Deletion in BSTs

BST Deletion: Lazy Approach

To remove a node with a given key:

Set its value to null.
Leave key in tree to guide search (but don't consider it equal in search).



Cost: ~ 2 ln N' per insert, search, and delete (if keys in random order), where N' is the number of key-value pairs ever inserted in the BST.

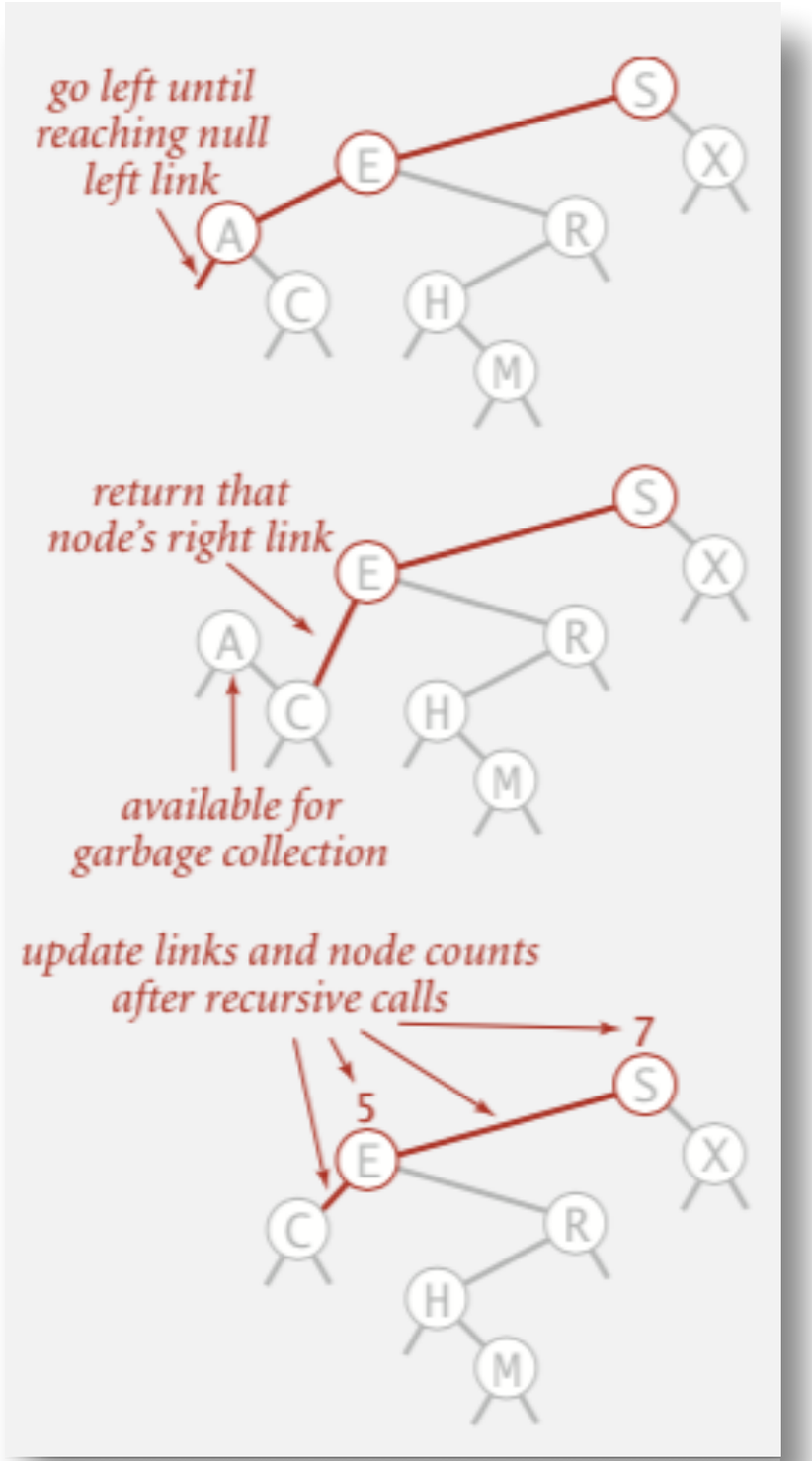
Unsatisfactory solution: Tombstone (memory) overload.

Deleting the Minimum

To delete the minimum key:

Go left until finding a node with a null left link.
Replace that node by its right link.
Update subtree counts.

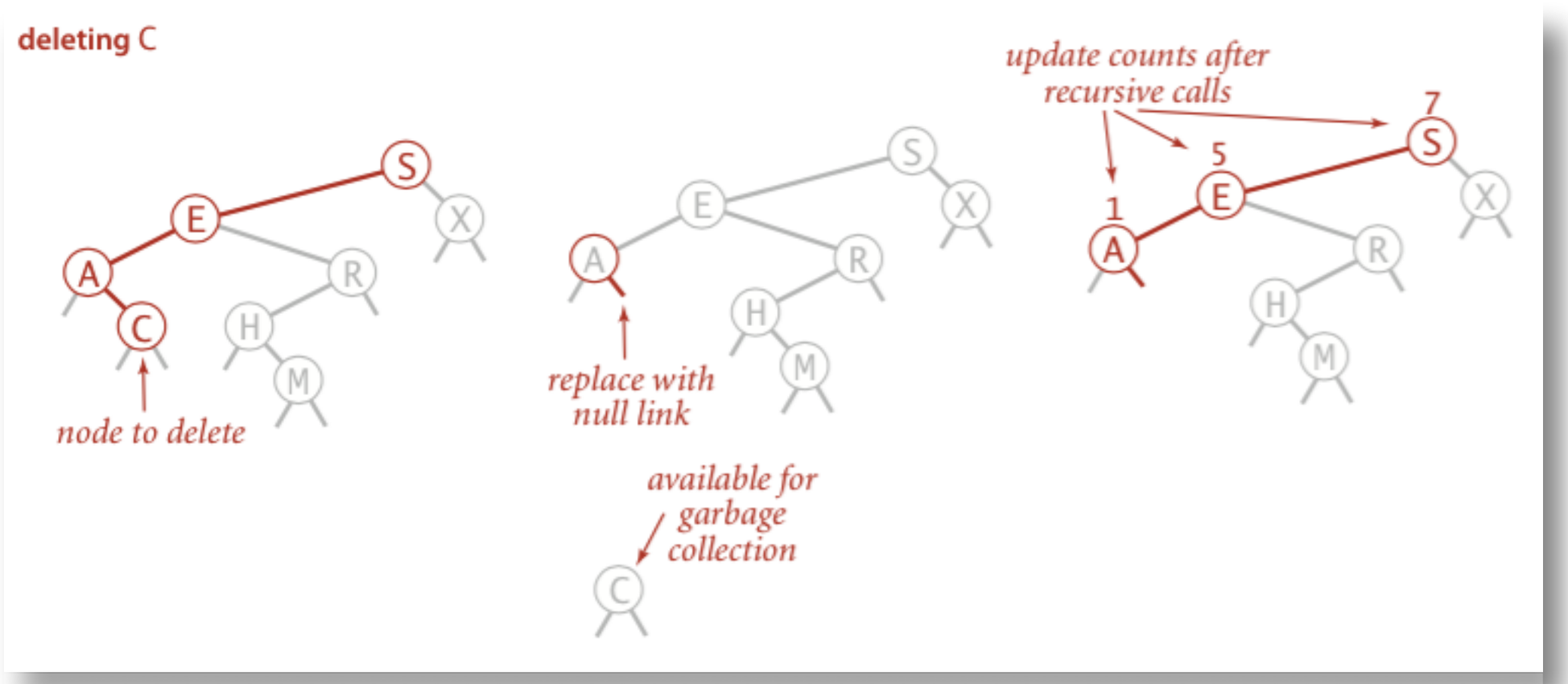
```
public void deleteMin()
{  root = deleteMin(root);  }

private Node deleteMin(Node x)
{
   if (x.left == null) return x.right;
   x.left = deleteMin(x.left);
   x.count = 1 + size(x.left) + size(x.right);
   return x;
}
```



Hibbard Deletion

To delete a node with key k: search for node t containing key k

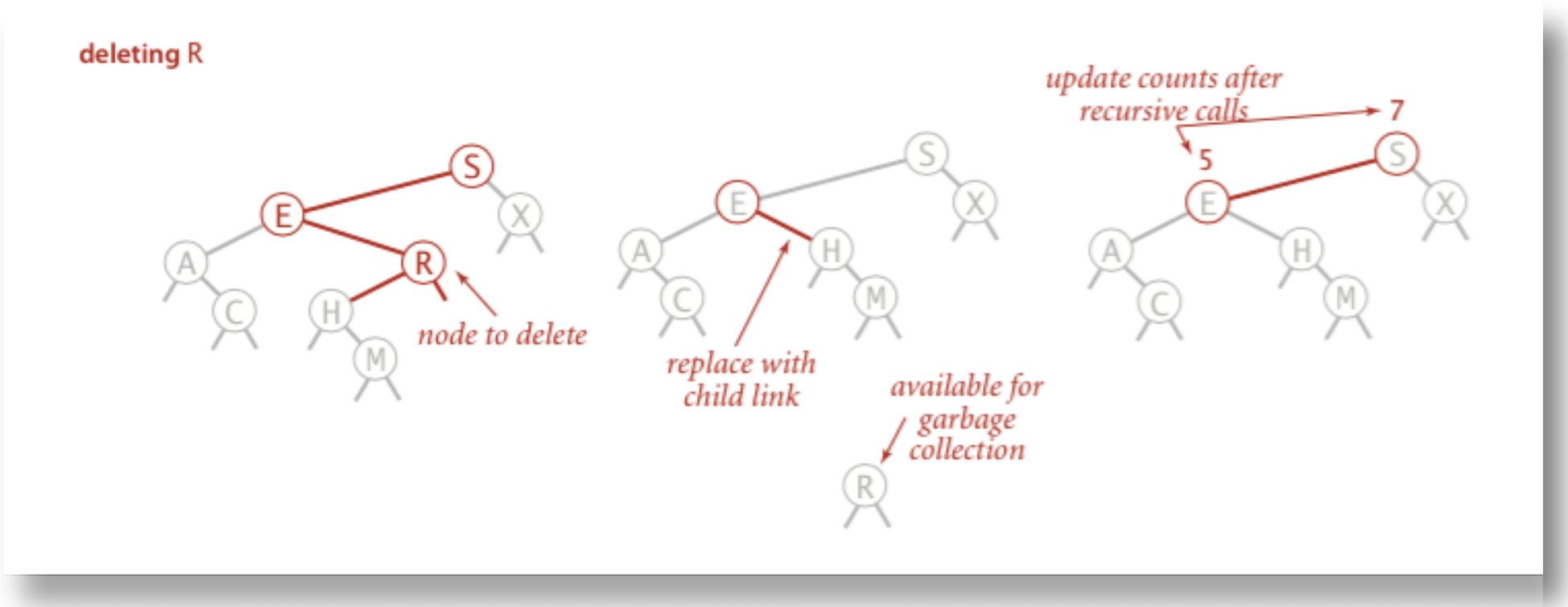
Case 0: [0 children] Delete t by setting parent link to null.



Hibbard Deletion

To delete a node with key k: search for node t containing key k

Case 1: [1 child] Delete t by replacing parent link.

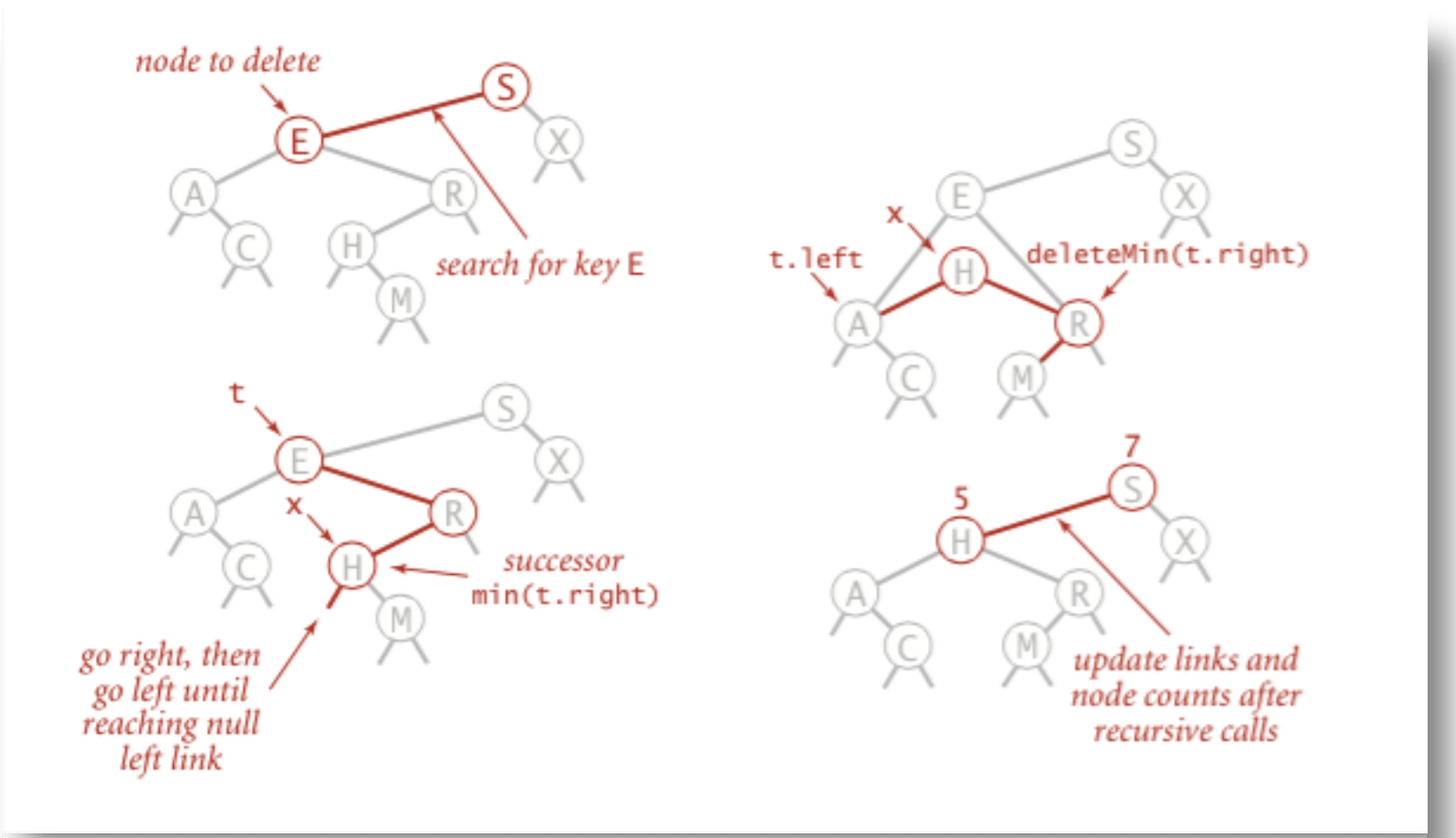


Hibbard Deletion

Find successor x of t.
Delete the minimum in t's right subtree.
Put x in t's spot.

~~x has no~~ left child
but ~~don't ga~~rbage collect
~~still~~ a BST

Case 2: [2 children].



