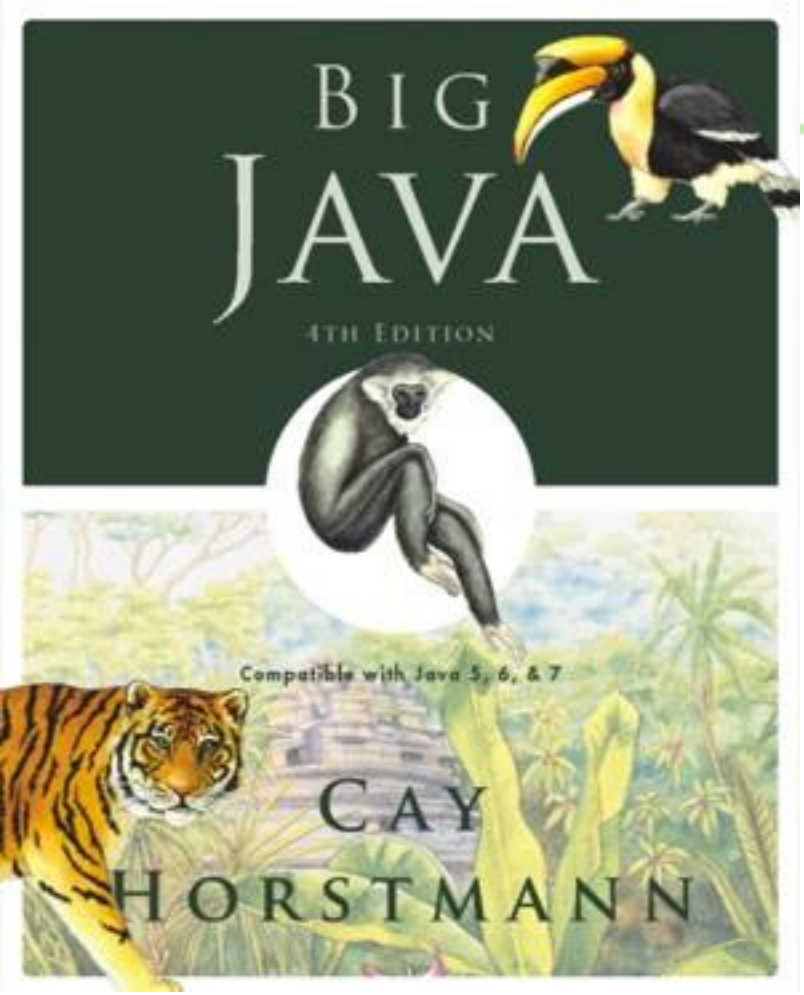


الفصل ١٦ - هياكل البيانات المتقدمة

Chapter 16 – Advanced Data Structures

Chapter Goals

- To learn about the set and map data types
- To understand the implementation of hash tables
- To be able to program hash functions
- To learn about binary trees
- To become familiar with the heap data structure
- To learn how to implement the priority queue data type
- To understand how to use heaps for sorting

- لمعرفة المزيد عن أنواع البيانات مجموعة وخريطة
- لفهم تنفيذ الجداول التجزئة
- لتكون قادرة على البرنامج وظائف تجزئة
- لمعرفة المزيد عن الأشجار الثنائية
- للتعرف على بنية البيانات كومة
- لمعرفة كيفية تنفيذ الأولوية طابور نوع البيانات
- لفهم كيفية استخدام أكوام لفرز

Sets

مجموعات

- **Set**: Unordered collection of distinct elements
- Elements can be added, located, and removed
- Sets don’t have duplicates

- مجموعة: جمع غير مرقمة من العناصر المتميزة
- يمكن إضافة عناصر، يقع، وإزالة
- مجموعات ليس لديها التكرارات

Fundamental Operations on a Set

- Adding an element
  - *Adding an element has no effect if the element is already in the set*
- Removing an element
  - *Attempting to remove an element that isn't in the set is silently ignored*
- Containment testing (Does the set contain a given object?)
- Listing all elements (in arbitrary order)

- إضافة عنصر
- إضافة عنصر له أي تأثير إذا العنصر هو بالفعل في المجموعة
- إزالة عنصر
- محاولة إزالة عنصر غير موجود في مجموعة يتم تجاهل بصمت
- اختبار الاحتواء (هل يحتوي مجموعة كائن معين؟)
- تسرد كل العناصر (من أجل التعسفي)

A Set of Printers

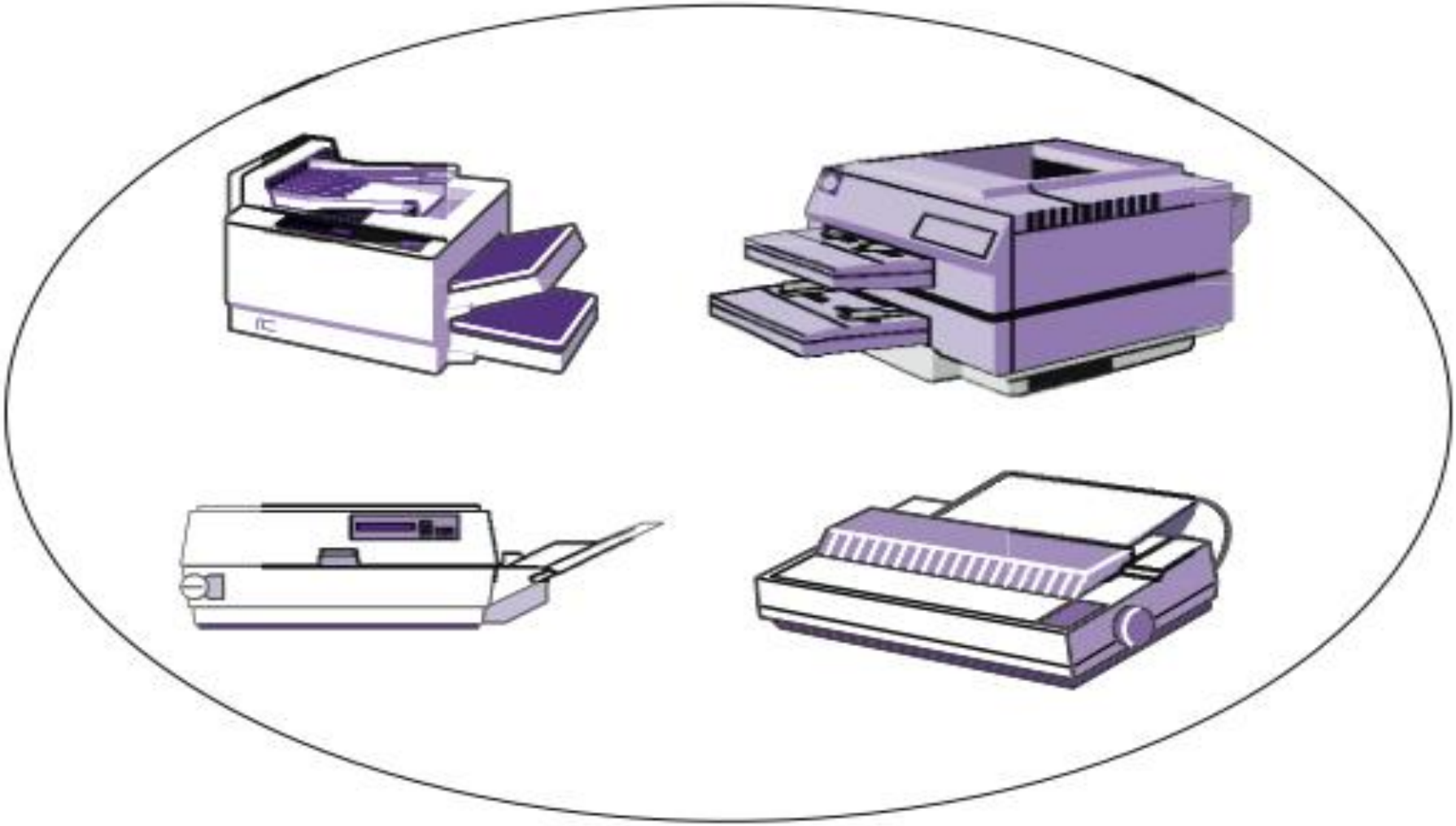


Figure 1 A Set of Printers

Sets

مجموعات

- We could use a linked list to implement a set
  - *Adding, removing, and containment testing would be relatively slow*
- There are data structures that can handle these operations much more quickly
  - *Hash tables*
  - *Trees*

- يمكننا استخدام قائمة مرتبطة لتنفيذ مجموعة
- إضافة وإزالة، وسيكون الاختبار الاحتواء تكون بطيئة نسبيا
- هناك هياكل البيانات التي يمكن التعامل مع هذه العمليات على نحو أسرع بكثير
- الجداول التجزئة
- الأشجار

Sets

- Standard Java library provides set implementations based on both data structures
  - *HashSet*
  - *TreeSet*
- Both of these data structures implement the `Set` interface
- As a rule of thumb, use a hash set unless you want to visit the set elements in sorted order

- توفر المكتبة القياسية جافا مجموعة تطبيقات على أساس كل من هياكل البيانات
- HashSet
- TreeSet
- كل من هذه الهياكل البيانات تطبيق واجهة مجموعة
- وكقاعدة عامة من الإبهام، واستخدام مجموعة التجزئة إلا إذا كنت ترغب في زيارة عناصر مجموعة في ترتيب فرزها

Set Classes and Interface in the Standard Library

تعيين فئات واجهة في مكتبة قياسي

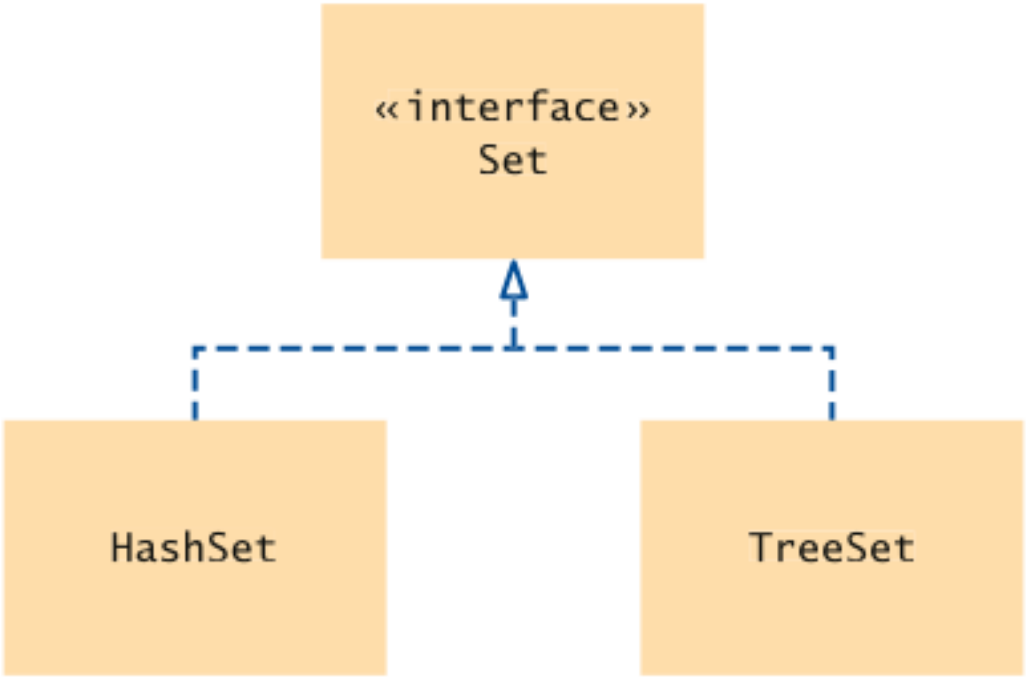


Figure 2
Set Classes and Interfaces in the Standard Library

Using a `Set`

- Example: Using a set of strings
- Construct the `Set`:

```
Set<String> names = new HashSet<String>();
```

or

```
Set<String> names = new TreeSet<String>();
```

- Add and remove elements:

```
names.add("Romeo");
names.remove("Juliet");
```

- Test whether an element is contained in the set:

```
if (names.contains("Juliet")) . . .
```

Iterator

- Use an iterator to visit all elements in a set
- A set iterator does not visit the elements in the order in which they were inserted
- An element cannot be added to a set at an iterator position
- A set element can be removed at an iterator position

- استخدام مكرر لزيارة جميع العناصر في مجموعة
- Aمكرر مجموعة لا زيارة العناصر بالترتيب الذي تم إدراجه
- لا يمكن إضافة عنصر إلى مجموعة في موقف مكرر
- يمكن إزالة عنصر مجموعة في موقف مكرر

Visiting All Elements with an Iterator

زيارة جميع عناصر مع مكرر

```
Iterator<String> iter = names.iterator();
while (iter.hasNext())
{
   String name = iter.next();
   Do something with name
}
```

or, using the “for each” loop:

```
for (String name : names)
{
   Do something with  name
}
```

Set Test Program

وضع برنامج اختبار

1. Read in all words from a dictionary file that contains correctly spelled words and place them into a set
2. Read all words from a document into a second set — here, the book “Alice in Wonderland”
3. Print all words from that set that are not in the dictionary set — potential misspellings

١. قراءة في كل كلمات من القاموس فاي لو أن يحتوي على كلمات مكتوبة بشكل صحيح ووضعها في مجموعة
٢. قراءة كل عبارة من مستند إلى المجموعة الثانية - هنا، وكتاب "أليس في بلاد العجائب"
٣. طباعة جميع الكلمات من أن المجموعة التي ليست في مجموعة القاموس - الأخطاء الإملائية المحتملة

ch16/spellcheck/SpellCheck.java

```
1  import java.util.HashSet;
2  import java.util.Scanner;
3  import java.util.Set;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6
7  /**
8     This program checks which words in a file are not present in a dictionary.
9  */
10 public class SpellCheck
11 {
12    public static void main(String[] args)
13       throws FileNotFoundException
14    {
15       // Read the dictionary and the document
16
17       Set<String> dictionaryWords = readWords("words");
18       Set<String> documentWords = readWords("alice30.txt");
19
```

Continued

ch16/spellcheck/SpellCheck.java (cont.)

```
15          // Read the dictionary and the document
16
17          Set<String> dictionaryWords = readWords("words");
18          Set<String> documentWords = readWords("alice30.txt");
19
20          // Print all words that are in the document but not the dictionary
21
22          for (String word : documentWords)
23          {
24             if (!dictionaryWords.contains(word))
25             {
26                System.out.println(word);
27             }
28          }
29       }
30
```

Continued

ch16/spellcheck/SpellCheck.java (cont.)

```
31      /**
32         Reads all words from a file.
33         @param filename the name of the file
34         @return a set with all lowercased words in the file. Here, a
35         word is a sequence of upper- and lowercase letters.
36      */
37      public static Set<String> readWords(String filename)
38         throws FileNotFoundException
39      {
40         Set<String> words = new HashSet<String>();
41         Scanner in = new Scanner(new File(filename));
42         // Use any characters other than a-z or A-Z as delimiters
43         in.useDelimiter("[^a-zA-Z]+");
44         while (in.hasNext())
45         {
46            words.add(in.next().toLowerCase());
47         }
48         return words;
49      }
50   }
```

Continued

ch16/spellcheck/SpellCheck.java (cont.)

Program Run:

```
neighbouring
croqueted
pennyworth
dutchess
comfits
xii
dinn
clamour
...
```

Self Check 16.1

Arrays and lists remember the order in which you added elements; sets do not. Why would you want to use a set instead of an array or list?

Answer: Efficient set implementations can quickly test whether a given element is a member of the set.

- المصفوفات والقوائم تذكر الترتيب الذي أضفته العناصر؛ مجموعات لا. لماذا تريد استخدام مجموعة بدلا من صفيف أو قائمة؟
- الجواب: يمكن أن مجموعة تطبيقات فعالة اختبار بسرعة ما إذا كان عنصر معين هو عضو في مجموعة.

Self Check 16.2

Why are set iterators different from list iterators?

Answer: Sets do not have an ordering, so it doesn't make sense to add an element at a particular iterator position, or to traverse a set backwards.

- لماذا يتم تعيين مختلفة iteratorsمن iteratorsالقائمة؟
- الجواب: مجموعات ليس لها ترتيب، لذلك لا معنى لإضافة عنصر في موقف مكرر معين، أو لاجتياز مجموعة الوراء.

Self Check 16.3

Suppose you changed line 18 of the `SpellCheck` program to use a `TreeSet` instead of a `HashSet`. How would the output change?

Answer: The words would be listed in sorted order.

- افترض أنك غيرت خط ١٨ من برنامج التدقيق الإملائي لاستخدام TreeSetبدلا من HashSet. كيف يمكن تغيير الانتاج؟
- الإجابة: سيتم سرد الكلمات من أجل فرزها.

Self Check 16.4

When would you choose a tree set over a hash set?

Answer: When it is desirable to visit the set elements in sorted order.

- عندما كنت ستختار شجرة موكلا مجموعة التجزئة؟
- الإجابة: عندما يكون مرغوبا فيه لزيارة عناصر مجموعة في ترتيب فرزها.

Maps

خرائط

- A map keeps associations between key and value objects
- Mathematically speaking, a map is a function from one set, the *key set*, to another set, the *value set*
- Every key in a map has a unique value
- A value may be associated with several keys
- Classes that implement the `Map` interface
  - *`HashMap`*
  - *`TreeMap`*
- Both of these data structures implement the `Map` interface
- As a rule of thumb, use a hash set unless you want to visit the set elements in sorted order

- خريطة تبقي الارتباطات بين الأشياء مفتاح وقيمة
- رياضيا الناطقة، والخريطة هي وظيفة من مجموعة واحدة، ومجموعة رئيسية، إلى مجموعة أخرى، ومجموعة قيمة
- كل مفتاح في خريطة له قيمة واحدة
- قد تترافق قيمة مع عدة مفاتيح
- الفئات التي تقوم بتنفيذ واجهة الخريطة
- HashMap
- TreeMap
- كل من هذه الهياكل البيانات تطبيق واجهة الخريطة
- وكقاعدة عامة من الإبهام، واستخدام مجموعة التجزئة إلا إذا كنت ترغب في زيارة عناصر مجموعة في ترتيب فرزها

An Example of a Map

مثال على خريطة

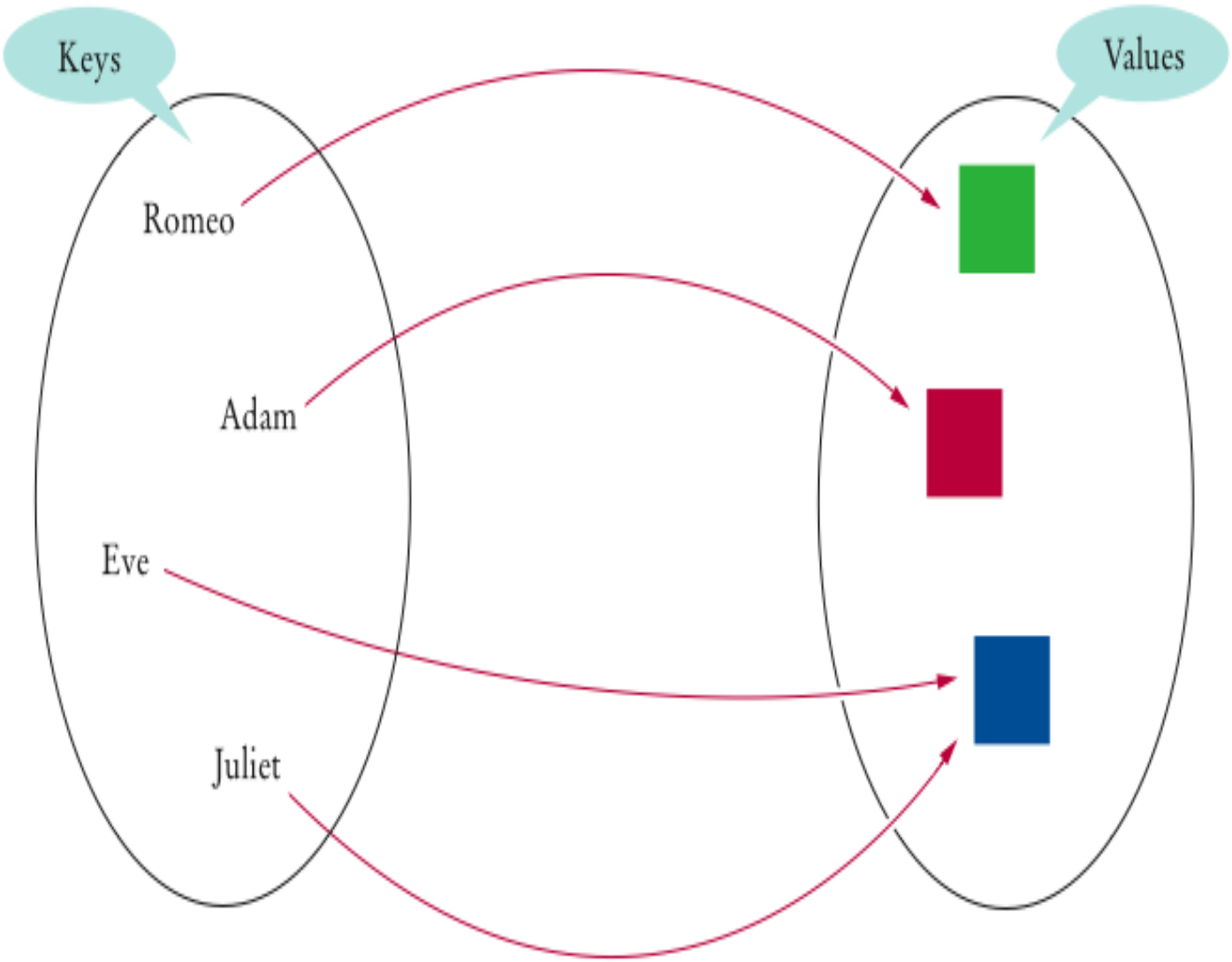


Figure 3 A Map

Map Classes and Interfaces

فصول الخريطة واجهات

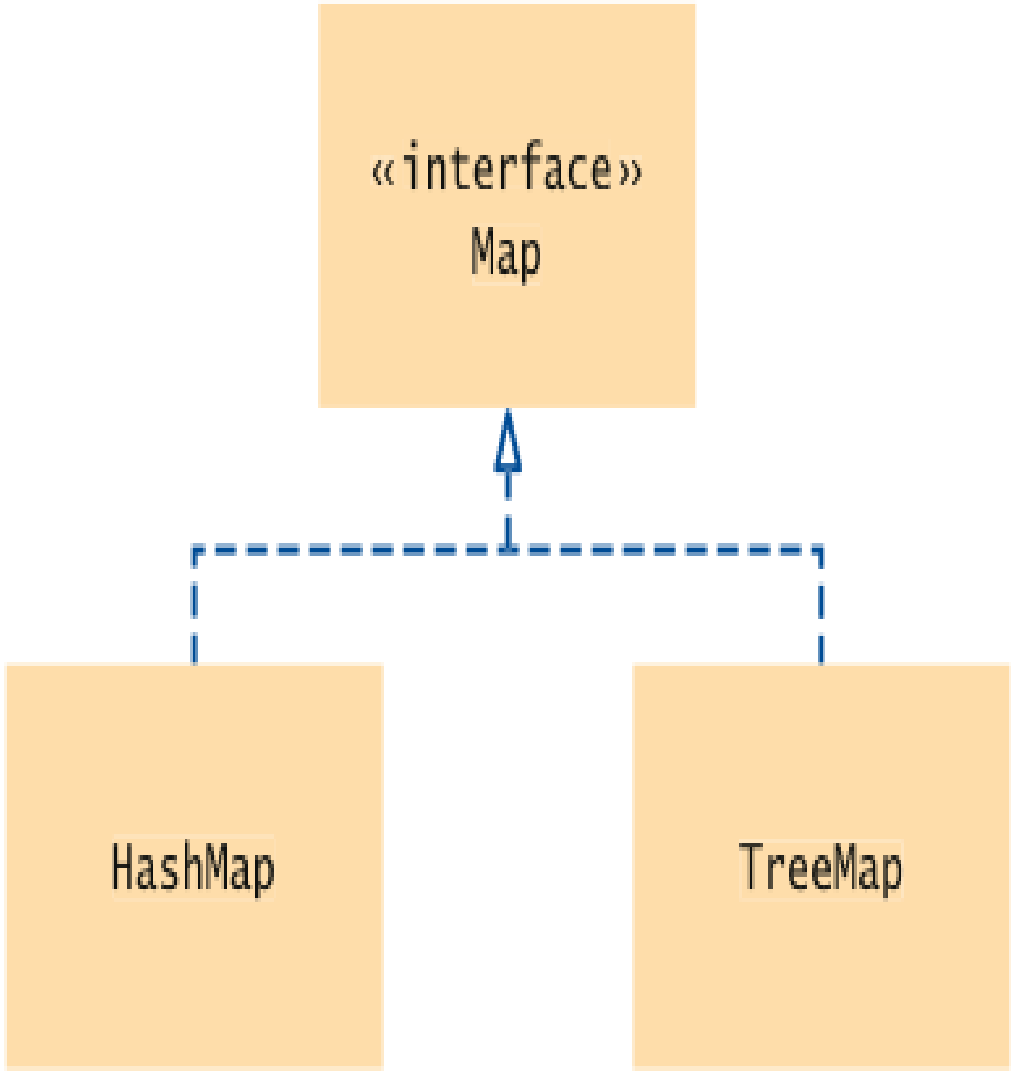


Figure 4
Map Classes and Interfaces in the Standard Library

Using a `Map`

- Example: Associate names with colors

مثال: أسماء المنتسبين مع الألوان

- Construct the `Map`:

```
Map<String, Color> favoriteColors =
   new HashMap<String, Color>();
```

or

```
Map<String, Color> favoriteColors =
   new TreeMap<String, Color>();
```

- Add an association:

إضافة رتباط

```
favoriteColors.put("Juliet", Color.RED);
```

- Change an existing association:

تغيير اقتران موجودة:

```
favoriteColors.put("Juliet",Color.BLUE);
```

Using a `Map`

- Get the value associated with a key: الحصول على قيمة المرتبطة مفتاح:

```
Color julietsFavoriteColor =
   favoriteColors.get("Juliet”);
```

- Remove a key and its associated value: إزالة مفتاح والقيمة المرتبطة بها:

```
favoriteColors.remove("Juliet");
```

Printing Key/Value Pairs

```
Set<String> keySet = m.keySet();
for (String key : keySet)
{
   Color value = m.get(key);
   System.out.println(key + ” : " + value);
}
```

ch16/map/MapDemo.java

```
 1  import java.awt.Color;
 2  import java.util.HashMap;
 3  import java.util.Map;
 4  import java.util.Set;
 5
 6  /**
 7     This program demonstrates a map that maps names to colors.
 8  */
 9  public class MapDemo
10  {
11     public static void main(String[] args)
12     {
13        Map<String, Color> favoriteColors = new
HashMap<String, Color>();
14        favoriteColors.put("Juliet", Color.BLUE);
15        favoriteColors.put("Romeo", Color.GREEN);
16        favoriteColors.put("Adam", Color.RED);
17        favoriteColors.put("Eve", Color.BLUE);
18
```

Continued

ch16/map/MapDemo.java (cont.)

```
19          // Print all keys and values in the map
20
21          Set<String> keySet = favoriteColors.keySet();
22          for (String key : keySet)
23          {
24             Color value = favoriteColors.get(key);
25             System.out.println(key + " : " + value);
26          }
27       }
28    }
```

Program Run:

```
Romeo : java.awt.Color[r=0,g=255,b=0]
Eve : java.awt.Color[r=0,g=0,b=255]
Adam : java.awt.Color[r=255,g=0,b=0]
Juliet : java.awt.Color[r=0,g=0,b=255]
```

Continued

Self Check 16.5

What is the difference between a set and a map?