Hibbard Deletion: Java Implementation

```
public void delete(Key key)
{  root = delete(root, key);  }

private Node delete(Node x, Key key) {
   if (x == null) return null;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) x.left  = delete(x.left,  key);
   else if (cmp > 0) x.right = delete(x.right, key);
   else {
      if (x.right == null) return x.left;
      if (x.left  == null) return x.right;

      Node t = x;
      x = min(t.right);
      x.right = deleteMin(t.right);
      x.left = t.left;
   }
   x.count = size(x.left) + size(x.right) + 1;
   return x;
}
```

search for key

no right child

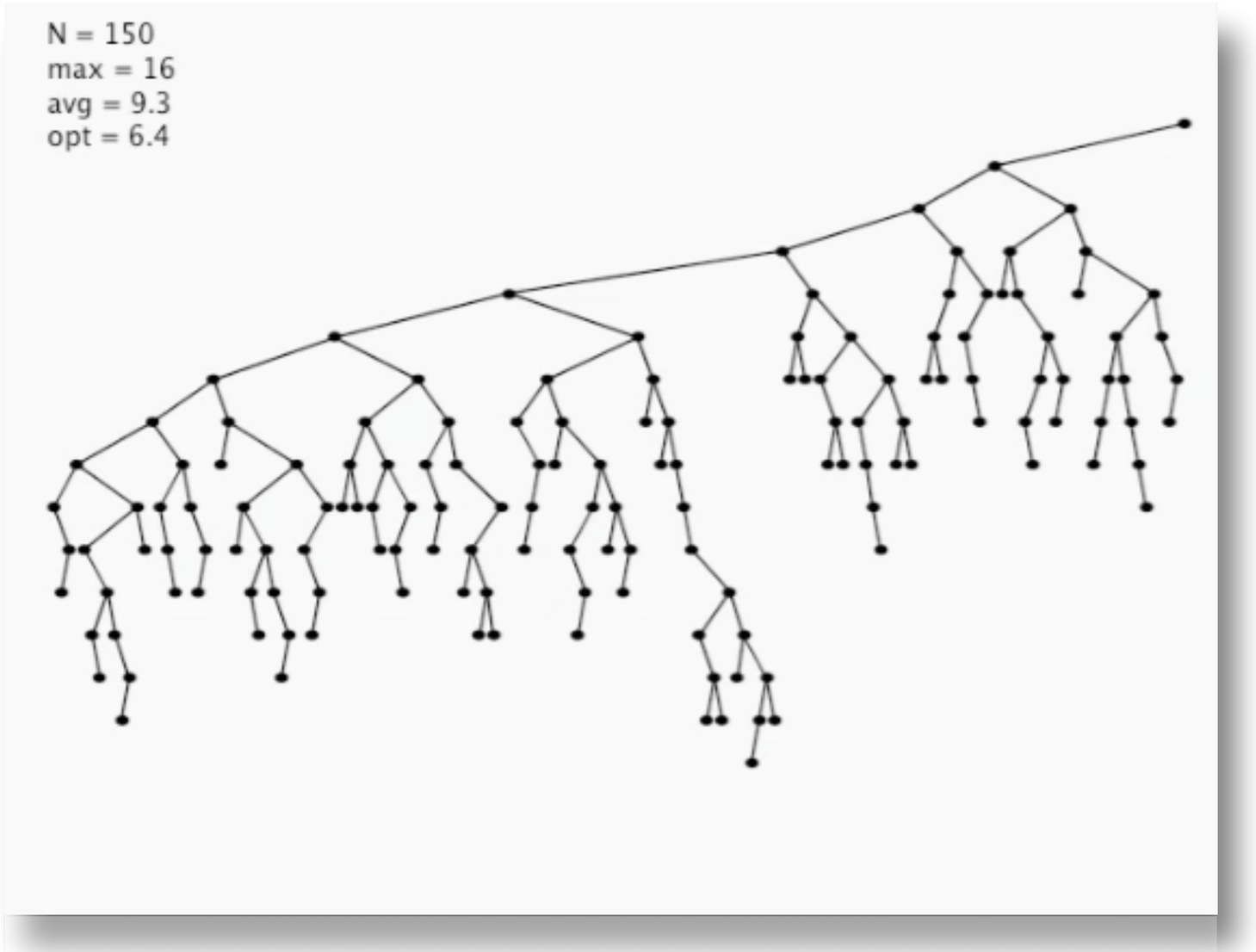
no left child

replace with successor

update subtree counts

Hibbard Deletion: Analysis

Unsatisfactory solution: Not symmetric.



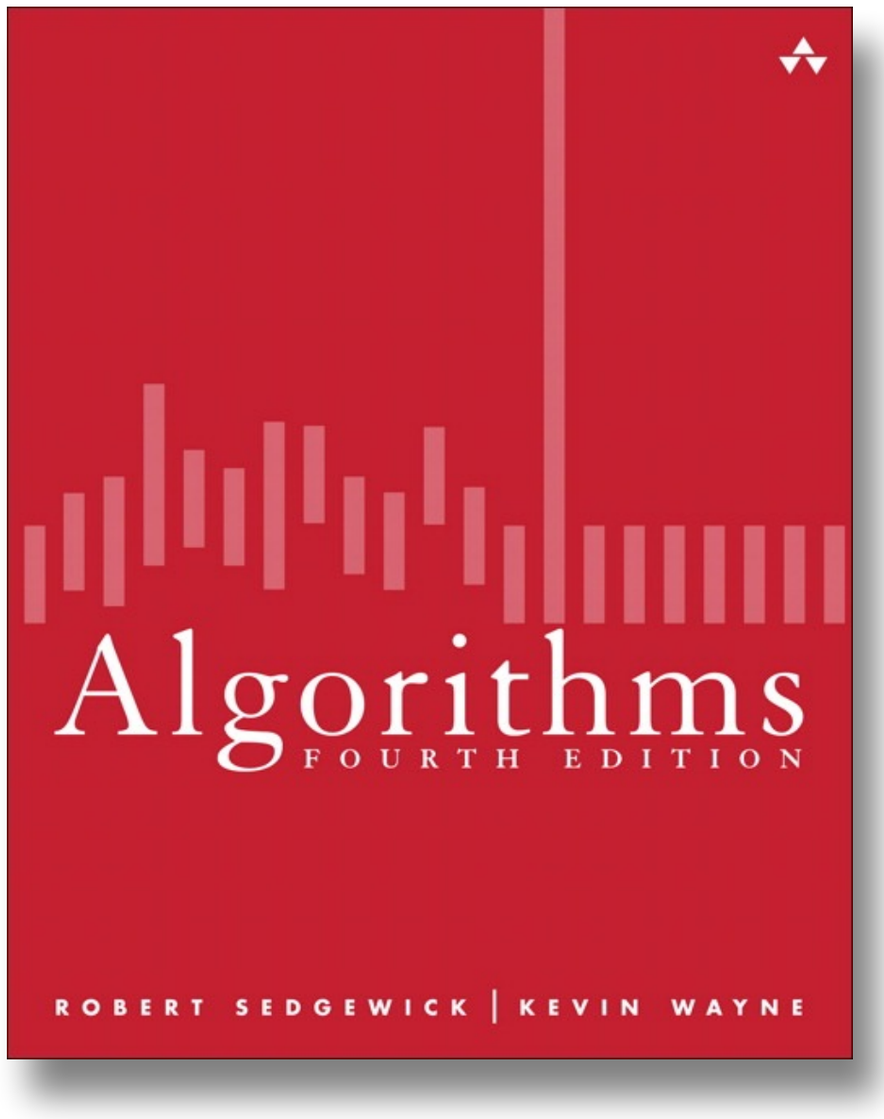
Surprising consequence: Trees not random (!) ⇒ √ N per op.

Longstanding open problem: Simple and efficient delete for BSTs.

Search Tree Implementations: Summary

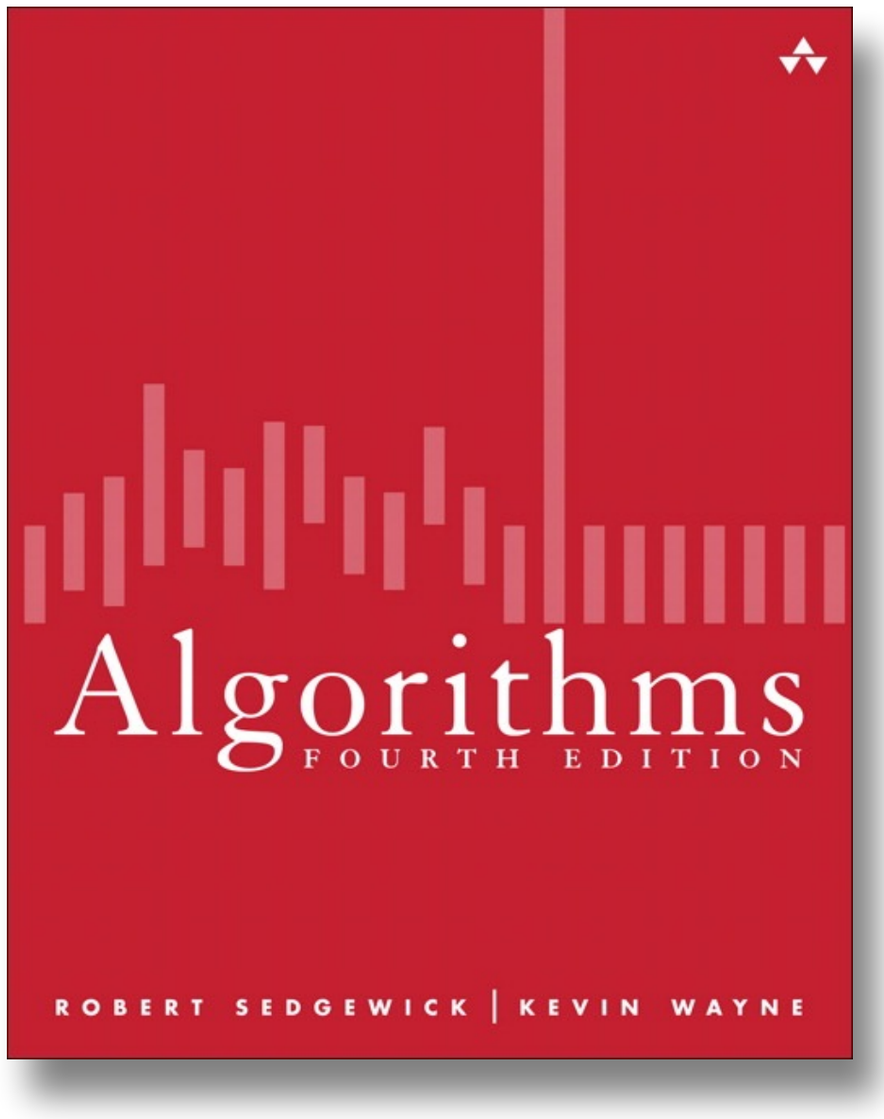
implementation	guarantee			average case			ordered ops?	operations on keys
	search	insert	delete	search hit	insert	delete		
sequential search (linked list)	N	N	N	$\frac{1}{2}\,N$	N	$\frac{1}{2}\,N$		`equals()`
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$\frac{1}{2}\,N$	$\frac{1}{2}\,N$	✔	`compareTo()`
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	$\sqrt{N}$	✔	`compareTo()`

other operations also become √N
if deletions allowed



Object Orientation Design

- Discovering new classes and methods
- Using CRC cards for class discovery
- Identify inheritance, aggregation, and dependency relationships between classes
- Describe class relationships using UML class diagrams

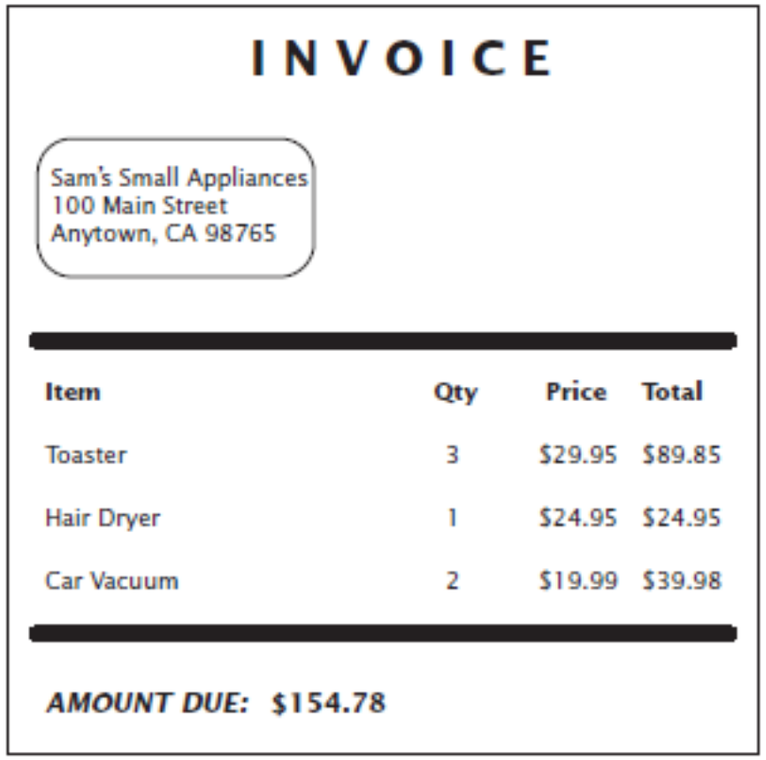


Object Orientation Design

- Discovering new classes and methods
- Using CRC cards for class discovery
- Identify inheritance, aggregation, and dependency relationships between classes
- Describe class relationships using UML class diagrams

Discovering Classes

- When designing a program, you work from a requirements specification
- The designer's task is to discover structures that make it possible to implement the requirements
- To discover classes, look for nouns in the problem description.
- Find methods by looking for verbs in the task description.