Answer: A set stores elements. A map stores associations between keys and values.

- ما هو الفرق بين مجموعة وخريطة؟
- الإجابة: مجموعة مخازن العناصر. خريطة مخازن الجمعيات بين المفاتيح والقيم.

Self Check 16.6

Why is the collection of the keys of a map a set?

Answer: The ordering does not matter, and you cannot have duplicates.

- لماذا هو مجموعة من مفاتيح خريطة مجموعة؟
- الإجابة: لا يهم يأمر، وأنت لا يمكن أن يكون التكرارات.

Hash Tables

- **Hashing** can be used to find elements in a data structure quickly without making a linear search
- A **hash table** can be used to implement sets and maps
- A **hash function** computes an integer value (called the **hash code**) from an object
- A good hash function minimizes *collisions* — identical hash codes for different objects
- To compute the hash code of object `x`:

```
int h = x.hashCode();
```

- تجزئة: يمكن استخدامها للعثور على العناصر في بنية البيانات بسرعة من دون البحث الخطي
- جدول تجزئة: يمكن استخدامها لتنفيذ مجموعات والخرائط
- ودالة البعثرة: يحسب قيمة عددية (تسمى رمز التجزئة) من كائن
- ودالة البعثرة جيدة يقلل من التصادمات - رموز تجزئة مماثلة لكائنات مختلفة
- لحساب رمز التجزئة من وجوه

Sample Strings and Their Hash Codes

عينة سلاسل ورموز تلك البعثرة

String	Hash Code
"Adam"	2035631
"Eve"	700068
"Harry"	69496448
"Jim"	74478
"Joe"	74656
"Juliet"	-2065036585
"Katherine"	2079199209
"Sue"	83491

Simplistic Implementation of a Hash Table

تنفيذ التبسيط من جدول تجزئة

- To implement
  - *Generate hash codes for objects*
  - *Make an array*
  - *Insert each object at the location of its hash code*
- To test if an object is contained in the set
  - *Compute its hash code*
  - *Check if the array position with that hash code is already occupied*

- لتنفيذ
- توليد رموز تجزئة الأجسام
- جعل مجموعة
- إدراج كل كائن في موقع رمز التجزئة لها
- لاختبار إذا ورد كائن في المجموعة
- حساب رمز التجزئة لها
- معرفة ما اذا كان موقف مجموعة مع ذلك رمز التجزئة محتلة بالفعل

Simplistic Implementation of a Hash Table

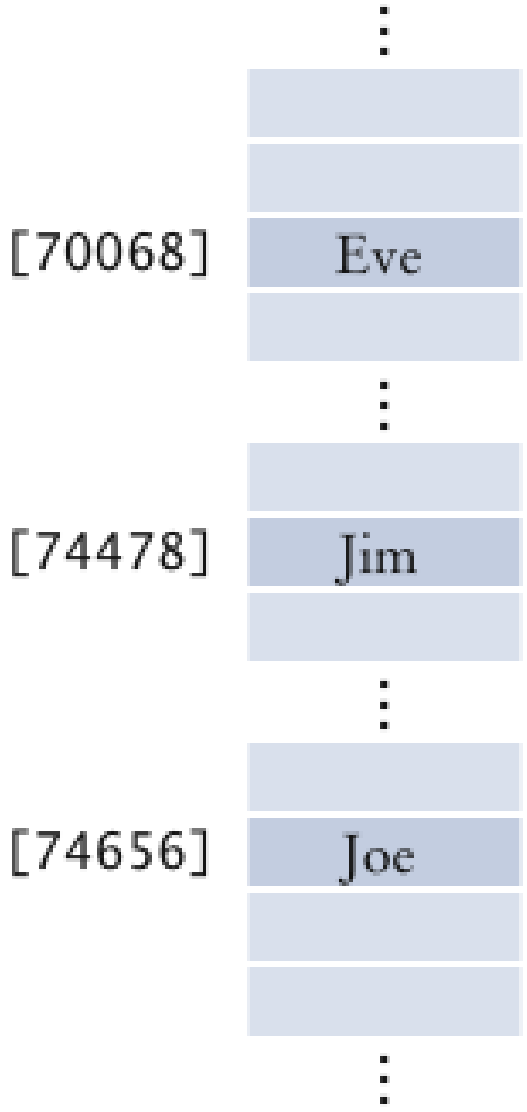


Figure 5
A Simplistic Implementation of a Hash Table

Problems with Simplistic Implementation

- It is not possible to allocate an array that is large enough to hold all possible integer index positions
- It is possible for two different objects to have the same hash code

- ليس من الممكن تخصيص مجموعة كبيرة بما يكفي لاستيعاب كافة المواقف مؤشر عدد صحيح ممكنة
- فمن الممكن لكائنين مختلفة لديهم نفس رمز التجزئة

Solutions

حلول

- Pick a reasonable array size and reduce the hash codes to fall inside the array

```
int h = x.hashCode();
if (h < 0) h = -h;
position = h % buckets.length;
```

- When elements have the same hash code:
  - *Use a node sequence to store multiple objects in the same array position*
  - *These node sequences are called buckets*

اختيار حجم مجموعة معقولة والحد من رموز التجزئة تقع داخل المصفوفة

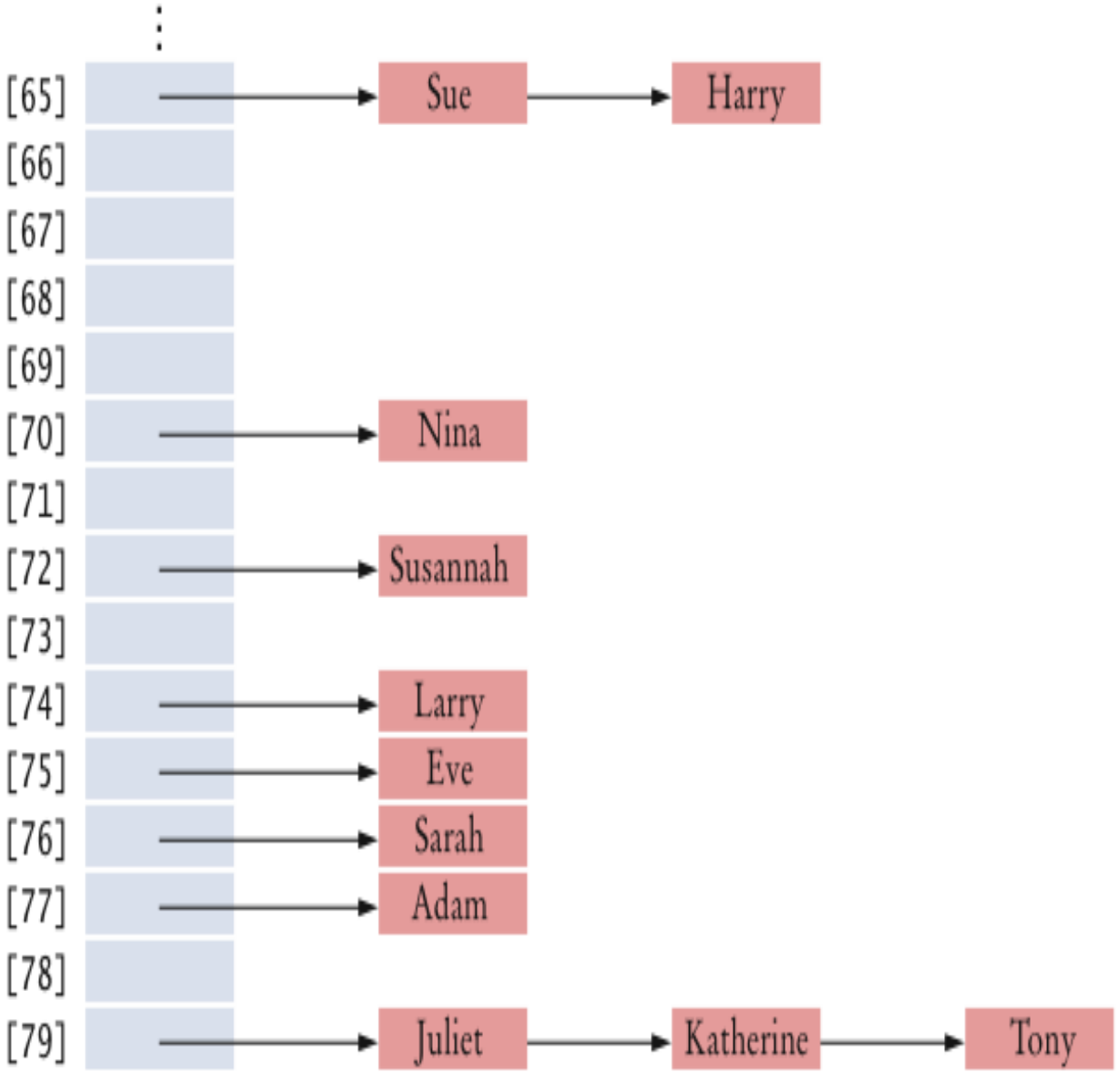
```
int h = x.hashCode();
if (h < 0) h = -h;
position = h % buckets.length;
```

- عندما يكون عناصر نفس رمز التجزئة:
- استخدام تسلسل عقدة لتخزين كائنات متعددة في موقف مجموعة نفسه
- وتسمى هذه المتتاليات عقدة الدلاء

Hash Table with Buckets to Store Elements with Same Hash Code

- جدول التجزئة مع دلاء إلى مخزن عناصر مع نفس رمز تجزئة

Figure 6
A Hash Table with Buckets to Store Elements with the Same Hash Code



Algorithm for Finding an Object `x` in a Hash Table

1. Get the index `h` into the hash table
   - *Compute the hash code*
   - *Reduce it modulo the table size*
2. Iterate through the elements of the bucket at position `h`
   - *For each element of the bucket, check whether it is equal to `x`*
3. If a match is found among the elements of that bucket, then `x` is in the set
   - *Otherwise, `x` is not in the set*

خوارزمية لإيجاد كائن العاشر في جدول تجزئة

- الحصول على مؤشر ساعة في جدول التجزئة
- حساب رمز التجزئة
- الحد منه مودولو حجم الجدول
- تكرار خلال عناصر دلو في موقف ح
- لكل عنصر من دلو، والتحقق ما إذا كان يساوي x
- إذا تم العثور على تطابق بين عناصر هذا دلو، ثم xهو في المجموعة
- خلاف ذلك، س ليس في المجموعة

Hash Tables

- A hash table can be implemented as an array of buckets
- Buckets are sequences of nodes that hold elements with the same hash code
- If there are few collisions, then adding, locating, and removing hash table elements takes constant time
  - *Big-Oh notation: O(1)*
- For this algorithm to be effective, the bucket sizes must be small
- The table size should be a prime number larger than the expected number of elements
  - *An excess capacity of 30% is typically recommended*

- ويمكن تنفيذ جدول تجزئة باعتبارها مجموعة من الدلاء
- دلاء هي متواليات من العقد التي تحمل العناصر مع نفس رمز التجزئة
- إذا كان هناك عدد قليل من الاصطدامات، ثم إضافة، وتحديد مكان وإزالة عناصر جدول التجزئة يستغرق وقتا المستمر
- كبير أوه تدوين: O (1)
- لهذه الخوارزمية لتكون فعالة، يجب أن تكون أحجام دلو صغير
- يجب أن يكون حجم الجدول عدد أولي أكبر من العدد المتوقع من العناصر
- وعادة ما أوصت الطاقة الفائضة من ٣٠٪

Hash Tables

الجداول التجزئة

- Adding an element: Simple extension of the algorithm for finding an object
  - *Compute the hash code to locate the bucket in which the element should be inserted*
  - *Try finding the object in that bucket*
  - *If it is already present, do nothing; otherwise, insert it*
- Removing an element is equally simple
  - *Compute the hash code to locate the bucket in which the element should be inserted*
  - *Try finding the object in that bucket*
  - *If it is present, remove it; otherwise, do nothing*
- If there are few collisions, adding or removing takes *O*(1) time