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

- Classes that come to mind:

- Invoice
- LineItem
- Customer

Discovering Classes

- Good idea to keep a list of candidate classes.
- Brainstorm: put all ideas for classes onto the list.
- Cross not useful ones later.
- Concepts from the problem domain are good candidates for classes.
- Not all classes can be discovered from the program requirements:
  - Most programs need specialized classes

INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

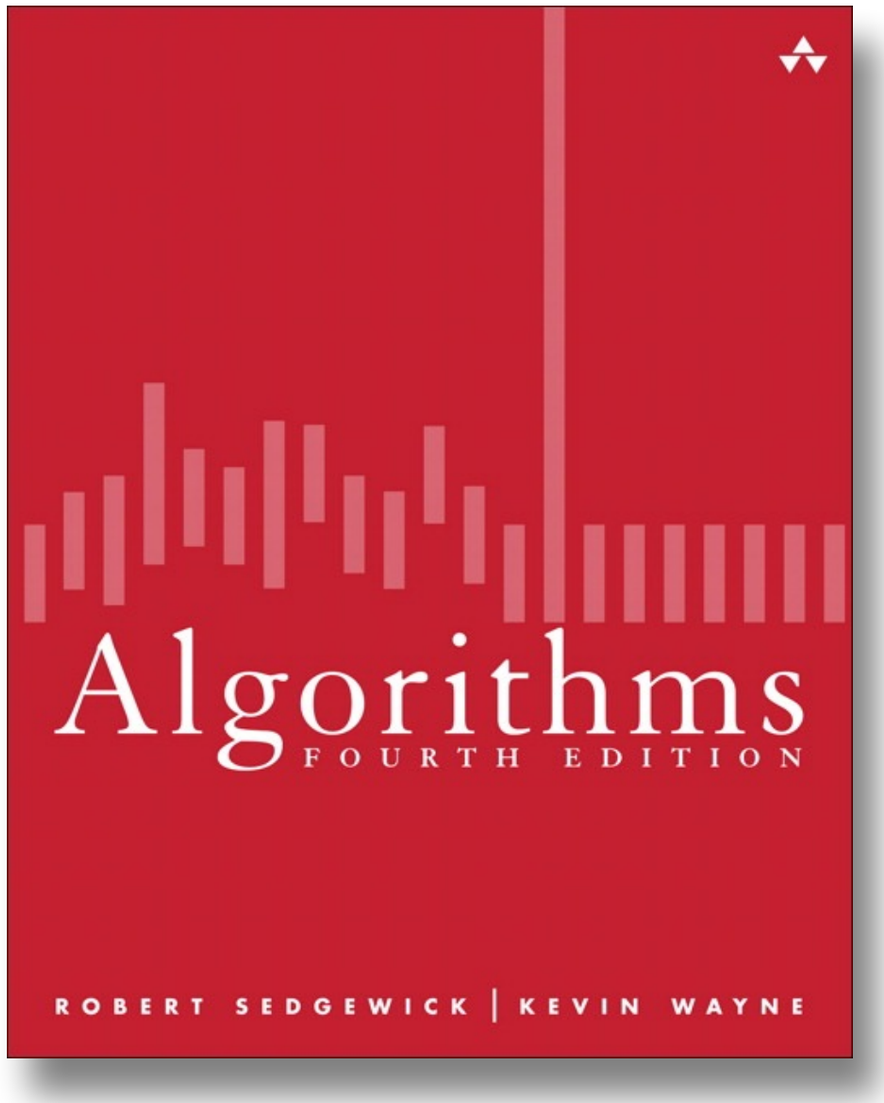
Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

- Classes that come to mind:

- Invoice
- LineItem
- Customer

Lecture Overview



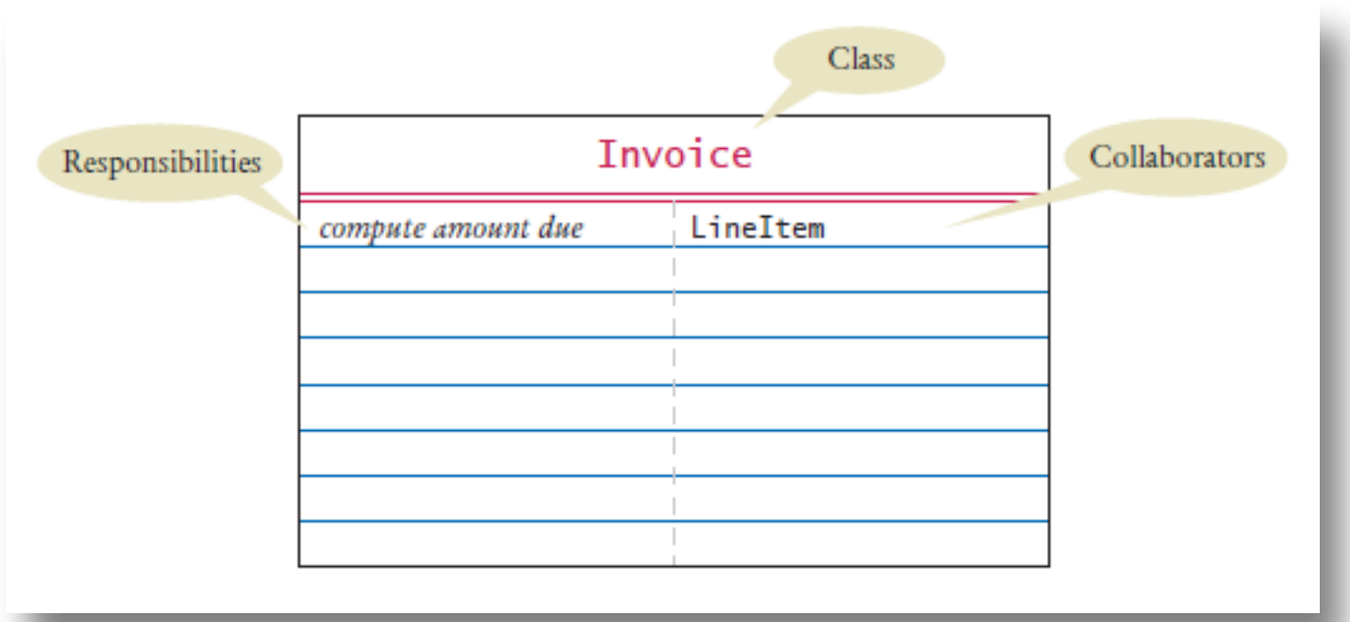
Object Orientation Design

- Discovering new classes and methods
- Using CRC cards for class discovery
- Identify inheritance, aggregation, and dependency relationships between classes
- Describe class relationships using UML class diagrams

The CRC Card Method

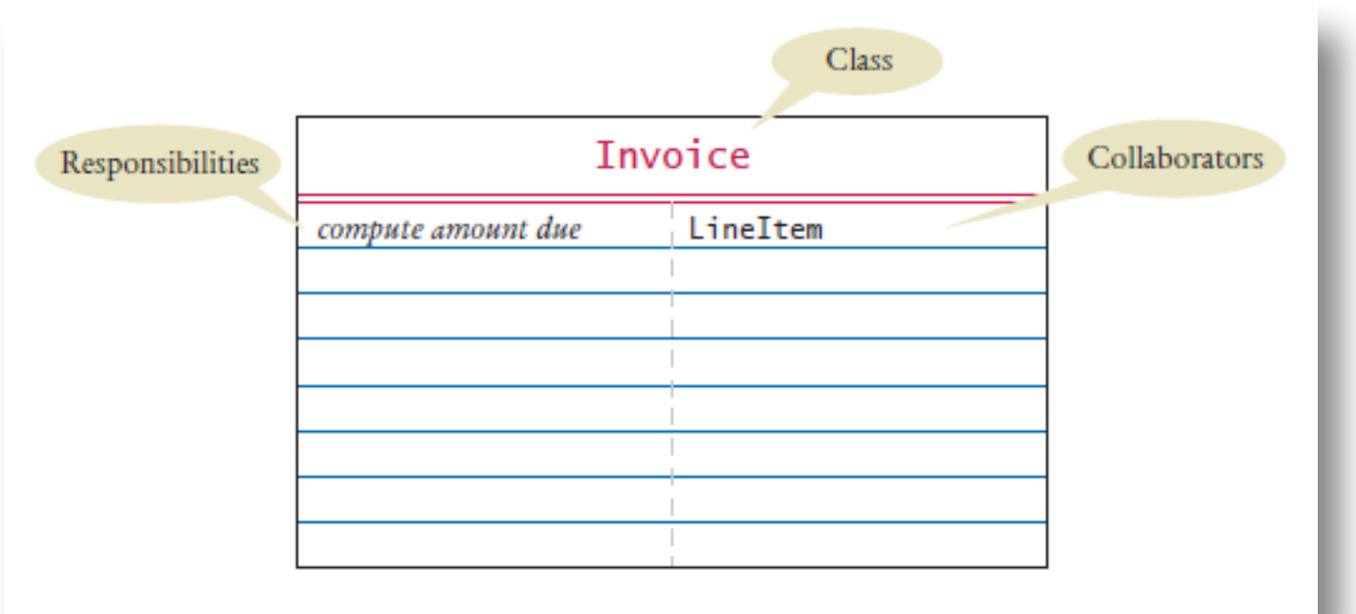
- After you have a set of classes
  - Define the behavior (methods) of each class
- Look for verbs in the task description
  - Match the verbs to the appropriate objects
- The invoice program needs to compute the amount due
  - Which class is responsible for this method?

    Invoice class

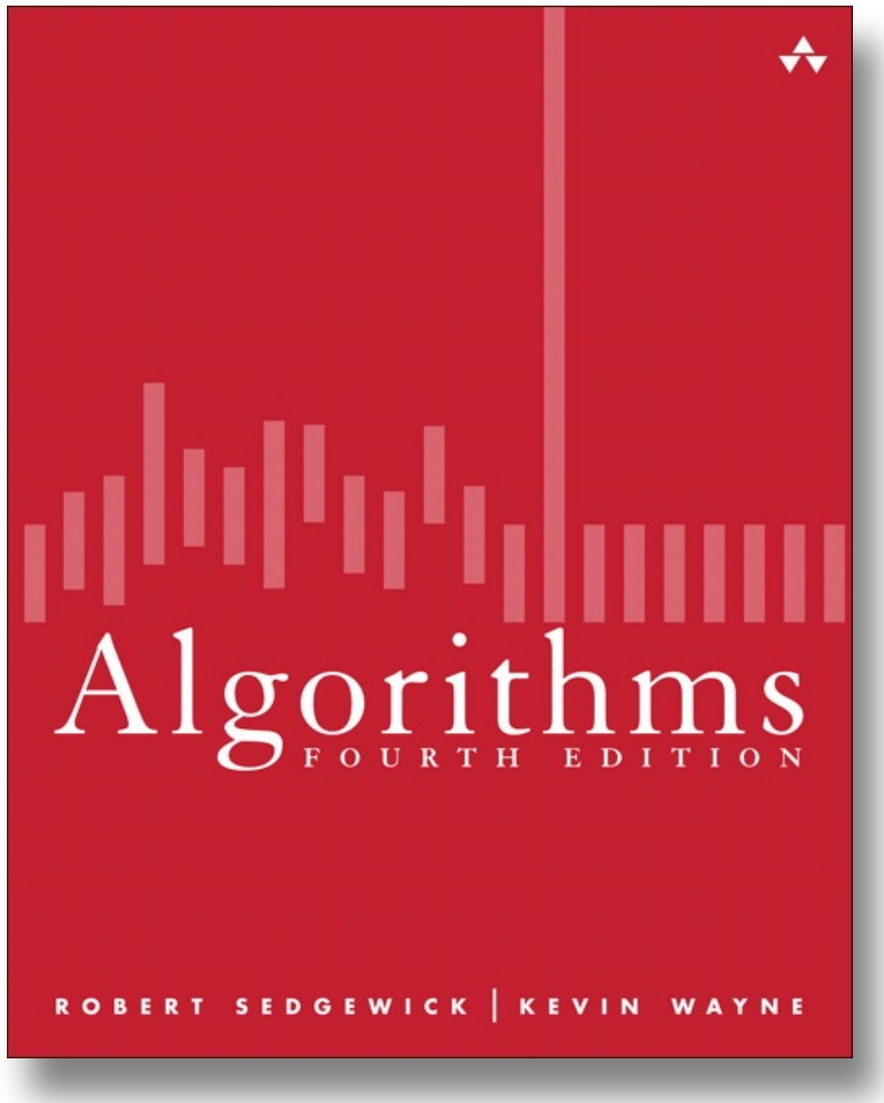


The CRC Card Method

- To find the class responsibilities, use the CRC card method.
- A CRC card describes a class, its responsibilities, and its collaborating classes.
  - CRC - stands for “classes”, “responsibilities”, “collaborators”
- Use an index card for each class.
- Pick the class that should be responsible for each method (verb).
- Write the responsibility onto the class card.
- Indicate what other classes are needed to fulfill responsibility (collaborators).



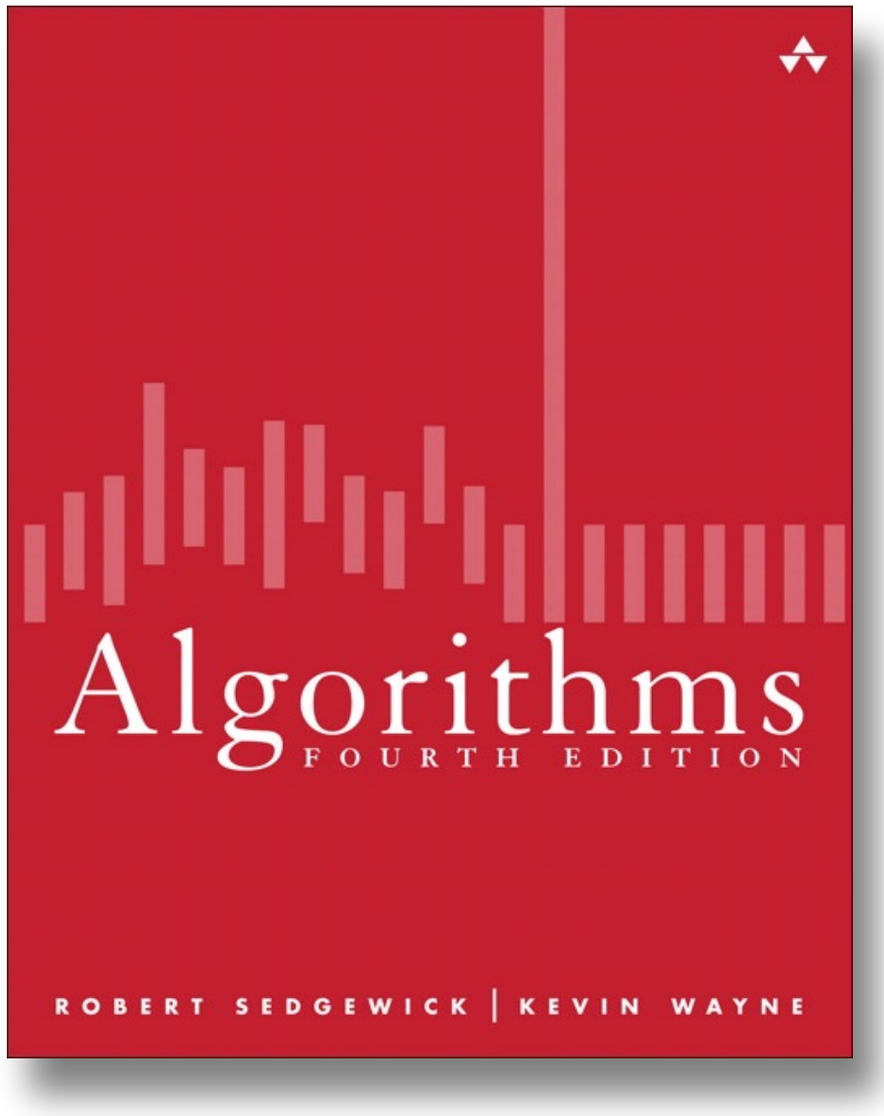
Lecture Overview



Object Orientation Design

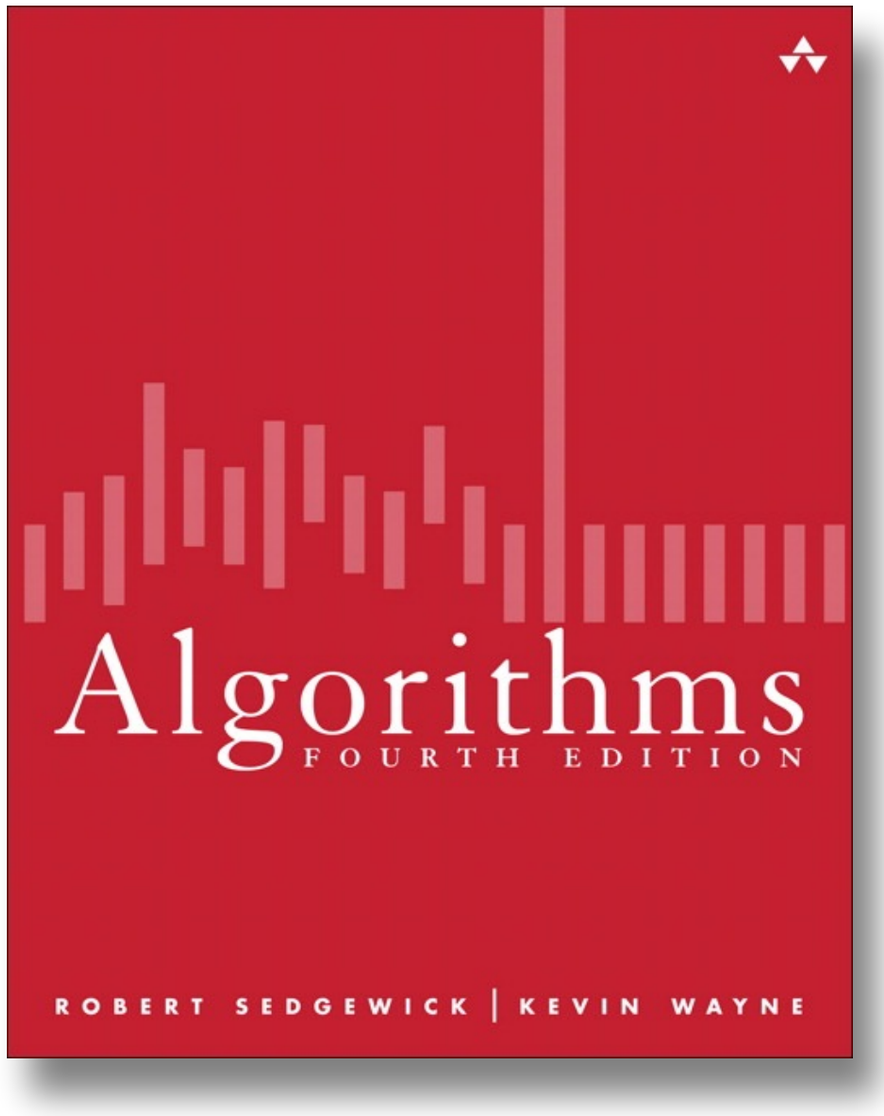
- Discovering new classes and methods
- Using CRC cards for class discovery
- Identify inheritance, aggregation, and dependency relationships between classes
- Describe class relationships using UML class diagrams