- إضافة عنصر: تمديد بسيط للخوارزمية لإيجاد كائن
- حساب رمز التجزئة لتحديد موقع دلو التي ينبغي أن تدرج العنصر
- محاولة العثور على الكائن في ذلك دلو
- إذا كان موجود بالفعل، لا تفعل شيئا. خلاف ذلك، أدخله
- إزالة عنصر بسيط على قدم المساواة
- حساب رمز التجزئة لتحديد موقع دلو التي ينبغي أن تدرج العنصر
- محاولة العثور على الكائن في ذلك دلو
- إذا كان موجودا، إزالته. خلاف ذلك، لا تفعل شيئا
- إذا كان هناك عدد قليل من الاصطدامات، إضافة أو إزالة يأخذ O (1) وقت

ch16/hashtable/HashSet.java

```
 1  import java.util.AbstractSet;
 2  import java.util.Iterator;
 3  import java.util.NoSuchElementException;
 4
 5  /**
 6     A hash set stores an unordered collection of objects, using
 7     a hash table.
 8  */
 9  public class HashSet extends AbstractSet
10  {
11     private Node[] buckets;
12     private int size;
13
14     /**
15        Constructs a hash table.
16        @param bucketsLength the length of the buckets array
17     */
18     public HashSet(int bucketsLength)
19     {
20        buckets = new Node[bucketsLength];
21        size = 0;
22     }
23
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
24      /**
25         Tests for set membership.
26         @param x an object
27         @return true if x is an element of this set
28      */
29      public boolean contains(Object x)
30      {
31         int h = x.hashCode();
32         if (h < 0) h = -h;
33         h = h % buckets.length;
34
35         Node current = buckets[h];
36         while (current != null)
37         {
38            if (current.data.equals(x)) return true;
39            current = current.next;
40         }
41         return false;
42      }
43
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
44      /**
45         Adds an element to this set.
46         @param x an object
47         @return true if x is a new object, false if x was
48         already in the set
49      */
50      public boolean add(Object x)
51      {
52         int h = x.hashCode();
53         if (h < 0) h = -h;
54         h = h % buckets.length;
55
56         Node current = buckets[h];
57         while (current != null)
58         {
59            if (current.data.equals(x))
60               return false; // Already in the set
61            current = current.next;
62         }
63         Node newNode = new Node();
64         newNode.data = x;
65         newNode.next = buckets[h];
66         buckets[h] = newNode;
67         size++;
68         return true;
69      }
70
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
71      /**
72         Removes an object from this set.
73         @param x an object
74         @return true if x was removed from this set, false
75         if x was not an element of this set
76      */
77      public boolean remove(Object x)
78      {
79         int h = x.hashCode();
80         if (h < 0) h = -h;
81         h = h % buckets.length;
82
83         Node current = buckets[h];
84         Node previous = null;
85         while (current != null)
86         {
87            if (current.data.equals(x))
88            {
89               if (previous == null) buckets[h] = current.next;
90               else previous.next = current.next;
91               size--;
92               return true;
93            }
94            previous = current;
95            current = current.next;
96         }
97         return false;
98      }
99
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
100       /**
101          Returns an iterator that traverses the elements of this set.
102          @return a hash set iterator
103       */
104       public Iterator iterator()
105       {
106          return new HashSetIterator();
107       }
108
109       /**
110          Gets the number of elements in this set.
111          @return the number of elements
112       */
113       public int size()
114       {
115          return size;
116       }
117
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
118     class Node
119     {
120        public Object data;
121        public Node next;
122     }
123
124     class HashSetIterator implements Iterator
125     {
126        private int bucket;
127        private Node current;
128        private int previousBucket;
129        private Node previous;
130
131        /**
132           Constructs a hash set iterator that points to the
133           first element of the hash set.
134        */
135        public HashSetIterator()
136        {
137           current = null;
138           bucket = -1;
139           previous = null;
140           previousBucket = -1;
141        }
142
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
143     public boolean hasNext()
144 {
145       if (current != null && current.next != null)
146             return true;
147       for (int b = bucket + 1; b < buckets.length; b++)
148          if (buckets[b] != null) return true;
149       return false;
150 }
151
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
152         public Object next()
153         {
154            previous = current;
155            previousBucket = bucket;
156            if (current == null || current.next == null)
157            {
158               // Move to next bucket
159               bucket++;
160
161               while (bucket < buckets.length
162                     && buckets[bucket] == null)
163                  bucket++;
164               if (bucket < buckets.length)
165                  current = buckets[bucket];
166               else
167                  throw new NoSuchElementException();
168            }
169            else // Move to next element in bucket
170               current = current.next;
171            return current.data;
172         }
173
```

Continued

ch16/hashtable/HashSet.java (cont.)

```
174         public void remove()
175         {
176            if (previous != null && previous.next == current)
177               previous.next = current.next;
178            else if (previousBucket < bucket)
179               buckets[bucket] = current.next;
180            else
181               throw new IllegalStateException();
182            current = previous;
183            bucket = previousBucket;
184         }
185      }
186   }
```

Continued

ch16/hashtable/HashSetDemo.java

```
 1  import java.util.Iterator;
 2  import java.util.Set;
 3
 4  /**
 5     This program demonstrates the hash set class.
 6  */
 7  public class HashSetDemo
 8  {
 9     public static void main(String[] args)
10     {
11        Set names = new HashSet(101); // 101 is a prime
12
13        names.add("Harry");
14        names.add("Sue");
15        names.add("Nina");
16        names.add("Susannah");
17        names.add("Larry");
18        names.add("Eve");
19        names.add("Sarah");
20        names.add("Adam");
21        names.add("Tony");
22        names.add("Katherine");
23        names.add("Juliet");
```

Continued

ch16/hashtable/HashSetDemo.java (cont.)

```
24          names.add("Romeo");
25          names.remove("Romeo");
26          names.remove("George");
27
28          Iterator iter = names.iterator();
29          while (iter.hasNext())
30             System.out.println(iter.next());
31       }
32    }
```

Program Run:

```
Harry
Sue
Nina
Susannah
Larry
Eve
Sarah
Adam
Juliet
Katherine
Tony
```

Self-Check 15.7

If a hash function returns 0 for all values, will the `HashSet` work correctly?

Answer: Yes, the hash set will work correctly. All elements will be inserted into a single bucket.

- إذا وظيفة تجزئة بإرجاع ٠ لجميع القيم، فإن HashSetتعمل بشكل صحيح؟
- الجواب: نعم، فإن مجموعة التجزئة تعمل بشكل صحيح. سيتم إدراج جميع العناصر في دلو واحد.

Self Check 16.8

What does the `hasNext` method of the `HashSetIterator` do when it has reached the end of a bucket?

Answer: It locates the next bucket in the bucket array and points to its first element.

- ماذا طريقة hasNext منHashSetIterator تفعل عندما وصلت إلى نهاية دلو؟
- الجواب: انه يقع دلو المقبل في مجموعة دلو ونقطة لأول عنصرها.

Computing Hash Codes

- A hash function computes an integer hash code from an object
- Choose a hash function so that different objects are likely to have different hash codes.
- Bad choice for hash function for a string
  - *Adding the unicode values of the characters in the string:*

```
int h = 0;
for (int i = 0; i < s.length(); i++)
   h = h + s.charAt(i);
```

  - *Because permutations (`"eat"` and `"tea"`) would have the same hash code*

- ودالة البعثرة يحسب رمز صحيح التجزئة من كائن
- اختيار وظيفة تجزئة بحيث يحتمل أن يكون لها رموز تجزئة مختلفة كائنات مختلفة.
- سوء اختيار للدالة البعثرة لسلسلة مضيفا القيم يونيكود من الشخصيات في السلسلة:

```
int h = 0;
for (int i = 0; i
   < s.length();
         i++)
      h = h +
   s.charAt(i);
```

- لأن التباديل ("أكل" و "الشاي") سوف يكون له نفس رمز التجزئة

Computing Hash Codes

- Hash function for a string `s` from standard library

```
final int HASH_MULTIPLIER = 31;
int h = 0;
for (int i = 0; i < s.length(); i++)
   h = HASH_MULTIPLIER * h + s.charAt(i)
```

- وظيفة تجزئة ل ق سلسلة من المكتبة القياسية

- For example, the hash code of *"eat"* is

```
31 * (31 * 'e' + 'a') + 't' = 100184
```

- The hash code of *"tea"* is quite different, namely

```
31 * (31 * 't' + 'e') + 'a' = 114704
```

- رمز تجزئة "الشاي" مختلف تماما، وهي

A `hashCode` Method for the `Coin` Class

- There are two instance fields: `String` coin name and `double` coin value
- Use `String`'s `hashCode` method to get a hash code for the name
- To compute a hash code for a floating-point number:
  - *Wrap the number into a `Double` object*
  - *Then use `Double`'s `hashCode` method*
- Combine the two hash codes using a prime number as the `HASH_MULTIPLIER`

- هناك حقلين سبيل المثال: اسم سلسلة عملة وقيمة عملة مزدوجة
- استخدام أسلوب hashCodeسلسلة للحصول على رمز التجزئة للحصول على اسم
- لحساب رمز التجزئة لعدد الفاصلة العائمة:
- التفاف الرقم إلى كائن مزدوج
- ثم استخدام طريقة مزدوجة لhashCode
- الجمع بين الرموز تجزئة اثنين باستخدام عدد الوزراء باعتباره HASH_MULTIPLIER

A `hashCode` Method for the `Coin` Class

```
class Coin
{
   public int hashCode()
   {
      int h1 = name.hashCode();
      int h2 = new Double(value).hashCode();
      final int HASH_MULTIPLIER = 29;
      int h = HASH_MULTIPLIER * h1 + h2;
      return h;
   }
   ...
}
```

Creating Hash Codes for your Classes

- Use a prime number as the `HASH_MULTIPLIER`
- Compute the hash codes of each instance field
- For an integer instance field just use the field value
- Combine the hash codes:

```
int h = HASH_MULTIPLIER * h1 + h2;
h = HASH_MULTIPLIER * h + h3;
h = HASH_MULTIPLIER *h + h4;
...
return h;
```

- استخدام عدد الوزراء باعتباره HASH_MULTIPLIER
- حساب رموز تجزئة كل حقل المثال
- لحقل سبيل المثال عدد صحيح مجرد استخدام قيمة الحقل
- الجمع بين رموز التجزئة:

Creating Hash Codes for your Classes

- Your `hashCode` method must be compatible with the `equals` method
  - *if* `x.equals(y)` *then* `x.hashCode() == y.hashCode()`
- You get into trouble if your class defines an `equals` method but not a `hashCode` method
  - *If we forget to define* `hashCode` *method for* `Coin` *it inherits the method from* `Object` *superclass*
  - *That method computes a hash code from the memory location of the object*
  - *Effect: Any two objects are very likely to have a different hash code*

```
Coin coin1 = new Coin(0.25, "quarter");
Coin coin2 = new Coin(0.25, "quarter");
```

- In general, define either both `hashCode` and `equals` methods or neither

Hash Maps

- In a hash map, only the keys are hashed
- The keys need compatible `hashCode` and `equals` method

- في خارطة التجزئة، وتجزئته فقط مفاتيح
- مفاتيح تحتاج hashCode متوافق ويساوي طريقة

ch16/hashcode/Coin.java

```
1  /**
2     A coin with a monetary value.
3  */
4  public class Coin
5  {
6     private double value;
7     private String name;
8
9     /**
10       Constructs a coin.
11       @param aValue the monetary value of the coin.
12       @param aName the name of the coin
13    */
14    public Coin(double aValue, String aName)
15    {
16       value = aValue;
17       name = aName;
18    }
19
```

Continued

ch16/hashcode/Coin.java (cont.)

```
20      /**
21         Gets the coin value.
22         @return the value
23      */
24      public double getValue()
25      {
26         return value;
27      }
28
29      /**
30         Gets the coin name.
31         @return the name
32      */
33      public String getName()
34      {
35         return name;
36      }
37
```

Continued

ch16/hashcode/Coin.java (cont.)

```
38      public boolean equals(Object otherObject)
39      {
40         if (otherObject == null) return false;
41         if (getClass() != otherObject.getClass()) return false;
42         Coin other = (Coin) otherObject;
43         return value == other.value && name.equals(other.name);
44      }
45
46      public int hashCode()
47      {
48         int h1 = name.hashCode();
49         int h2 = new Double(value).hashCode();
50         final int HASH_MULTIPLIER = 29;
51         int h = HASH_MULTIPLIER * h1 + h2;
52         return h;
53      }
54
55      public String toString()
56      {
57         return "Coin[value=" + value + ",name=" + name + "]";
58      }
59   }
```

ch16/hashcode/CoinHashCodePrinter.java

```
 1  import java.util.HashSet;
 2  import java.util.Set;
 3
 4  /**
 5     A program that prints hash codes of coins.
 6  */
 7  public class CoinHashCodePrinter
 8  {
 9     public static void main(String[] args)
10     {
11        Coin coin1 = new Coin(0.25, "quarter");
12        Coin coin2 = new Coin(0.25, "quarter");
13        Coin coin3 = new Coin(0.05, "nickel");
14
15        System.out.println("hash code of coin1=" + coin1.hashCode());
16        System.out.println("hash code of coin2=" + coin2.hashCode());
17        System.out.println("hash code of coin3=" + coin3.hashCode());
18
19        Set<Coin> coins = new HashSet<Coin>();
20        coins.add(coin1);
21        coins.add(coin2);
22        coins.add(coin3);
23
```

Continued

ch16/hashcode/CoinHashCodePrinter.java (cont.)

```
24          for (Coin c : coins)
25             System.out.println(c);
26       }
27    }
```

Program Run:

```
hash code of coin1=-1513525892
hash code of coin2=-1513525892
hash code of coin3=-1768365211
Coin[value=0.25,name=quarter]
Coin[value=0.05,name=nickel]
```

Self Check 16.9

What is the hash code of the string `"to"`?

Answer: 31 × 116 + 111 = 3707

- ما هو رمز تجزئة السلسلة "ب"؟
- الجواب: ٣١ × ١١٦ + ١١١ = ٣٧٠٧

Self Check 16.10

What is the hash code of `new Integer(13)`?

Answer: 13.