Lecture Overview



Relationships Between Classes

- Dependency
- Aggregation
- Inheritance

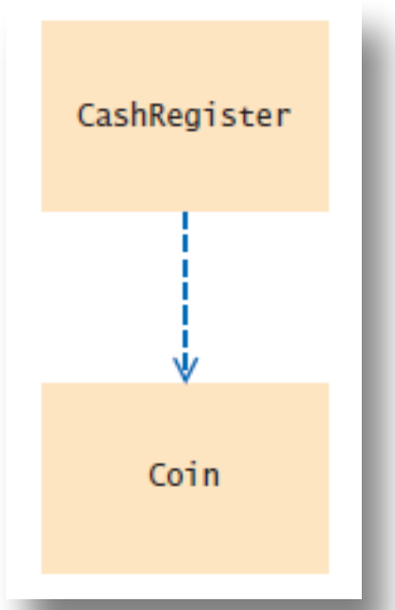


Relationships Between Classes

- Dependency
- Aggregation
- Inheritance

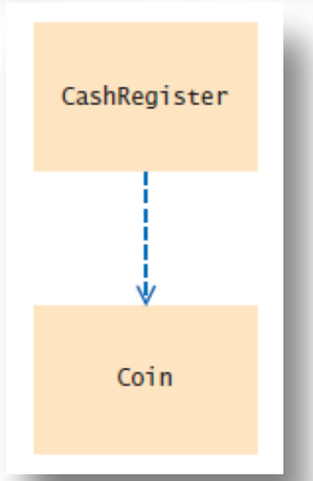
Dependency

- A class depends on another class if it uses objects of that class.
  - The “knows about” relationship.
- Example: CashRegister depends on Coin

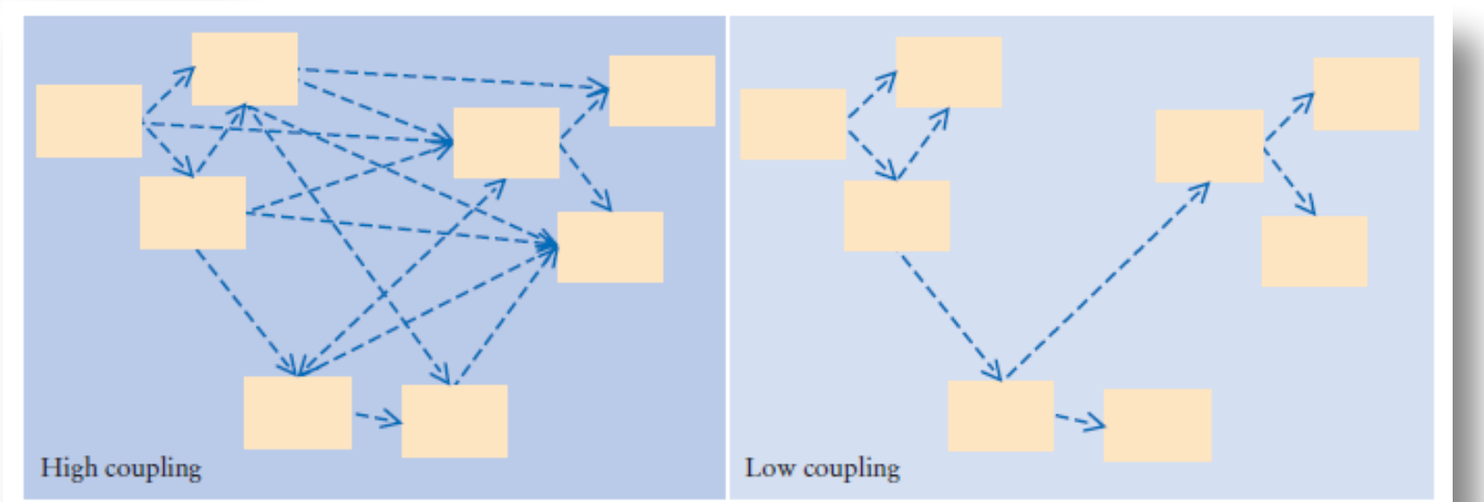


Dependency

- A class depends on another class if it uses objects of that class.
  - The “knows about” relationship.
- Example: CashRegister depends on Coin

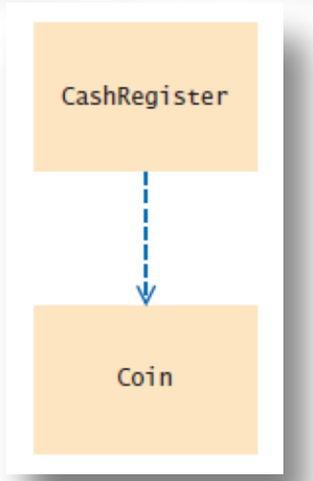


- It is a good practice to minimize the coupling (i.e., dependency) between classes.

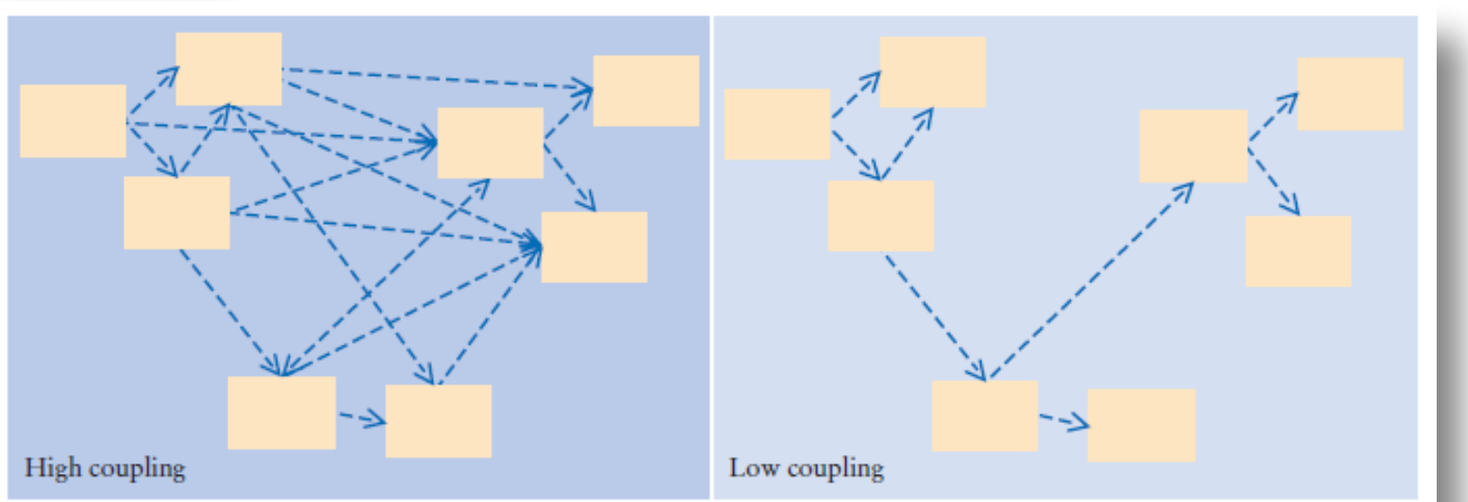


Dependency

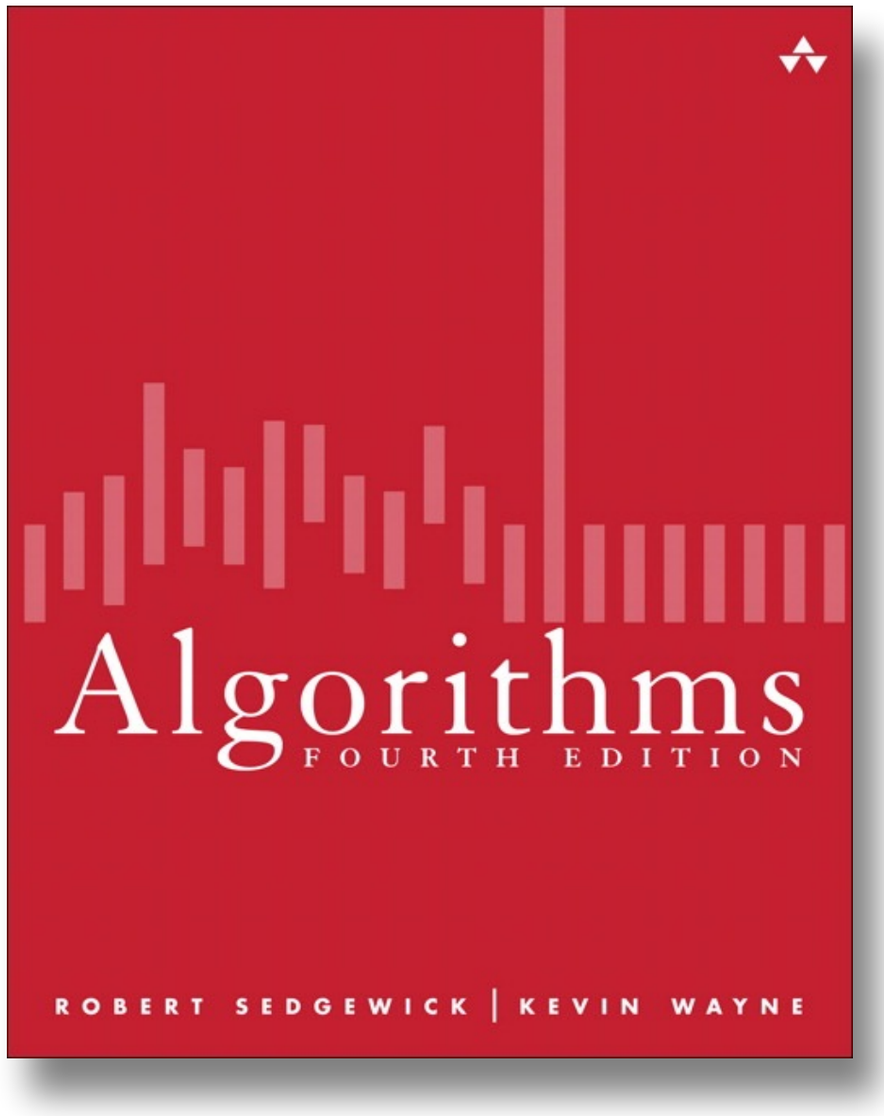
- A class depends on another class if it uses objects of that class.
  - The “knows about” relationship.
- Example: CashRegister depends on Coin



- It is a good practice to minimize the coupling (i.e., dependency) between classes.



- When a class changes, coupled classes may also need updating.

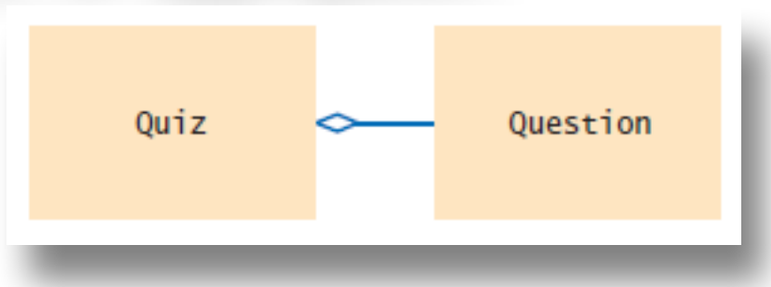


Relationships Between Classes

- Dependency
- Aggregation
- Inheritance

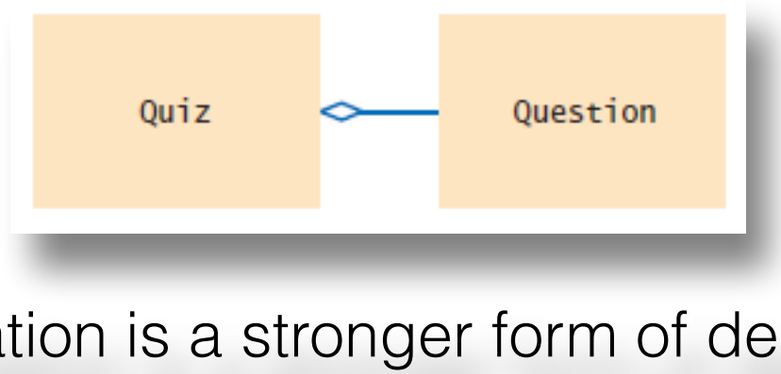
Aggregation

- A class aggregates another if its objects contain objects of the other class.
  - Has-a relationship
- Example: a Quiz class aggregates a Question class.
- The UML for aggregation:



Aggregation

- A class aggregates another if its objects contain objects of the other class.
  - Has-a relationship
- Example: a Quiz class aggregates a Question class.
- The UML for aggregation:

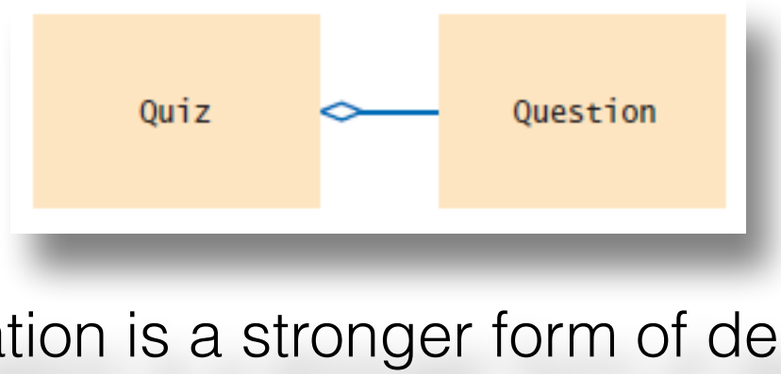


- Aggregation is a stronger form of dependency.
- Use aggregation to remember another object between method calls.
- Use an instance variable

```
public class Quiz {
   private ArrayList<Question> questions;
   . . .
}
```