Binary Search Trees

- Binary search trees allow for fast insertion and removal of elements
- They are specially designed for fast searching
- **A binary tree** consists of two nodes, each of which has two child nodes
- All nodes in **a binary search tree** fulfill the property that:
  - *Descendants to the left have smaller data values than the node data value*
  - *Descendants to the right have larger data values than the node data value*

- البحث الاشجار ثنائية تسمح الإدراج السريع وإزالة العناصر
- وهي مصممة خصيصا للالبحث السريع
- تتكون شجرة ثنائية من عقدتين، كل واحدة منها لديها عقدتين الطفل
- جميع العقد في شجرة البحث الثنائية الوفاء الممتلكات التي:
- أحفاد إلى اليسار لديها قيم البيانات أصغر من قيمة البيانات العقدة
- أحفاد إلى اليمين لديها قيم البيانات أكبر من قيمة البيانات العقدة

A Binary Search Tree

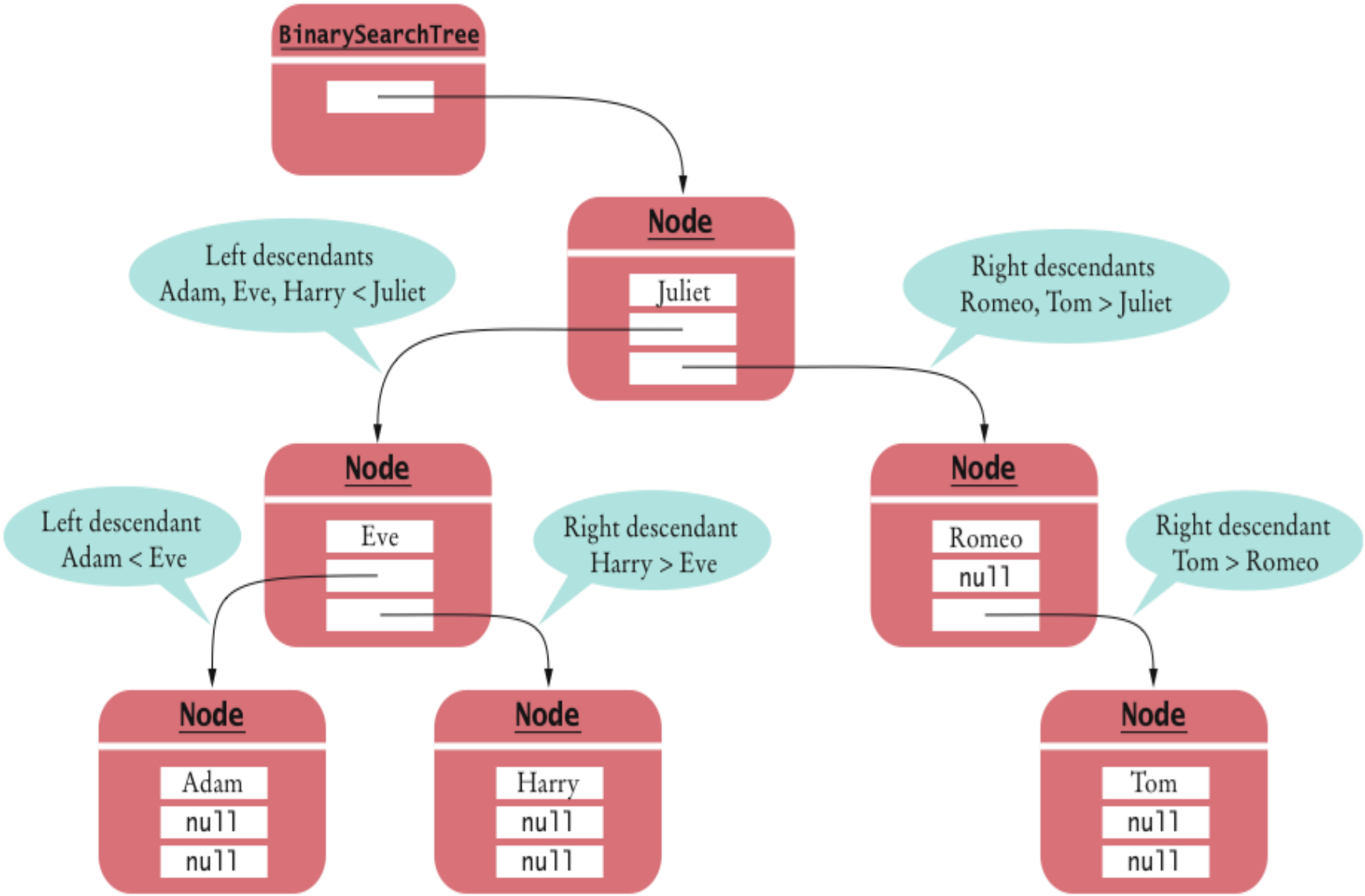


Figure 7 A Binary Search Tree

A Binary Tree That Is Not a Binary Search Tree

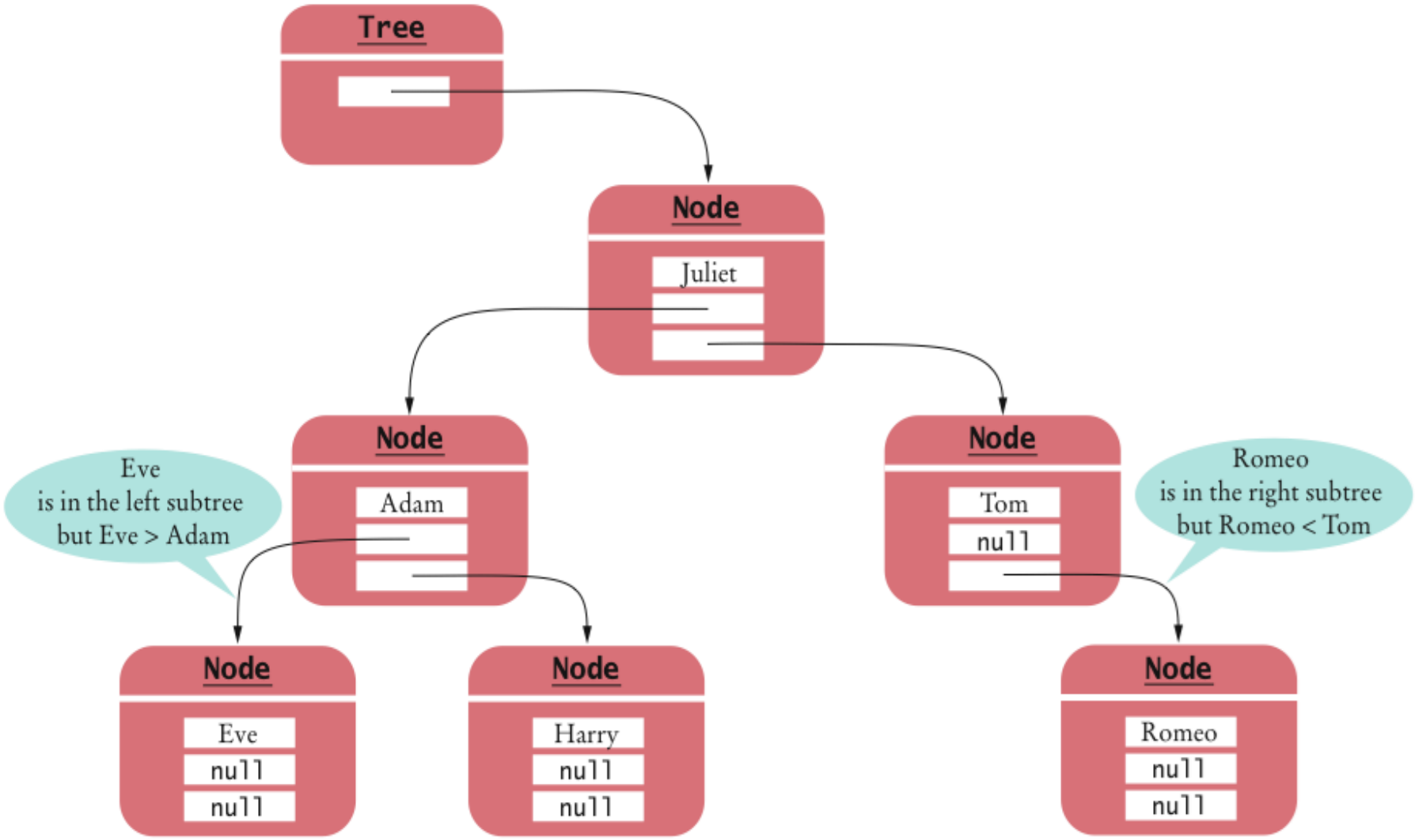


Figure 8 A Binary Tree That Is Not a Binary Search Tree

Implementing a Binary Search Tree

تنفيذ ثنائي البحث شجرة

- Implement a class for the tree containing a reference to the root node
- Implement a class for the nodes
  - *A node contains two references (to left and right child nodes)*
  - *A node contains a data field*
  - *The data field has type* `Comparable`*, so that you can compare the values in order to place them in the correct position in the binary search tree*

- تنفيذ فئة لشجرة تحتوي على مرجع إلى عقدة الجذر
- تنفيذ فئة للعقد
- عقدة يحتوي على اثنين من المراجع (على العقد التابعة اليسار واليمين)
- يحتوي على عقدة حقل بيانات
- حقل بيانات تمت كتابة قابلة للمقارنة، بحيث يمكنك مقارنة القيم من أجل وضعها في الموضع الصحيح في شجرة البحث الثنائية

Implementing a Binary Search Tree

```
public class BinarySearchTree
{
   private Node root;

   public BinarySearchTree() { ... }
   public void add(Comparable obj) { ... }
   ...
   private class Node
   {
      public Comparable data;
      public Node left;
      public Node right;

      public void addNode(Node newNode) { ... }
      ...
   }
}
```

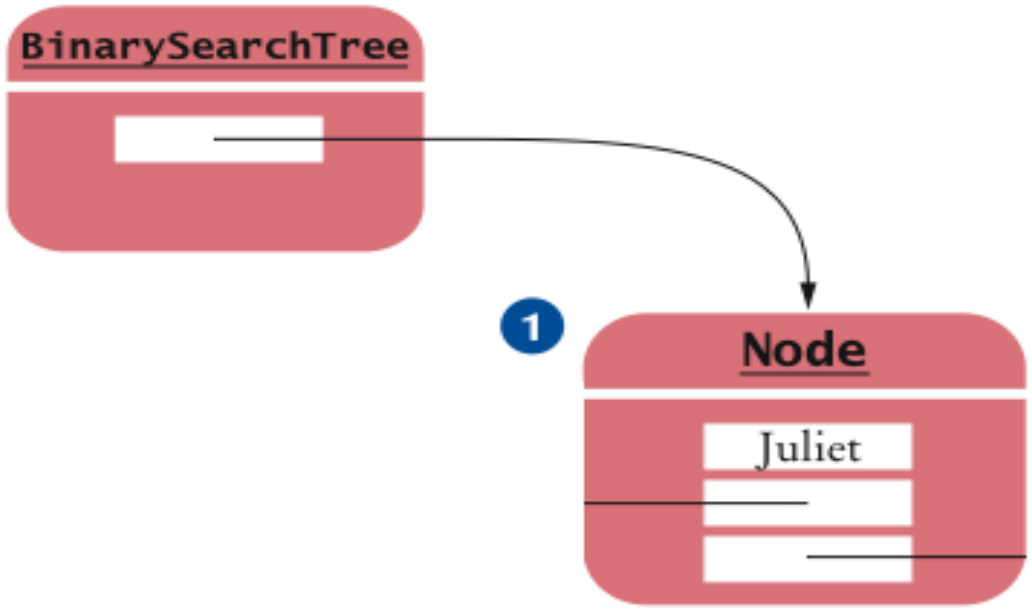
Insertion Algorithm

- If you encounter a non-`null` node reference, look at its `data` value
  - *If the `data` value of that node is larger than the one you want to insert, continue the process with the left subtree*
  - *If the existing `data` value is smaller, continue the process with the right subtree*
- If you encounter a `null` node pointer, replace it with the new node

- إذا واجهت إشارة عقدة غير فارغة، والنظر في قيمة البيانات الخاصة به
- إذا كانت قيمة البيانات من تلك العقدة أكبر من واحد كنت تريد إدراج؟ مواصلة العملية مع الشجرة اليسرى
- إذا كانت قيمة البيانات الموجودة أصغر، ومواصلة عملية مع الشجرة الفرعية الصحيحة
- إذا واجهت مؤشر عقدة لاغية، والاستعاضة عنها عقدة جديدة

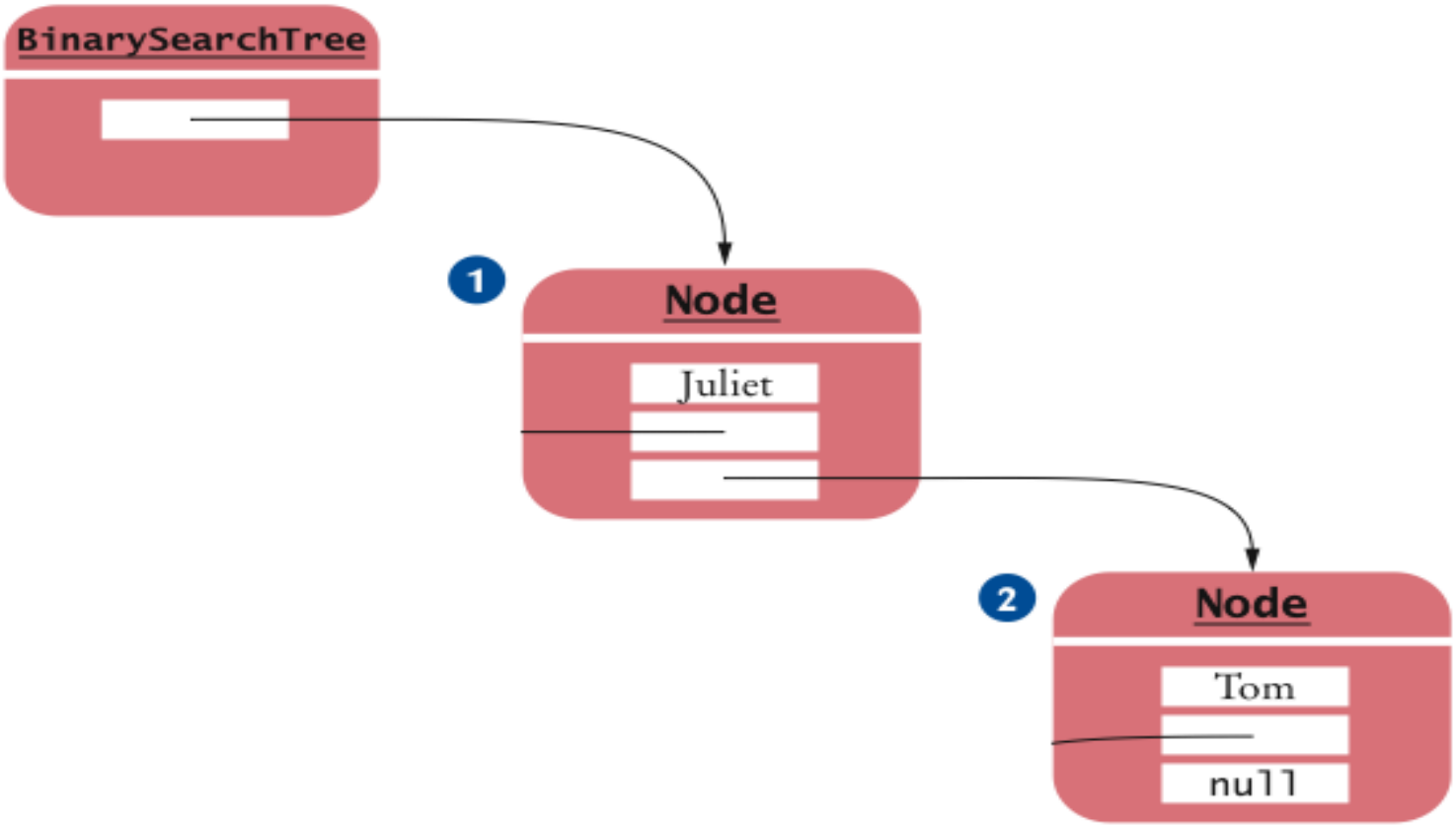
Example

```
BinarySearchTree tree = new BinarySearchTree();
tree.add("Juliet");  1
```