Aggregation

- A class aggregates another if its objects contain objects of the other class.
  - Has-a relationship
- Example: a Quiz class aggregates a Question class.
- The UML for aggregation:

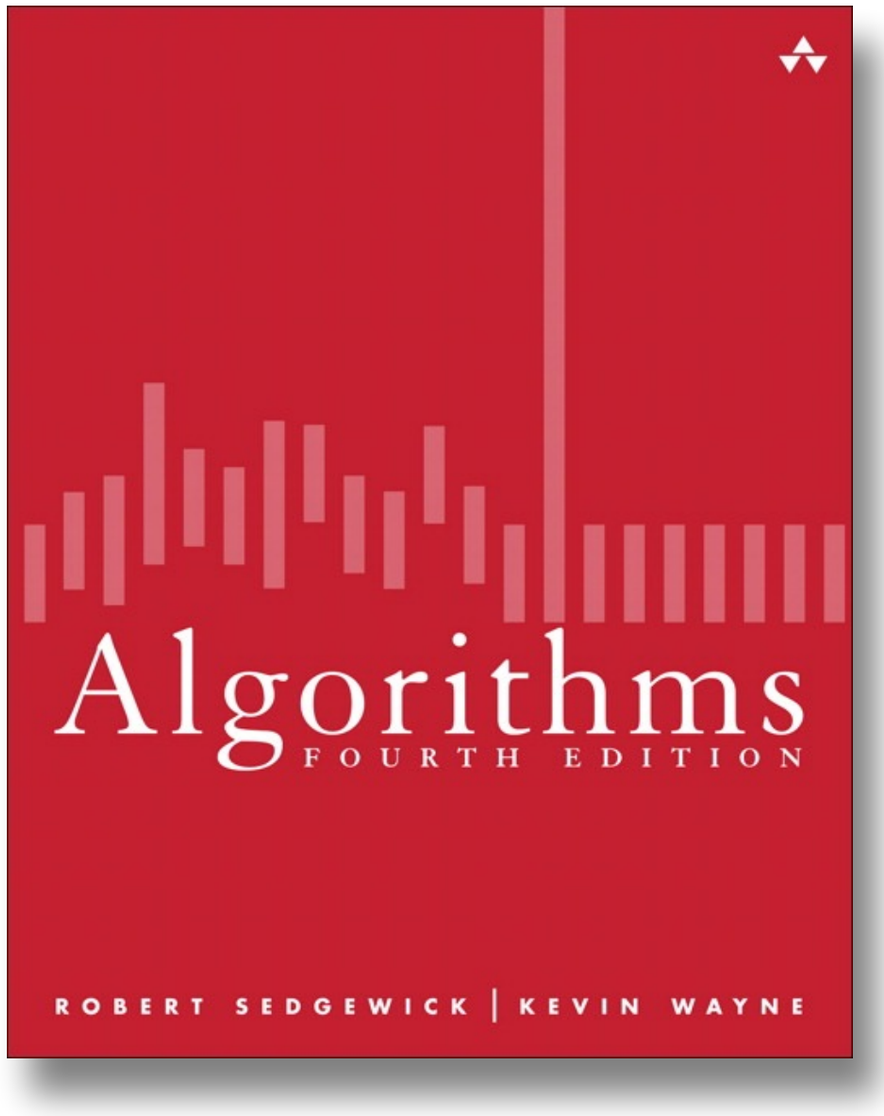


- Aggregation is a stronger form of dependency.
- Use aggregation to remember another object between method calls.
- Use an instance variable

```
public class Quiz {
   private ArrayList<Question> questions;
   . . .
}
```

- A class may use the Scanner class without ever declaring an instance variable of class Scanner.
  - This is dependency NOT aggregation

Lecture Overview

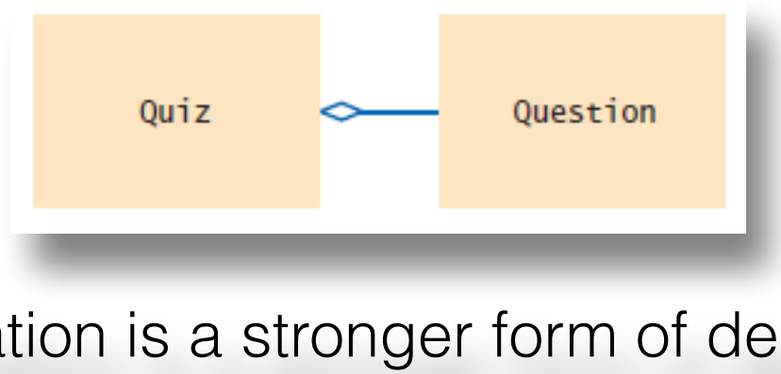


Relationships Between Classes

- Dependency
- Aggregation
- Inheritance

Inheritance

- A class aggregates another if its objects contain objects of the other class.
  - Has-a relationship
- Example: a Quiz class aggregates a Question class.
- The UML for aggregation:



- Aggregation is a stronger form of dependency.
- Use aggregation to remember another object between method calls.
- Use an instance variable

```
public class Quiz {
   private ArrayList<Question> questions;
   . . .
}
```

- A class may use the Scanner class without ever declaring an instance variable of class Scanner.
  - This is dependency NOT aggregation

Inheritance

- Inheritance is a relationship between a more general class (the superclass) and a more
- specialized class (the subclass).
  - The “is-a” relationship.
  - Example: Every truck is a vehicle.
- Inheritance is sometimes inappropriately used when the has-a relationship would be more appropriate.
- Should the class Tire be a subclass of a class Circle?

```
public class Quiz {
   private ArrayList<Question> questions;
   . . .
}
```

Inheritance

- Inheritance is a relationship between a more general class (the superclass) and a more
- specialized class (the subclass).
  - The “is-a” relationship.
  - Example: Every truck is a vehicle.
- Inheritance is sometimes inappropriately used when the has-a relationship would be more appropriate.

- Should the class Tire be a subclass of a class Circle? **No**
  - A tire has a circle as its boundary
  - Use aggregation

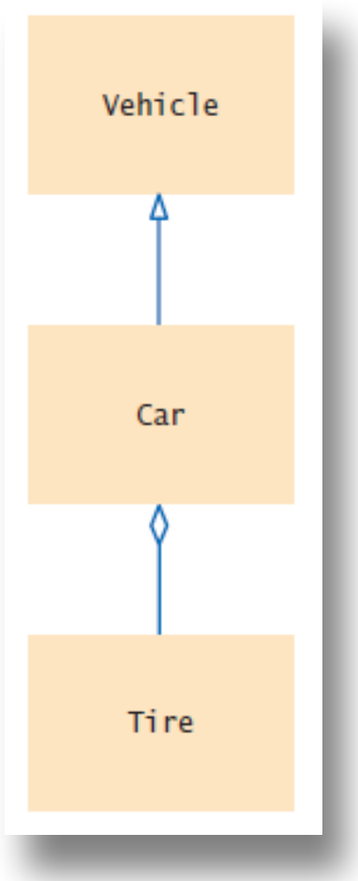
```
public class Tire {
   private String rating;
   private Circle boundary;
   . . .
}
```

Inheritance

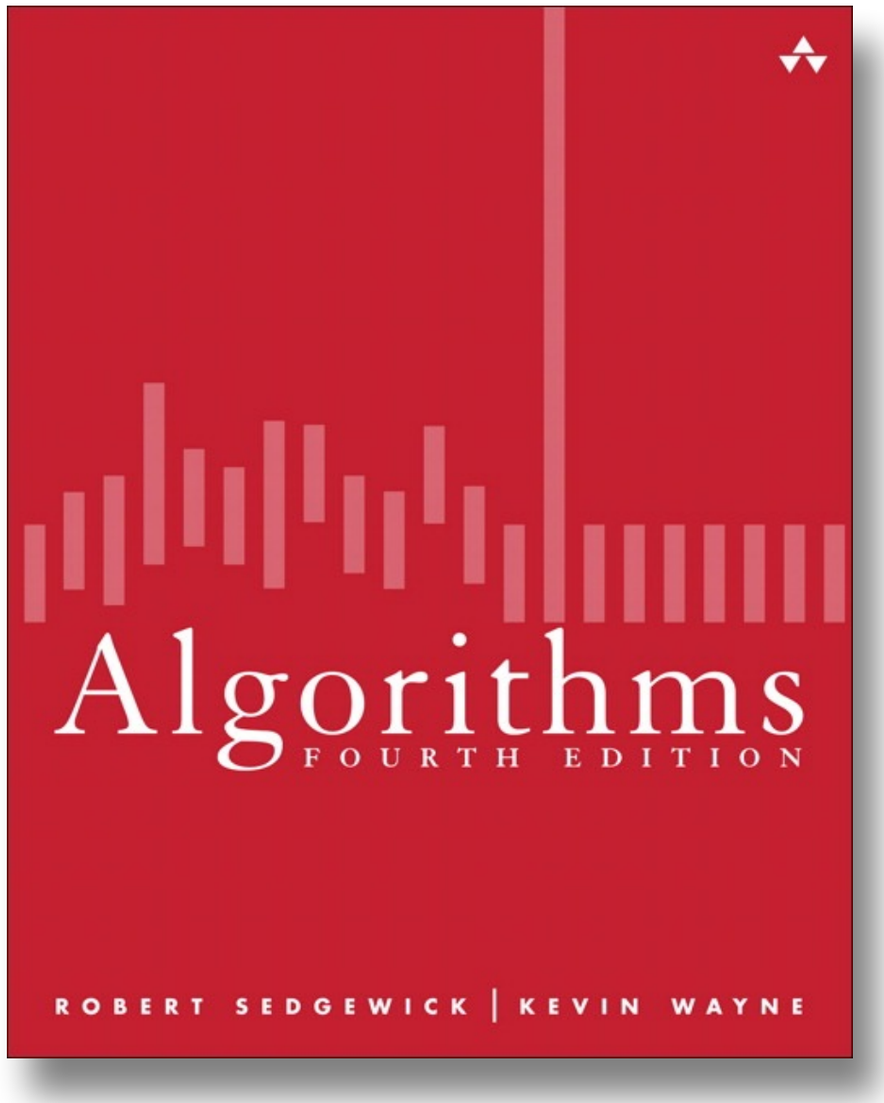
- Every car is a vehicle. (Inheritance)
- Every car has a tire (or four). (Aggregation)

```
class Car extends Vehicle {
   private Tire[] tires;
   . . .
}
```

- Aggregation denotes that objects of one class contain references to objects of another class.



Lecture Overview



Object Orientation Design

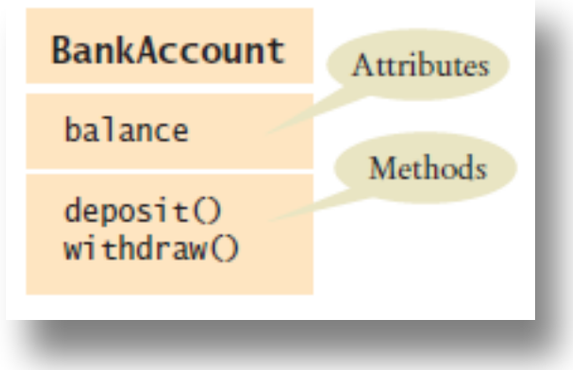
- Discovering new classes and methods
- Using CRC cards for class discovery
- Identify inheritance, aggregation, and dependency relationships between classes
- Describe class relationships using UML class diagrams

UML Relationship Symbols

Relationship	Symbol	Line Style	Arrow Tip
Inheritance		Solid	Triangle
Interface Implementation		Dotted	Triangle
Aggregation		Solid	Diamond
Dependency		Dotted	Open

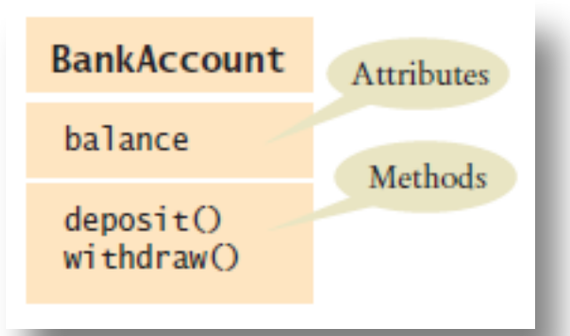
UML Relationship Symbols

- Attributes and Methods in UML Diagrams



UML Relationship Symbols

- Attributes and Methods in UML Diagrams



- Multiplicities
  - any number (zero or more): *
  - one or more: 1..*
  - zero or one: 0..1
  - exactly one: 1

UML Relationship Symbols

- Aggregation and Association, and Composition

  - Association: More general relationship between classes.
  - Use early in the design phase.
  - A class is associated with another if you can navigate from objects of one class to objects of the other.

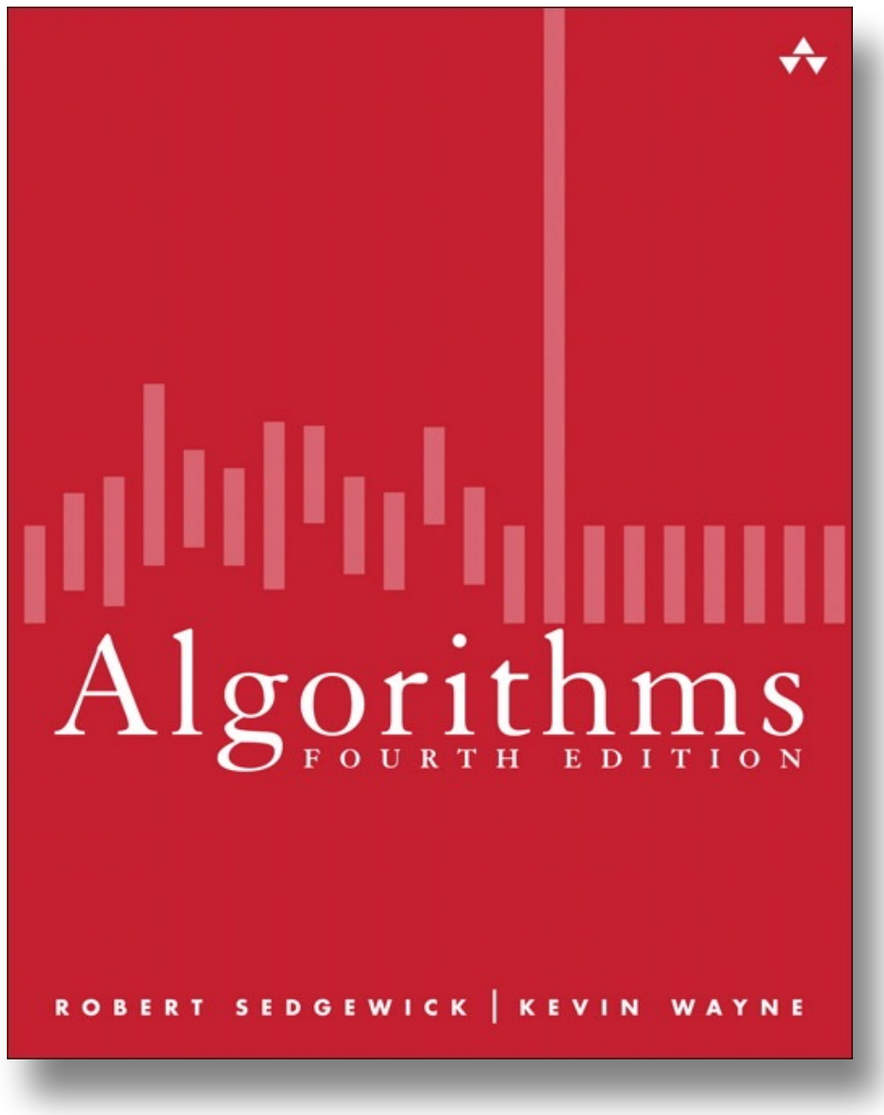
  - Given a Bank object, you can navigate to Customer objects.