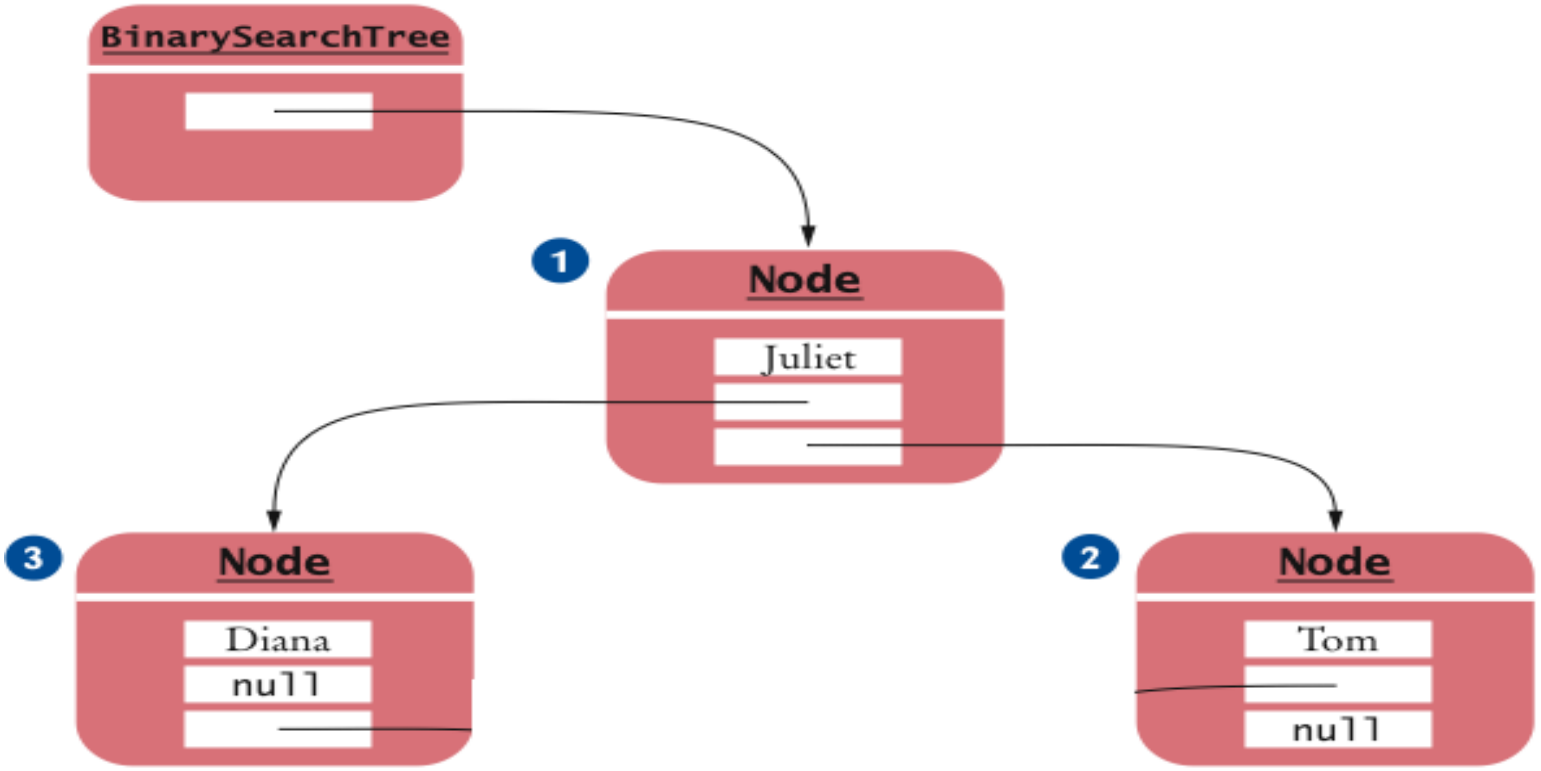
Example

```
BinarySearchTree tree = new BinarySearchTree();
tree.add("Juliet");  1
tree.add("Tom");     2
```



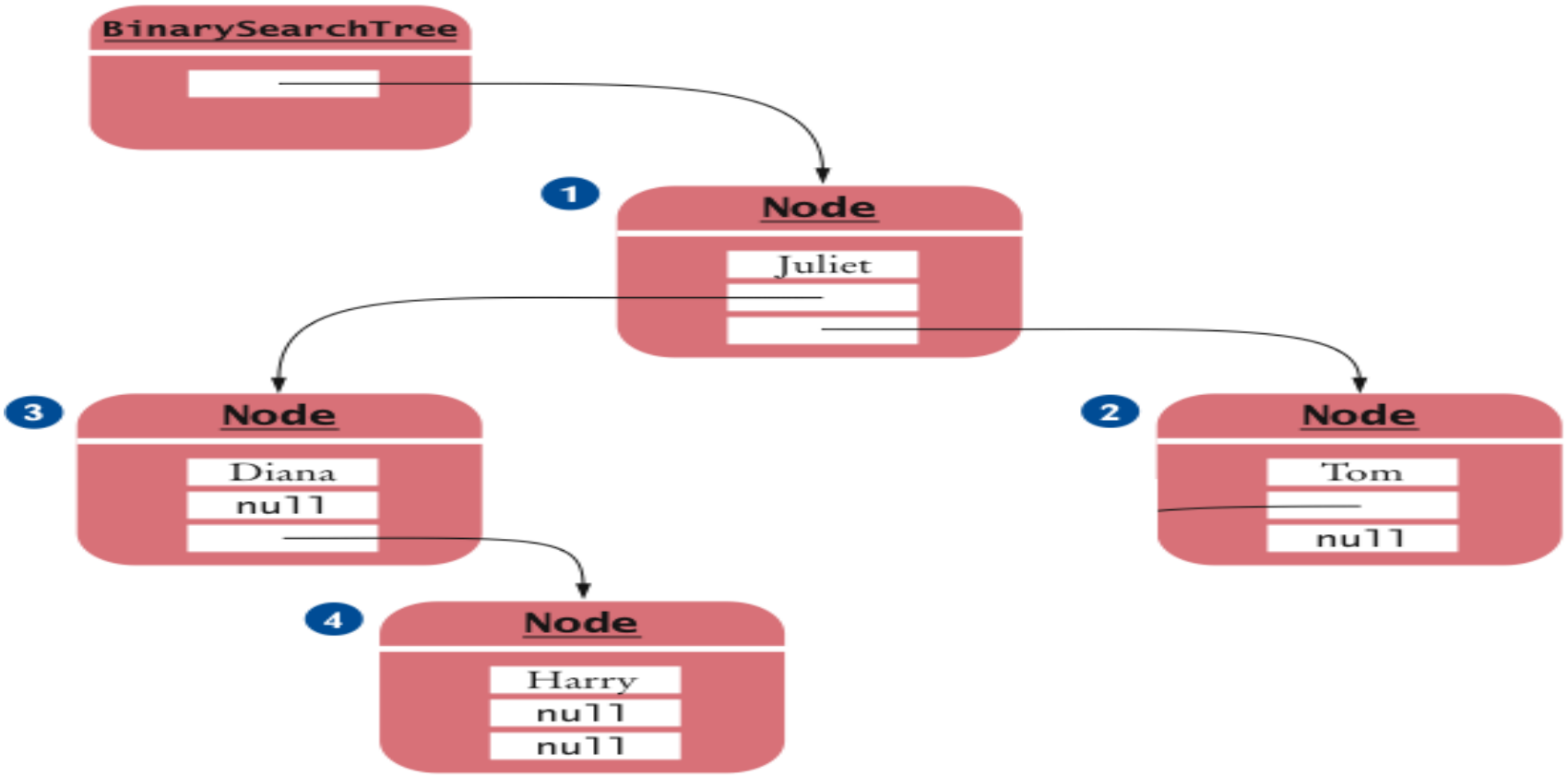
Example

- `BinarySearchTree tree = new BinarySearchTree();`
  `tree.add("Juliet");` 1
  `tree.add("Tom");` 2
  `tree.add("Diana")` 3



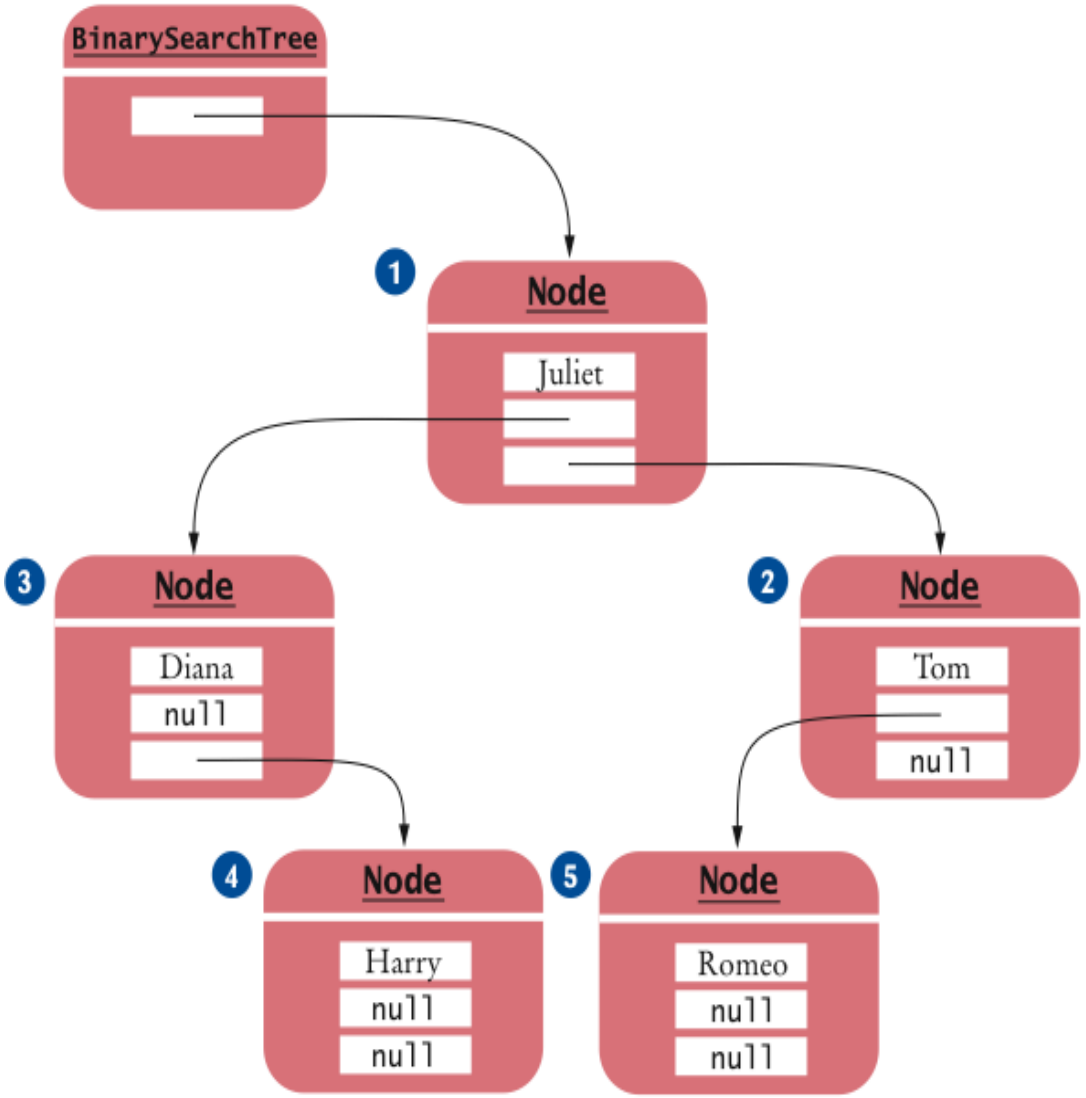
Example

- ```
  BinarySearchTree tree = new BinarySearchTree();
  tree.add("Juliet");1
  tree.add("Tom"); 2
  tree.add(”Diana")3
  tree.add("Harry"); 4
  ```


  ```

Example

- `BinarySearchTree tree = new BinarySearchTree();`
  `tree.add("Juliet");` 1
  `tree.add("Tom");` 2
  `tree.add(”Diana")` 3
  `tree.add("Harry");` 4
  `tree.add("Romeo");` 5



`add` Method of the `BinarySearchTree` Class

```
public void add(Comparable obj)
{
   Node newNode = new Node();
   newNode.data = obj;
   newNode.left = null;
   newNode.right = null;
   if (root == null) root = newNode;
   else root.addNode(newNode);
}
```

addNode Method of the Node Class

```
private class Node
{
   ...
   public void addNode(Node newNode)
   {
      int comp = newNode.data.compareTo(data);
      if (comp < 0)
      {
         if (left == null) left = newNode;
         else left.addNode(newNode);
      }
   else if (comp > 0)
   {
      if (right == null) right = newNode;
      else right.addNode(newNode);
      }
   }
   ...
}
```

Binary Search Trees

- When removing a node with only one child, the child replaces the node to be removed
- When removing a node with two children, replace it with the smallest node of the right subtree

- عند إزالة عقدة مع طفل واحد فقط، والطفل محل العقدة المراد إزالتها
- عند إزالة عقدة مع اثنين من الأطفال، والاستعاضة عنها أصغر عقدة من الشجرة الفرعية الصحيحة

Removing a Node with One Child

إزالة عقدة مع طفل واحد

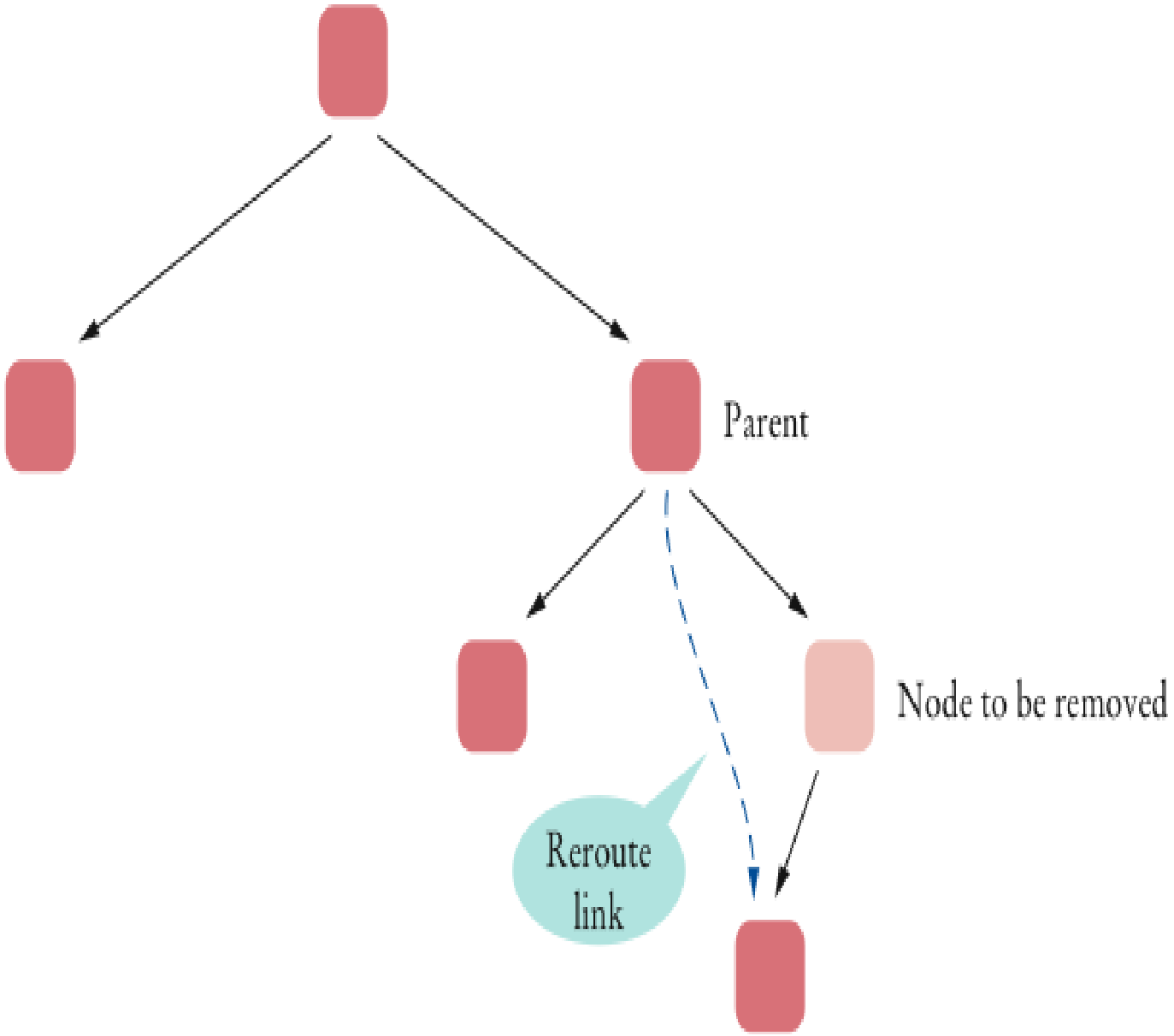


Figure 11
Removing a Node with One Child

Removing a Node with Two Children

إزالة عقدة مع اثنين من الأطفال

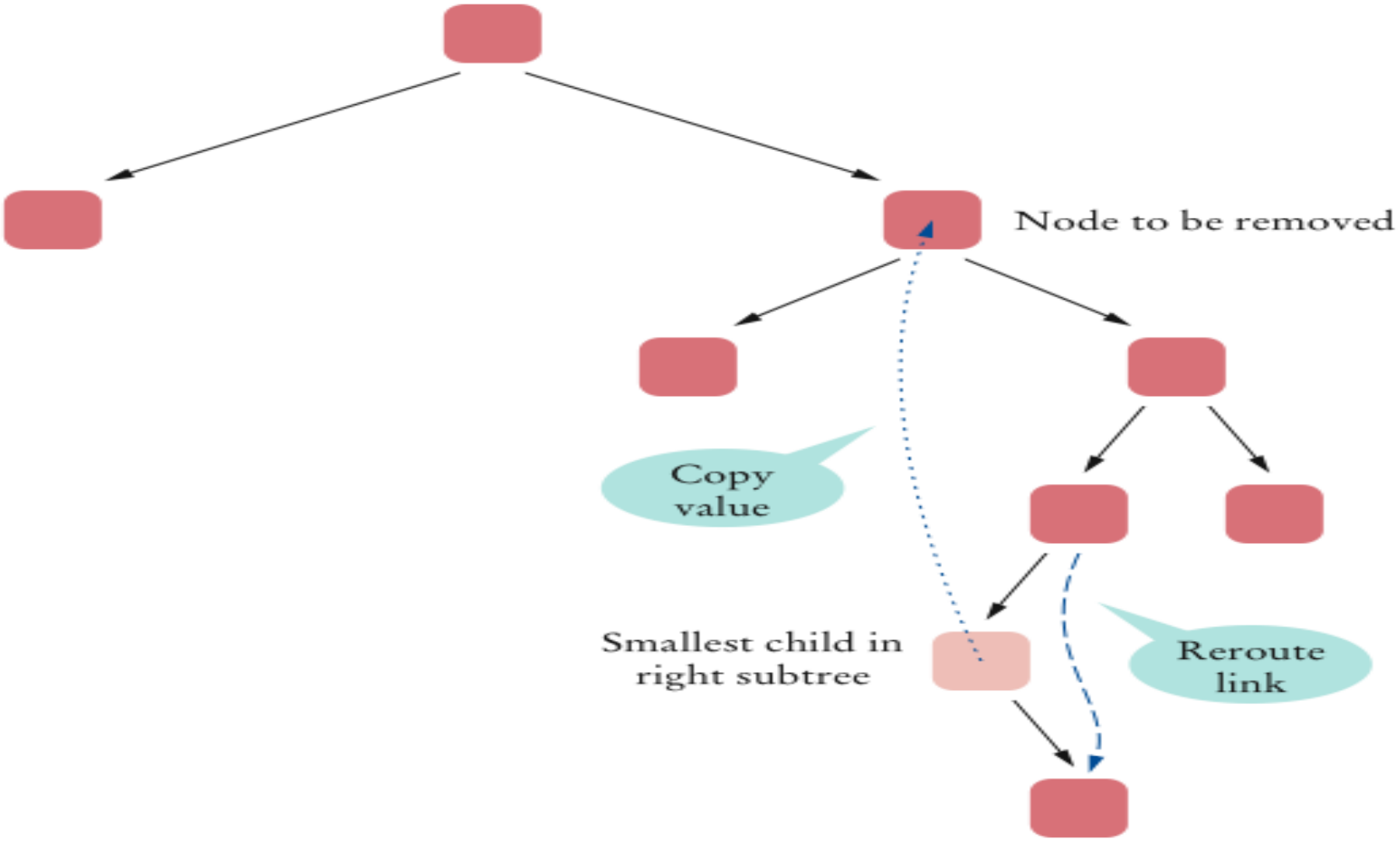


Figure 12 Removing a Node with Two Children

Binary Search Trees

- **Balanced tree:** each node has approximately as many descendants on the left as on the right
- If a binary search tree is balanced, then adding an element takes $O(\log(n))$ time
- If the tree is unbalanced, insertion can be slow
  - *Perhaps as slow as insertion into a linked list*

- شجرة متوازنة: كل عقدة لديه ما يقرب من أكبر عدد ممكن من أحفاد على اليسار كما على اليمين
- إذا شجرة البحث الثنائية متوازنة، ثم إضافة عنصر يأخذ $O(\log(n))$ وقت
- إذا كانت الشجرة غير متوازنة، ويمكن أن تكون بطيئة الإدراج
- ربما بطيئة مثل الإدراج في قائمة مرتبطة

Balanced and Unbalanced Trees

الأشجار المتوازنة وغير المتوازنة

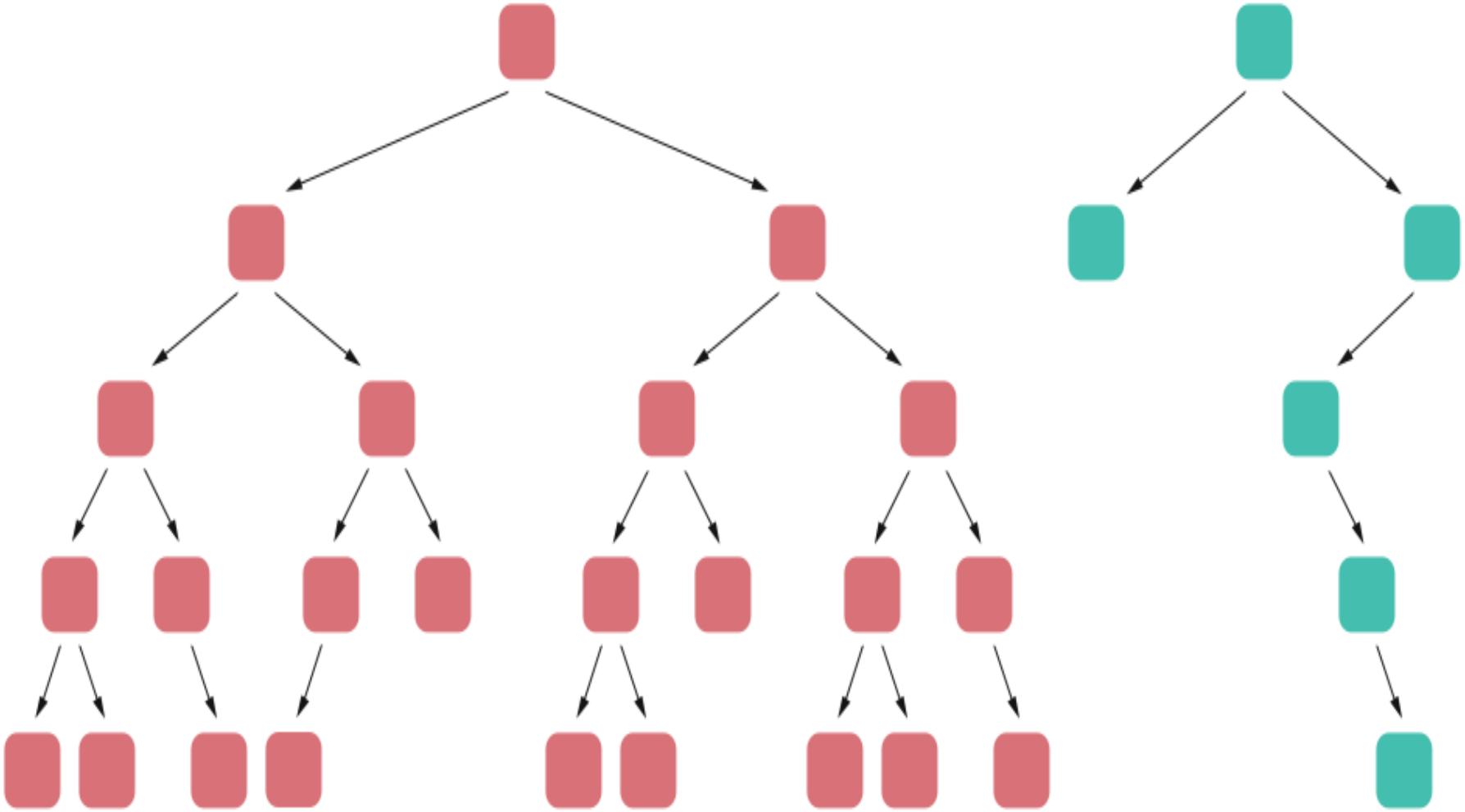


Figure 13 Balanced and Unbalanced Trees

ch16/tree/BinarySearchTree.java