An Association Relationship

- Composition: one of the classes can not exist without the other.

A Composition Relationship

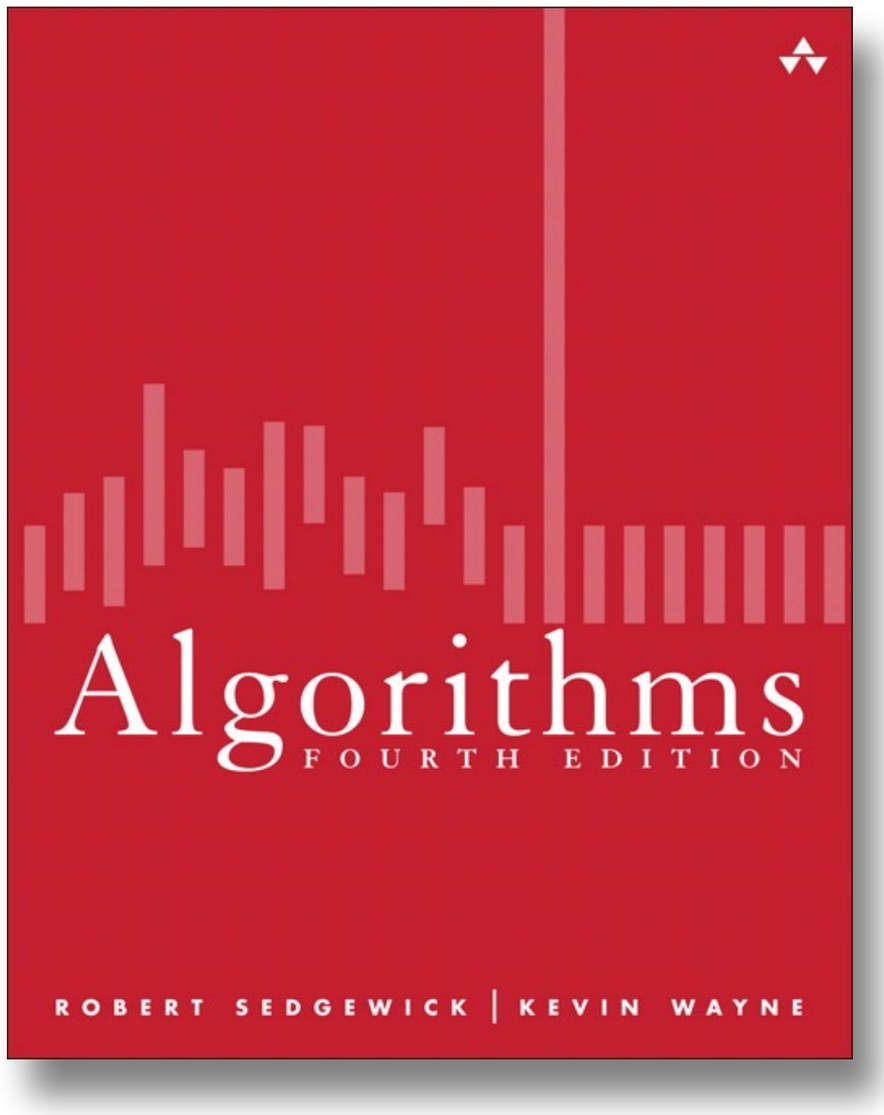
Lecture Overview



Object Orientation Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

Lecture Overview



Object Orientation Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

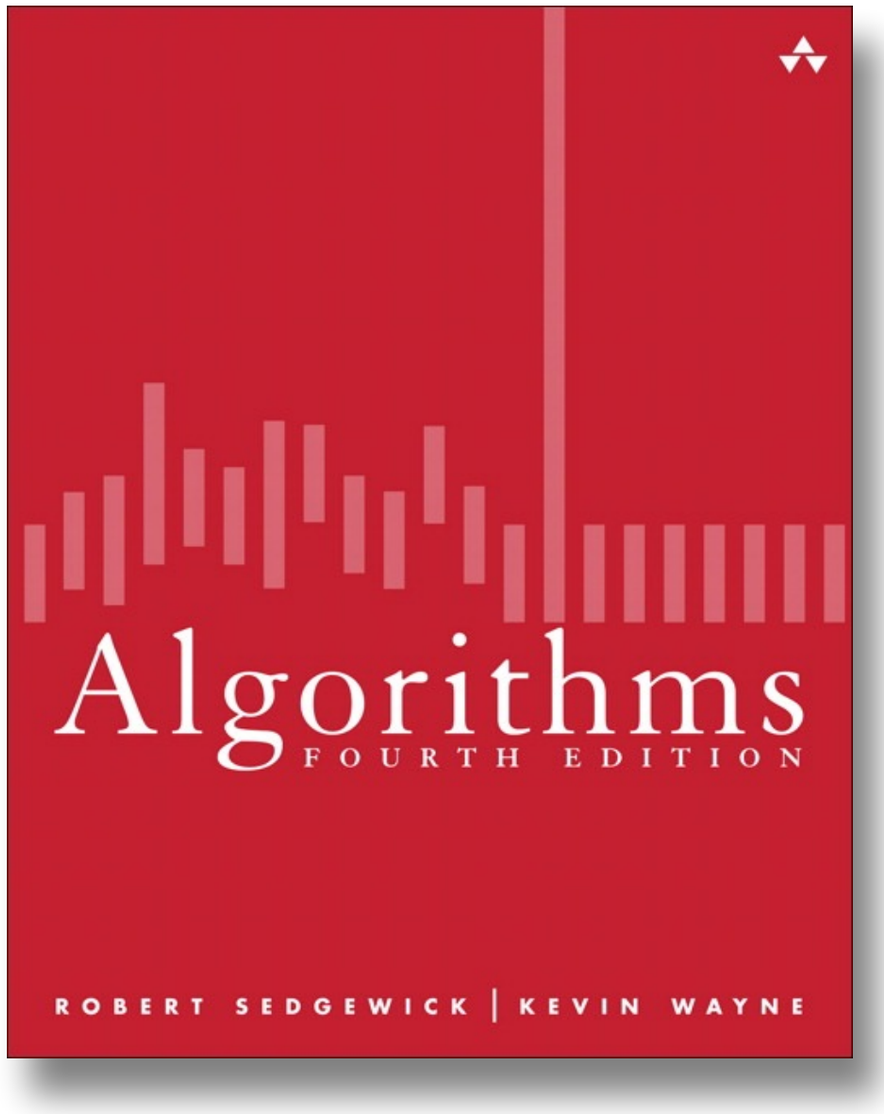
Requirements

- Start the development process by gathering and documenting program requirements.
- Task: Print out an invoice
- Invoice: Describes the charges for a set of products in certain quantities.
- Omit complexities
  - Dates, taxes, and invoice and customer numbers
- Print invoice
  - Billing address, all line items, amount due
- Line item
  - Description, unit price, quantity ordered, total price
- For simplicity, do not provide a user interface.
- Test program: Adds line items to the invoice and then prints it.

Requirements

- Printing an Invoice.
- An invoice lists the charges for each item and the amount due.

```
                  I N V O I C E

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Description                    Price Qty Total
Toaster                        29.95   3  89.85
Hair dryer                     24.95   1  24.95
Car vacuum                     19.99   2  39.98

AMOUNT DUE: $154.78
```

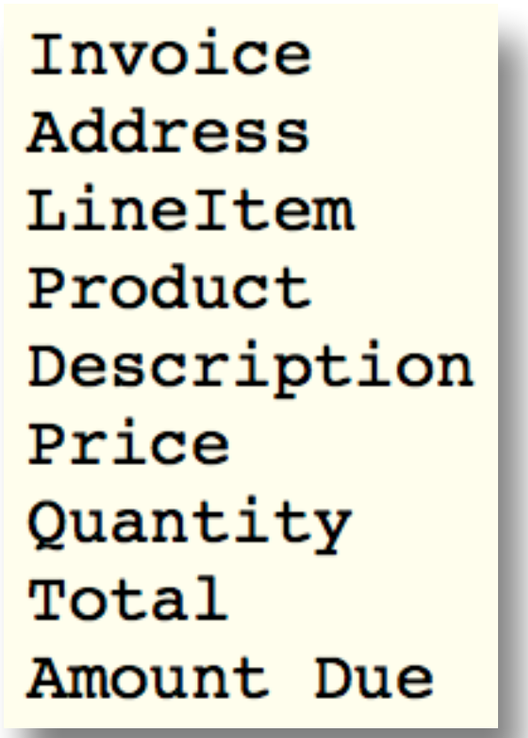


Object Orientation Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

CRC cards

- Use CRC cards to find classes, responsibilities, and collaborators.
- Discover classes
- Nouns are possible classes:



CRC cards

- Analyze classes:

```
Invoice
Address
LineItem     // Records the product and the quantity
Product
Description // Field of the Product class
Price       // Field of the Product class
Quantity    // Not an attribute of a Product
Total       // Computed – not stored anywhere
Amount Due  // Computed – not stored anywhere
```

- Classes after a process of elimination:

```
Invoice
Address
LineItem
Product
```

CRC cards

- Invoice and Address must be able to format themselves:

CRC cards

- Invoice and Address must be able to format themselves:

Invoice	Address
format the invoice	*format the address*

- Add collaborators to Invoice card:

Invoice	
format the invoice	Address
	LineItem

CRC cards

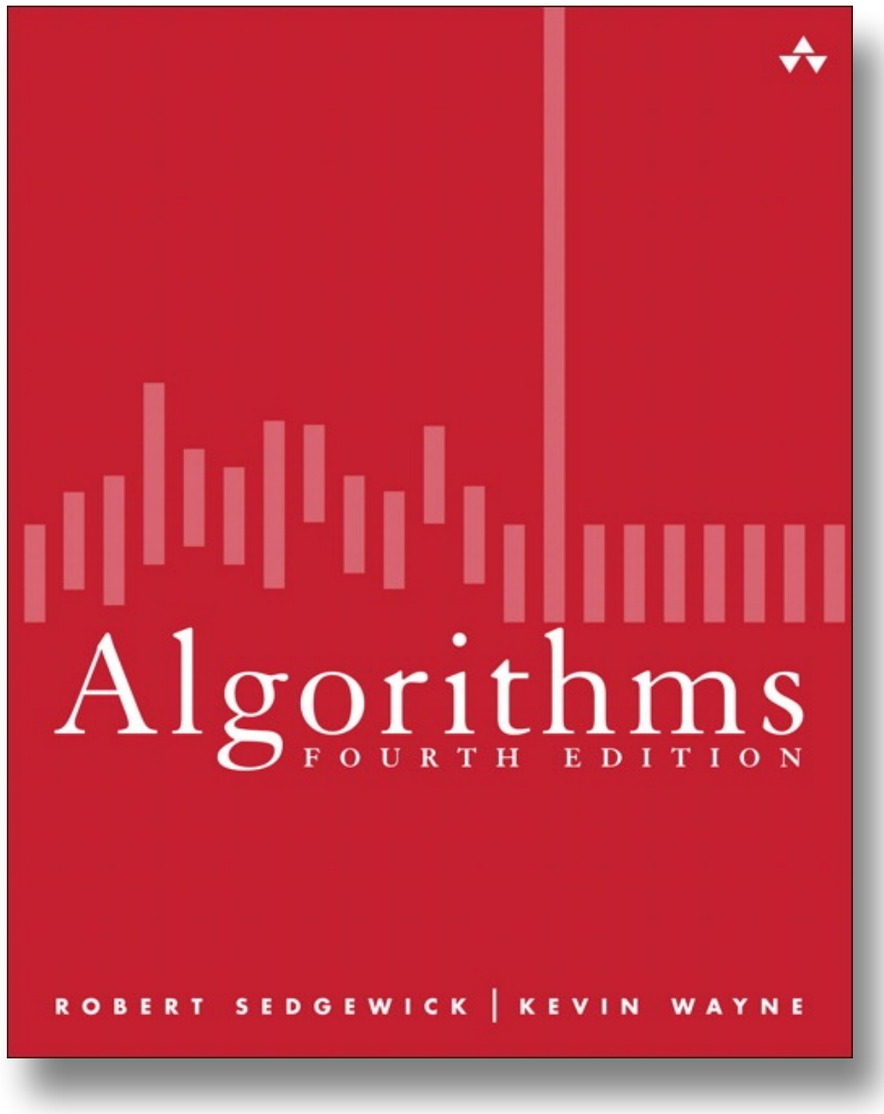
- Product and LineItem CRC cards:

Product	
get description	
get unit price	

LineItem	
format the item	Product
get tota l price	

- Invoice must be populated with products and quantities:

Invoice	
format the invoice	Address
add a product and quantity	LineItem
	Product

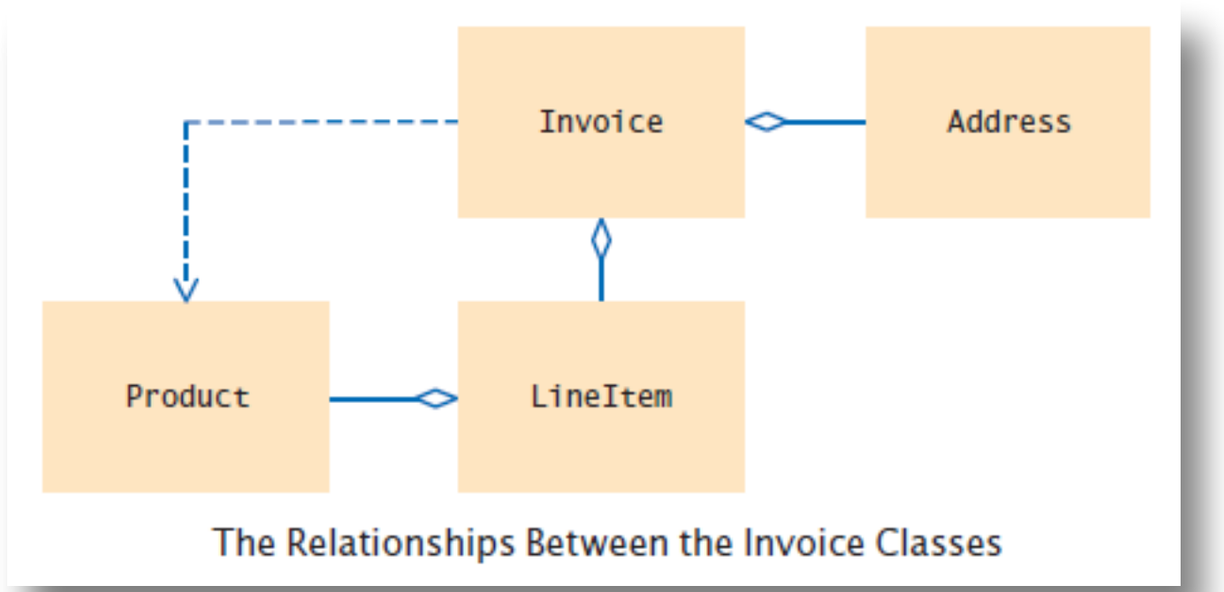


Object Orientation Process

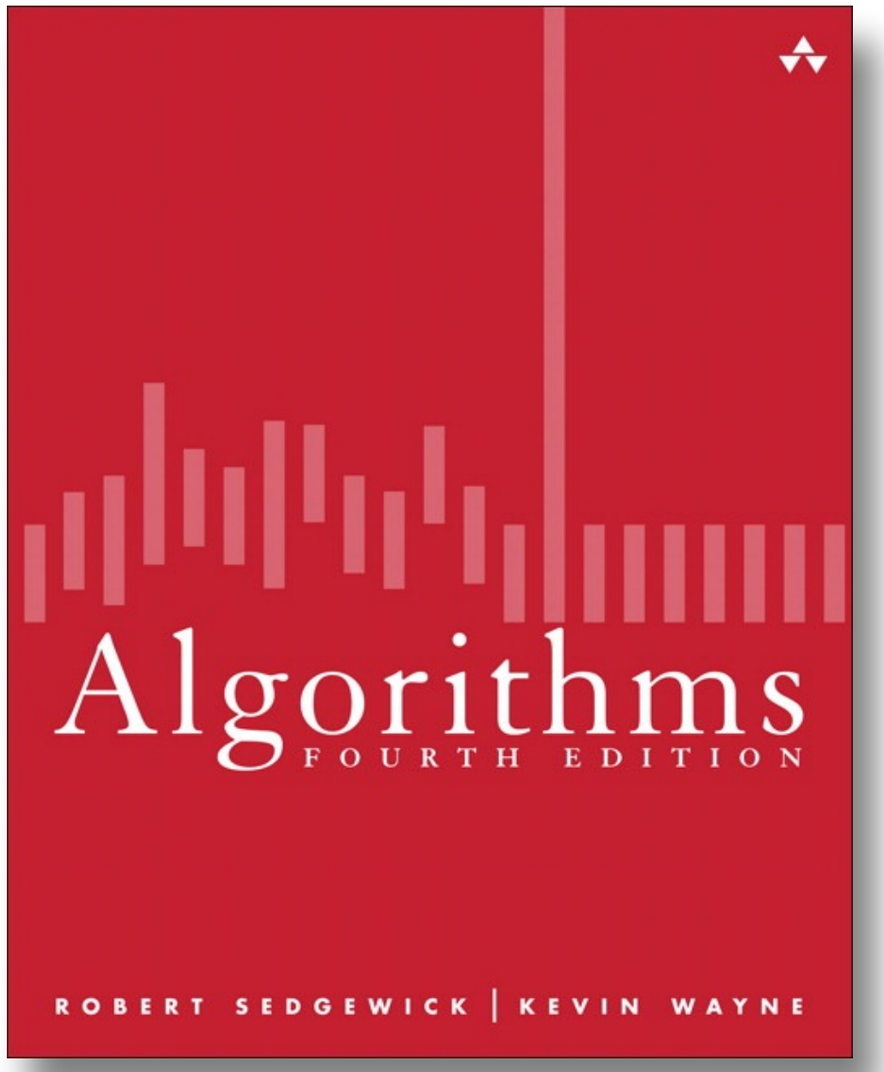
- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

UML Diagrams

- Product and LineItem CRC cards:



The Relationships Between the Invoice Classes



Object Orientation Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

Documentation

- Use javadoc comments (with the method bodies left blank) to record the behavior of the classes.
- Write a Java source file for each class:
  - Write the method comments for those methods that you have discovered,
  - Leave the body of the methods blank
- Run javadoc to obtain formatted version of documentation in HTML format.
- Advantages:
  - Share HTML documentation with other team members
  - Format is immediately useful: Java source files
  - Supply the comments of the key methods

Documentation

```
/**
   Describes an invoice for a set of purchased products.
*/
public class Invoice {
   /**
      Adds a charge for a product to this invoice.
      @param aProduct the product that the customer ordered
      @param quantity the quantity of the product
   */
   public void add(Product aProduct, int quantity) {
   }
   /**
      Formats the invoice.
      @return the formatted invoice
   */
   public String format() {
   }
}
```

Documentation

```
/**
   Describes a quantity of an article to purchase and its price.
*/
public class LineItem {
   /**
      Computes the total cost of this line item.
      @return the total price
   */
   public double getTotalPrice() {
   }
   /**
      Formats this item.
      @return a formatted string of this line item
   */
   public String format() {
   }
}
```

Documentation

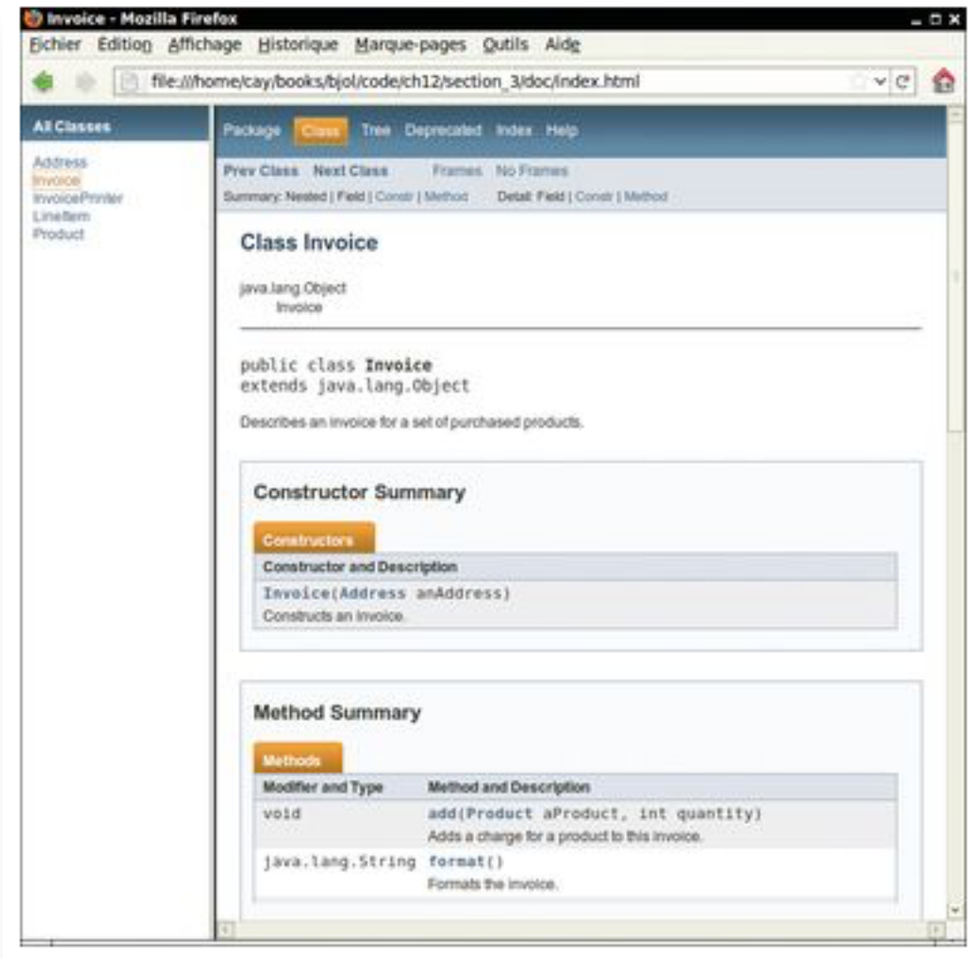
```
/**
   Describes a product with a description and a price.
*/
public class Product {
   /**
      Gets the product description.
      @return the description
   */
   public String getDescription() {
   }
   /**
      Gets the product price.
      @return the unit price
   */
   public double getPrice() {
   }
}
```

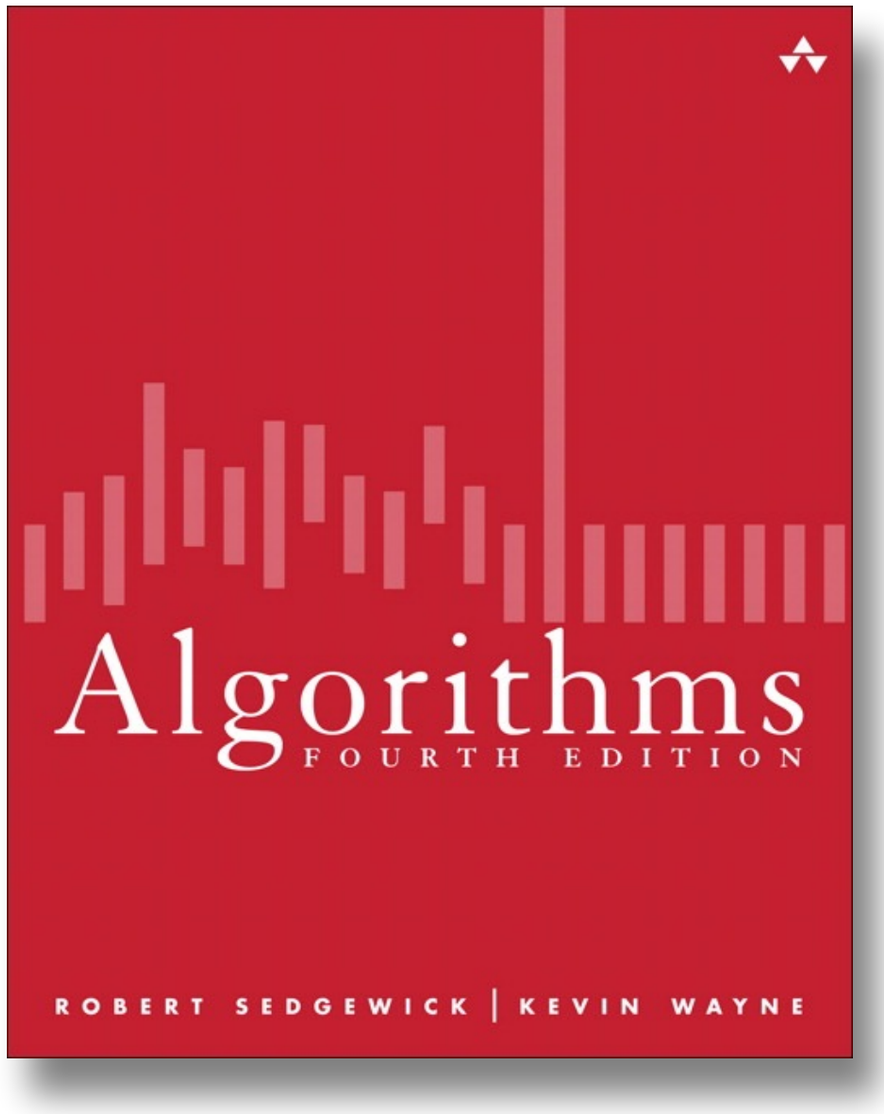
Documentation

```
/**
   Describes a mailing address.
*/
public class Address {
   /**
      Formats the address.
      @return the address as a string with three lines
   */
   public String format() {
   }
}
```

Documentation

- Class Documentation in HTML Format





Object Orientation Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use javadoc to document method behavior
- Implement your program

Implementation

- After completing the design, implement your classes.
- The UML diagram will give instance variables:
  - Look for aggregated classes
  - They yield instance variables

Implementation

- After completing the design, implement your classes.
- The UML diagram will give instance variables:
  - Look for aggregated classes
  - They yield instance variables

- Invoice aggregates Address and LineItem.
- Every invoice has one billing address.
- An invoice can have many line items:

```
public class Invoice {
   . . .
   private Address billingAddress;
   private ArrayList<LineItem> items;
}
```

Implementation

- After completing the design, implement your classes.
- The UML diagram will give instance variables:
  - Look for aggregated classes
  - They yield instance variables

- Invoice aggregates Address and LineItem.
- Every invoice has one billing address.
- An invoice can have many line items:

```
public class Invoice {
  . . .
  private Address billingAddress;
  private ArrayList<LineItem> items;
}
```

- A line item needs to store a Product object and quantity:

```
public class LineItem {
  . . .
  private int quantity;
  private Product theProduct;
}
```

Implementation

- The methods themselves are now very easy.
- Example:
  - getTotalPrice of LineItem gets the unit price of the product and multiplies it with the quantity

```
/**
   Computes the total cost of this line item.
   @return the total price
*/
public double getTotalPrice() {
   return theProduct.getPrice() * quantity;
}
```

Implementation

- The methods themselves are now very easy.
- Example:
  - getTotalPrice of LineItem gets the unit price of the product and multiplies it with the quantity

```
/**
   Computes the total cost of this line item.
   @return the total price
*/
public double getTotalPrice() {
   return theProduct.getPrice() * quantity;
}
```

- Also supply constructors

```
Address samsAddress  = new Address("Sam's Small Appliances",
                  "100 Main Street", "Anytown", "CA", "98765");

Invoice samsInvoice = new Invoice(samsAddress);
samsInvoice.add(new Product("Toaster", 29.95), 3);
samsInvoice.add(new Product("Hair dryer", 24.95), 1);
samsInvoice.add(new Product("Car vacuum", 19.99), 2);

System.out.println(samsInvoice.format());
```

Implementation

```
/**
   Describes an invoice for a set of purchased products.
*/
public class Invoice {
   private Address billingAddress;
   private ArrayList<LineItem> items;

   /**
     Constructs an invoice.
     @param anAddress the billing address
   */
   public Invoice(Address anAddress) {
      items = new ArrayList<LineItem>();
       billingAddress = anAddress;
   }

   /**
      Adds a charge for a product to this invoice.
      @param aProduct the product that the customer ordered
      @param quantity the quantity of the product
   */
   public void add(Product aProduct, int quantity) {
      LineItem anItem = new LineItem(aProduct, quantity);
      items.add(anItem);
   }

   /**
      Formats the invoice.
      @return the formatted invoice
   */
   public String format() {
      String r =  "                        I N V O I C E\n\n"
         + billingAddress.format()
         + String.format("\n\n%-30s%8s%5s%8s\n",
            "Description", "Price", "Qty", "Total");

      for (LineItem item : items) {
         r = r + item.format() + "\n";
      }

      r = r + String.format("\nAMOUNT DUE: $%8.2f", getAmountDue());

      return r;
   }

   /**
      Computes the total amount due.
      @return the amount due
   */
   private double getAmountDue() {
      double amountDue = 0;
      for (LineItem item : items) {
         amountDue = amountDue + item.getTotalPrice();
      }
   return amountDue;
   }
}
```

Implementation

```
/**
   Describes a quantity of an article to purchase.
*/
public class LineItem {
   private int quantity;
   private Product theProduct;

   /**
      Constructs an item from the product and quantity.
      @param aProduct the product
      @param aQuantity the item quantity
   */
   public LineItem(Product aProduct, int aQuantity) {
      theProduct = aProduct;
      quantity = aQuantity;
   }

   /**
      Computes the total cost of this line item.
      @return the total price
   */
   public double getTotalPrice() {
      return theProduct.getPrice() * quantity;
   }

   /**
      Formats this item.
      @return a formatted string of this item
   */
   public String format() {
      return String.format("%-30s%8.2f%5d%8.2f",
         theProduct.getDescription(), theProduct.getPrice(),
         quantity, getTotalPrice());
   }
}
```

Implementation

```
/**
   Describes a product with a description and a price.
*/
public class Product {
   private String description;
   private double price;

   /**
      Constructs a product from a description and a price.
      @param aDescription the product description
      @param aPrice the product price
   */
   public Product(String aDescription, double aPrice) {
      description = aDescription;
      price = aPrice;
   }

   /**
      Gets the product description.
      @return the description
   */
   public String getDescription() {
      return description;
   }

   /**
      Gets the product price.
      @return the unit price
   */
   public double getPrice() {
      return price;
   }
}
```

Implementation

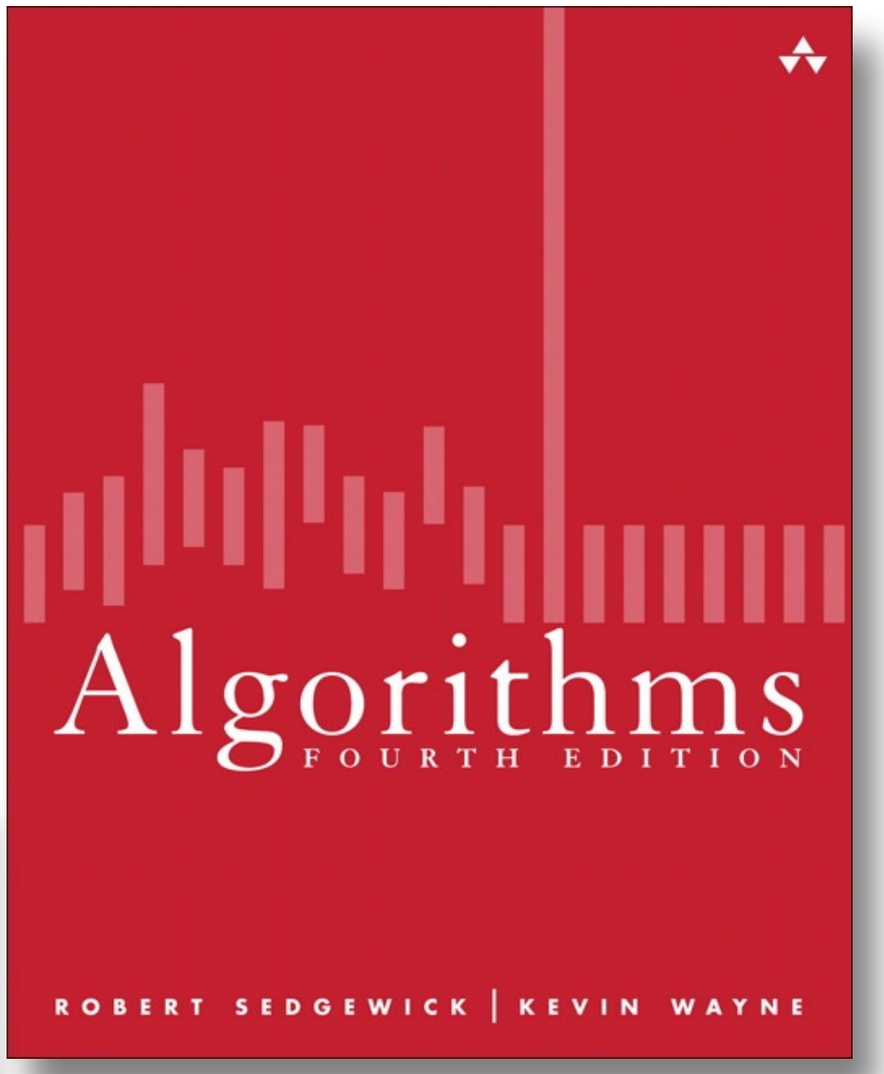
```
/**
   Describes a mailing address.
*/
public class Address {
   private String name;
   private String street;
   private String city;
   private String state;
   private String zip;

   /**
      Constructs a mailing address.
      @param aName the recipient name
      @param aStreet the street
      @param aCity the city
      @param aState the two-letter state code
      @param aZip the ZIP postal code
   */
   public Address(String aName, String aStreet,
         String aCity, String aState, String aZip) {
      name = aName;
      street = aStreet;
      city = aCity;
      state = aState;
      zip = aZip;
   }

   /**
      Formats the address.
      @return the address as a string with three lines
   */
   public String format() {
      return name + "\n" + street + "\n"
            + city + ", " + state + " " + zip;
   }
}
```

Homework 6

- Assignment - With an application that will present 2 questions and determine if the responses are true or false, you are to complete the methods that are inidcated by the // TODO: …

```
Who was the inventor of Java?
Your answer: James Gosling
true
In which country was the inventor of Java born?
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

```
Who was the inventor of Java?
Your answer: Topo Gigio
false
In which country was the inventor of Java born?
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 4
false
```



Assignment due next Monday at 11:59 PM

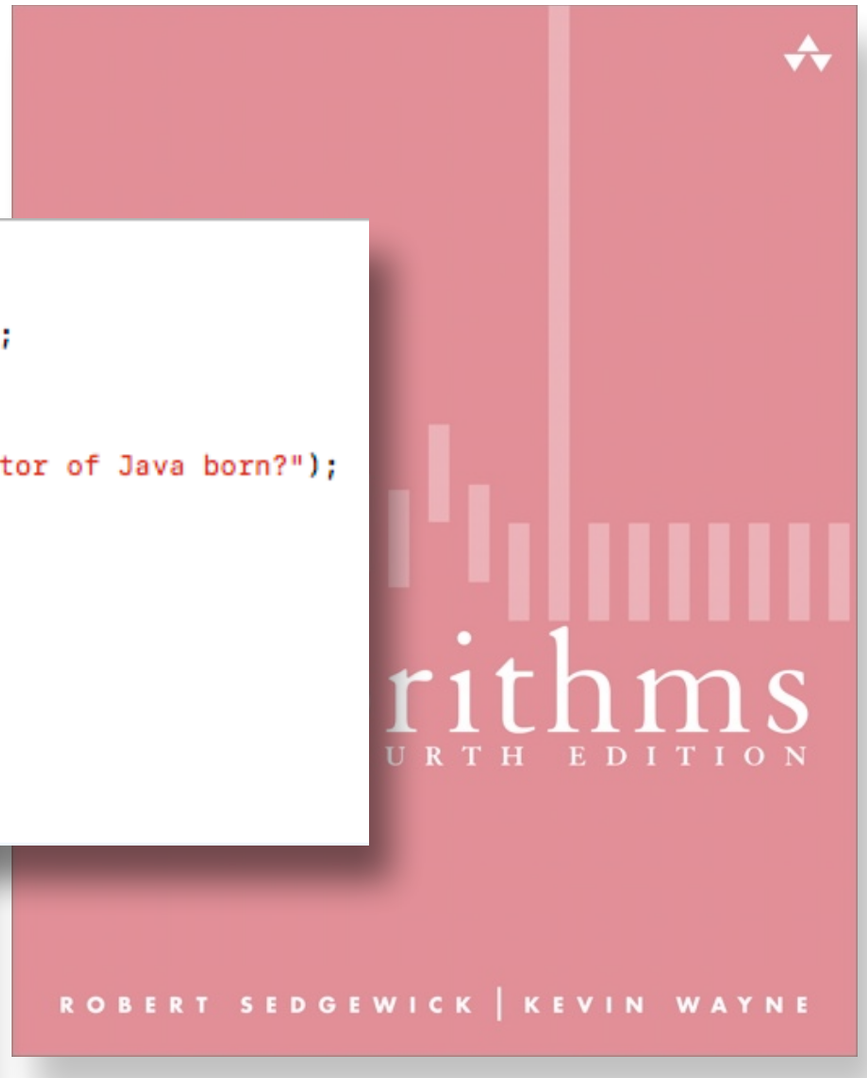
Homework 6

- Assignment - With an application that will present 2 questions and determine if the responses are true or false, you are to complete the methods that are inidcated by the // TODO: …

```
Who was the inventor of
Your answer: James Gosl
true
In which country was th
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

```
public class QuizDemo {
   public static void main(String[] args) {
      Question first = new Question();
      first.setText("Who was the inventor of Java?");
      first.setAnswer("James Gosling");

      ChoiceQuestion second = new ChoiceQuestion();
      second.setText("In which country was the inventor of Java born?");
      second.addChoice("Australia", false);
      second.addChoice("Canada", true);
      second.addChoice("Denmark", false);
      second.addChoice("United States", false);

      Quiz q = new Quiz();
      q.addQuestion(first);
      q.addQuestion(second);
      q.presentQuestions();
   }
}
```

```
false
In which country was the inventor of Java born?
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 4
false
```



Assignment due next Monday at 11:59 PM

Homework 6

- Assignment - With an application that will present 2 questions ar responses are true or f complete the methods the // TODO: …

```
Who was the inventor of Jav
Your answer: James Gosling
true
In which country was the in
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

```
Wh
Yo
fa
In
1:
2:
3:
4: United States
Your answer: 4
false
```

```
import java.util.ArrayList;
import java.util.Scanner;

/**
   A quiz contains a list of questions.
*/
public class Quiz {
   private ArrayList<Question> questions;

   /**
      Constructs a quiz with no questions.
   */
   public Quiz() {
      questions = new ArrayList<Question>();
   }

   /**
      Adds a question to this quiz.
      @param q the question
   */
   public void addQuestion(Question q) {
      questions.add(q);
   }

   /**
      Presents the questions to the user and checks the response.
   */
   public void presentQuestions() {
       // TODO: Implement this method
   }
}
```

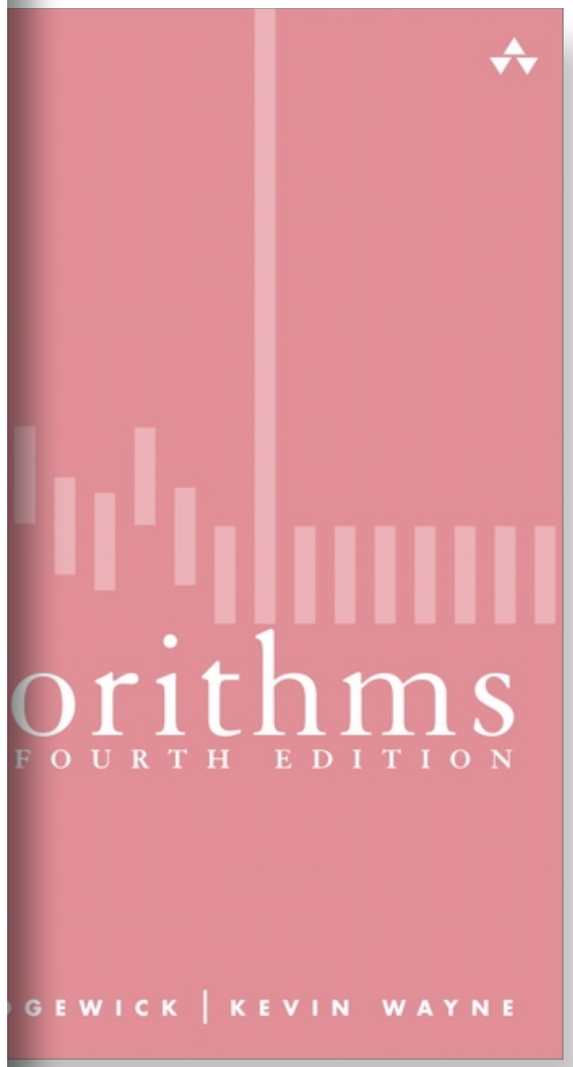
Assignment due next Monday at 11:59 PM

- Assignment - With an a
present 2 questions ar
responses are true or f
complete the methods
the // TODO: …

```
Who was the inventor of Jav
Your answer: James Gosling
true
In which country was the in
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

```
Wh
Yo
fa
In
1:
2:
3:
4:
Yo
fa
```

```
/**
   A question with a text and an answer.
*/
public class Question {
   private String text;
   private String answer;

   /**
      Constructs a question with empty question and answer.
   */
   public Question() {
      text = "";
      answer = "";
   }

   /**
      Sets the question text.
      @param questionText the text of this question
   */
   public void setText(String questionText) {
      text = questionText;
   }

   /**
      Sets the answer for this question.
      @param correctResponse the answer
   */
   public void setAnswer(String correctResponse) {
      answer = correctResponse;
   }

   /**
      Checks a given response for correctness.
      @param response the response to check
      @return true if the response was correct, false otherwise
   */
   public boolean checkAnswer(String response) {
       return false       // TODO: Implement this method
   }

   /**
      Displays this question.
   */
   public void display() {
       // TODO: Implement this method
   }
}
```



Assignment due next Monday at 11:59 PM

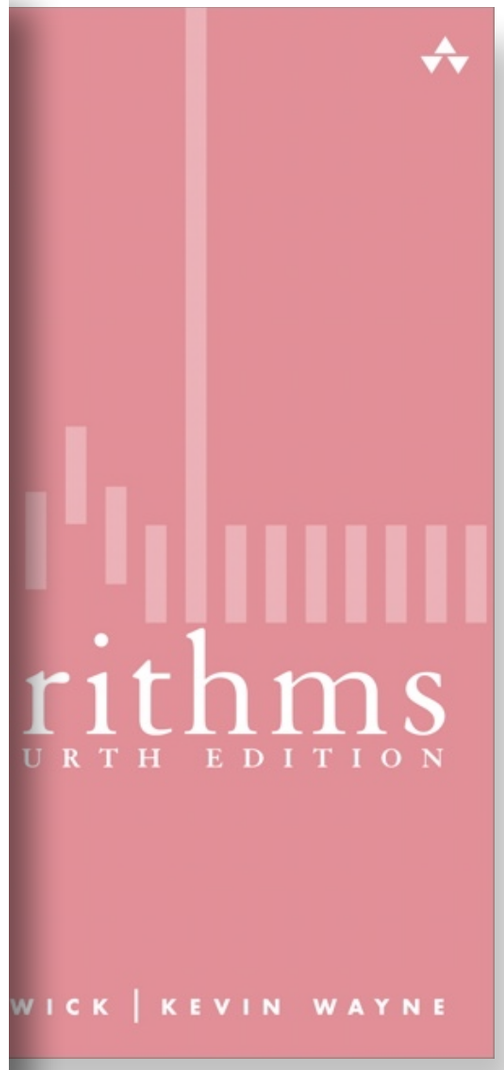
Homework 6

- Assignment - With a present 2 questions responses are true complete the metho the // TODO: …

```
Who was the inventor of
Your answer: James Gosl
true
In which country was th
1: Australia
2: Canada
3: Denmark
4: United States
Your answer: 2
true
```

```
Your answer: 4
false
```

```
import java.util.ArrayList;

/**
   A question with multiple choices.
*/
public class ChoiceQuestion extends Question {
   private ArrayList<String> choices;

   /**
      Constructs a choice question with no choices.
   */
   public ChoiceQuestion() {
      choices = new ArrayList<String>();
   }

   /**
      Adds an answer choice to this question.
      @param choice the choice to add
      @param correct true if this is the correct choice, false otherwise
   */
   public void addChoice(String choice, boolean correct) {
      // TODO: Implement this method
   }

   public void display() {
       // TODO: Implement this method

       // Display the question text
      // TODO: Implement this functionality

      // Display the answer choices
      // TODO: Implement this functionality
   }
}
```



Assignment due next Monday at 11:59 PM