```
1   /**
2      This class implements a binary search tree whose
3      nodes hold objects that implement the Comparable
4      interface.
5   */
6   public class BinarySearchTree
7   {
8      private Node root;
9
10     /**
11        Constructs an empty tree.
12     */
13     public BinarySearchTree()
14     {
15        root = null;
16     }
17
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
18      /**
19         Inserts a new node into the tree.
20         @param obj the object to insert
21      */
22      public void add(Comparable obj)
23      {
24         Node newNode = new Node();
25         newNode.data = obj;
26         newNode.left = null;
27         newNode.right = null;
28         if (root == null) root = newNode;
29         else root.addNode(newNode);
30      }
31
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
32      /**
33         Tries to find an object in the tree.
34         @param obj the object to find
35         @return true if the object is contained in the tree
36      */
37      public boolean find(Comparable obj)
38      {
39         Node current = root;
40         while (current != null)
41         {
42            int d = current.data.compareTo(obj);
43            if (d == 0) return true;
44            else if (d > 0) current = current.left;
45            else current = current.right;
46         }
47         return false;
48      }
49
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
50      /**
51         Tries to remove an object from the tree. Does nothing
52         if the object is not contained in the tree.
53         @param obj the object to remove
54      */
55      public void remove(Comparable obj)
56      {
57         // Find node to be removed
58
59         Node toBeRemoved = root;
60         Node parent = null;
61         boolean found = false;
62         while (!found && toBeRemoved != null)
63         {
64            int d = toBeRemoved.data.compareTo(obj);
65            if (d == 0) found = true;
66            else
67            {
68               parent = toBeRemoved;
69               if (d > 0) toBeRemoved = toBeRemoved.left;
70               else toBeRemoved = toBeRemoved.right;
71            }
72         }
73
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
74          if (!found) return;
75
76          // toBeRemoved contains obj
77
78          // If one of the children is empty, use the other
79
80          if (toBeRemoved.left == null || toBeRemoved.right == null)
81          {
82             Node newChild;
83             if (toBeRemoved.left == null)
84                newChild = toBeRemoved.right;
85             else
86                newChild = toBeRemoved.left;
87
88             if (parent == null) // Found in root
89                root = newChild;
90             else if (parent.left == toBeRemoved)
91                parent.left = newChild;
92             else
93                parent.right = newChild;
94             return;
95          }
96
97          // Neither subtree is empty
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
 98
 99          // Find smallest element of the right subtree
100
101          Node smallestParent = toBeRemoved;
102          Node smallest = toBeRemoved.right;
103          while (smallest.left != null)
104          {
105             smallestParent = smallest;
106             smallest = smallest.left;
107          }
108
109          // smallest contains smallest child in right subtree
110
111          // Move contents, unlink child
112
113          toBeRemoved.data = smallest.data;
114          if (smallestParent == toBeRemoved)
115             smallestParent.right = smallest.right;
116          else
117             smallestParent.left = smallest.right;
118       }
119
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
120     /**
121        Prints the contents of the tree in sorted order.
122     */
123     public void print()
124     {
125        if (root != null)
126           root.printNodes();
127        System.out.println();
128     }
129
130     /**
131        A node of a tree stores a data item and references
132        of the child nodes to the left and to the right.
133     */
134     class Node
135     {
136        public Comparable data;
137        public Node left;
138        public Node right;
139
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
140          /**
141             Inserts a new node as a descendant of this node.
142             @param newNode the node to insert
143          */
144          public void addNode(Node newNode)
145          {
146             int comp = newNode.data.compareTo(data);
147             if (comp < 0)
148             {
149                if (left == null) left = newNode;
150                else left.addNode(newNode);
151             }
152             else if (comp > 0)
153             {
154                if (right == null) right = newNode;
155                else right.addNode(newNode);
156             }
157          }
158
```

Continued

ch16/tree/BinarySearchTree.java (cont.)

```
159          /**
160             Prints this node and all of its descendants
161             in sorted order.
162          */
163          public void printNodes()
164          {
165             if (left != null)
166                left.printNodes();
167             System.out.print(data + " ");
168             if (right != null)
169                right.printNodes();
170          }
171       }
172    }
```

Self Check 16.11

What is the difference between a tree, a binary tree, and a balanced binary tree?

Answer: In a tree, each node can have any number of children. In a binary tree, a node has at most two children. In a balanced binary tree, all nodes have approximately as many descendants to the left as to the right.

- ما هو الفرق بين شجرة، شجرة الثنائية، وشجرة ثنائية متوازنة؟
- الجواب: في شجرة، ويمكن لكل عقدة يكون أي عدد من الأطفال. في شجرة ثنائية، عقدة اثنين على الأكثر الأطفال. في شجرة ثنائية متوازنة، كافة العقد لديها ما يقرب من أكبر عدد ممكن من نسل إلى اليسار فيما يتعلق بحق.

Self Check 16.12

Give an example of a string that, when inserted into the tree of Figure 10, becomes a right child of `Romeo`.

Answer: For example, `Sarah`. Any string between `Romeo` and `Tom` will do.

- تعطي مثالا على سلسلة، عند إدراجها في شجرة الشكل ١٠، ويصبح الطفل الأيمن من روميو.
- الجواب: على سبيل المثال، سارة. وأي سلسلة بين روميو وتوم القيام به.

Binary Tree Traversal

- Print the tree elements in sorted order:
  1. *Print the left subtree*
  2. *Print the data*
  3. *Print the right subtree*

- طباعة العناصر شجرة في ترتيب فرزها:
- طباعة الشجرة الفرعية اليسار
- طباعة البيانات
- طباعة الشجرة الفرعية الصحيحة

Example

- Let's try this out with the tree in Figure 10. The algorithm tells us to
  1. *Print the left subtree of* `Juliet`*; that is,* `Diana` *and descendants*
  2. *Print* `Juliet`
  3. *Print the right subtree of* `Juliet`*; that is,* `Tom` *and descendants*
- How do you print the subtree starting at `Diana`?
  1. *Print the left subtree of* `Diana`*. There is nothing to print*
  2. *Print* `Diana`
  3. *Print the right subtree of* `Diana`*, that is,* `Harry`

- دعونا محاولة ذلك مع شجرة في الشكل ١٠. الخوارزمية يخبرنا ل
- طباعة الشجرة الفرعية تبقى من جولييت. وهذا هو، ديانا وأحفاد
- طباعة جولييت
- طباعة الشجرة الفرعية الصحيحة جولييت؛ وهذا هو، توم وأحفاد
- كيف يمكنك طباعة الشجرة الفرعية ابتداء من الساعة ديانا؟
- طباعة الشجرة الفرعية تبقى من ديانا. لا يوجد شيء للطباعة
- طباعة ديانا
- طباعة الشجرة الفرعية الصحيحة من ديانا، وهذا هو، هاري

Example

- Algorithm goes on as above
- **Output:**

```
Diana Harry Juliet Romeo Tom
```

- The tree is printed in sorted order

- خوارزمية يمضي على النحو الوارد أعلاه
- الإخراج:
- ديانا هاري جولييت روميو توم
- طبعت الشجرة في ترتيب فرزها

Node Class `printNodes` Method

Worker method:

```
private class Node
{
   ...
   public void printNodes()
   {
      if (left != null)
         left.printNodes();
      System.out.println(data);
      if (right != null)
         right.printNodes();
   }
   ...
}
```

BinarySearchTree Class print Method

To print the entire tree, start this recursive printing process at the root:

- لطباعة الشجرة بأكملها، تبدأ هذه العملية الطباعة متكررة في الجذر:

```
public class BinarySearchTree
{
   ...
   public void print()
   {
      if (root != null)
         root.printNodes();
      System.out.println();
   }
   ...
}
```

Tree Traversal

- Tree traversal schemes include
  - *Preorder traversal*
  - *Inorder traversal*
  - *Postorder traversal*

- وتشمل خطط شجرة اجتياز
- بلاي اجتياز
- اتباعها اجتياز
- Postorderاجتياز

Preorder Traversal

بريوردير الانتقالي

- Visit the root
- Visit the left subtree
- Visit the right subtree

- زيارة الجذر
- زيارة الشجرة اليسار
- زيارة الشجرة الفرعية الصحيحة

Inorder Traversal

- Visit the left subtree
- Visit the root
- Visit the right subtree

- زيارة الشجرة اليسار
- زيارة الجذر
- زيارة الشجرة الفرعية الصحيحة

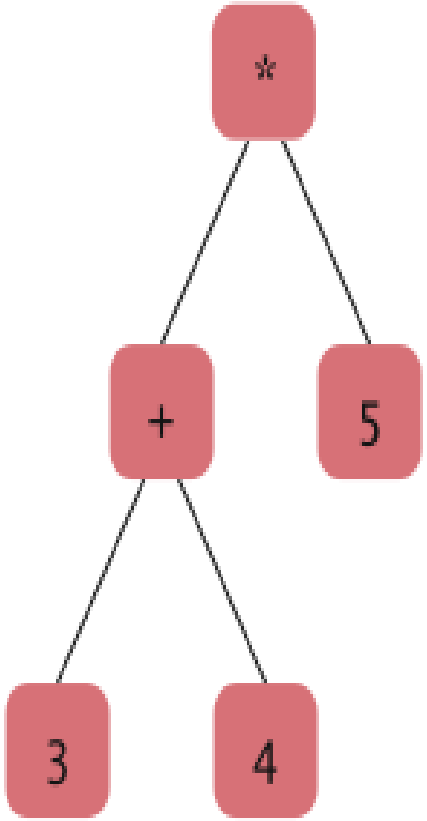
Postorder Traversal

- Visit the left subtree
- Visit the right subtree
- Visit the root

- زيارة الشجرة اليسار
- زيارة الشجرة الفرعية الصحيحة
- زيارة الجذر

Tree Traversal

- Postorder traversal of an expression tree yields the instructions for evaluating the expression on a stack-based calculator



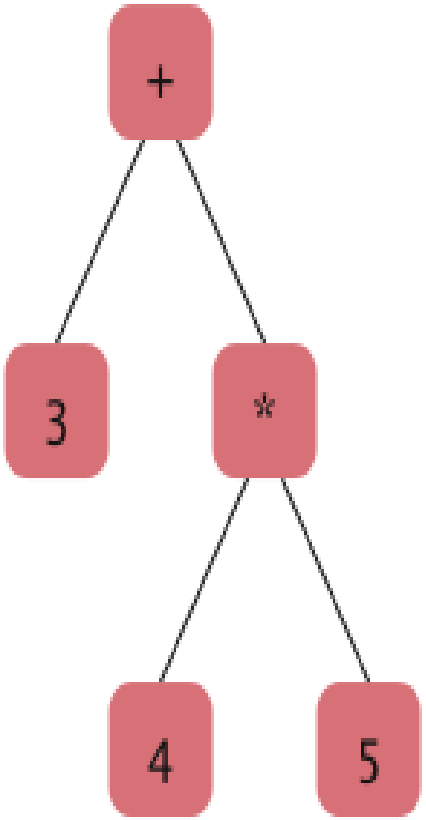


Figure 14 Expression Trees

- Postorderاجتياز شجرة التعبير غلة الإرشادات لتقييم التعبير على الآلة الحاسبة القائم على كومة

Tree Traversal

- The first tree (`(3 + 4) * 5`) yields

```
3 4 + 5 *
```

- Whereas the second tree (`3 + 4 * 5`) yields

```
3 4 5 * +
```

Self Check 16.13

What are the inorder traversals of the two trees in Figure 14?

Answer: For both trees, the inorder traversal is `3 + 4 * 5`.

- ما هي traversalsاتباعها من الشجرتين في الشكل ١٤؟
- الجواب: بالنسبة لكل من الأشجار، واجتياز اتباعها هو ٣ + ٤ * ٥.

Self Check 16.14

Are the trees in Figure 14 binary search trees?

Answer: No — for example, consider the children of `+`. Even without looking up the Unicode codes for `3`, `4`, and `+`, it is obvious that `+` isn't between `3` and `4`.

- هي شجرة في الشكل ١٤ أشجار البحث الثنائية؟
- الجواب: لا - على سبيل المثال، والنظر في بني +. حتى من دون البحث عن رموز يونيكود لمدة ٣ و ٤ و +، فمن الواضح أن + ليس بين ٣ و ٤.

Priority Queues

قوائم انتظار الأولوية

- **A priority queue** :collects elements, each of which has a priority
- Example: Collection of work requests, some of which may be more urgent than others
- When removing an element, element with highest priority is retrieved
  - *Customary to give low values to high priorities, with priority 1 denoting the highest priority*
- Standard Java library supplies a `PriorityQueue` class
- A data structure called *heap* is very suitable for implementing priority queues

- طابور الأولوية: يجمع العناصر، كل منها له الأولوية
- مثال: مجموعة من طلبات العمل، وبعضها قد يكون أكثر إلحاحا من غيرها
- عند إزالة عنصر، يتم استرداد العنصر مع أولوية قصوى
- العادة لإعطاء القيم المنخفضة للأولويات عالية، مع إعطاء الأولوية ١ تدل على الأولوية القصوى
- وازم المكتبات القياسية جافا فئة PriorityQueue
- هيكل بيانات تسمى كومة هو مناسب جدا لتنفيذ طوابير الأولوية

Example

- Consider this sample code:

النظر في هذه التعليمات البرمجية:

```
PriorityQueue<WorkOrder> q =
     new PriorityQueue<WorkOrder>;
q.add(new WorkOrder(3, "Shampoo carpets"));
q.add(new WorkOrder(1, "Fix overflowing sink"));
q.add(new WorkOrder(2, "Order cleaning supplies"));
```

- When calling `q.remove()` for the first time, the work order with priority 1 is removed
- Next call to `q.remove()` removes the order with priority 2

- عند استدعاء q.remove () للمرة الأولى، تتم إزالة من أجل العمل مع إعطاء الأولوية ١
- المكالمة التالية إلى q.remove () يزيل النظام مع الأولوية ٢

Heaps

- A **heap** (or, a *min-heap*) is a binary tree with two special properties
  1. *It is almost complete*
     - All nodes are filled in, except the last level may have some nodes missing toward the right
  2. *The tree fulfills the heap property*
     - All nodes store values that are at most as large as the values stored in their descendants
- Heap property ensures that the smallest element is stored in the root

- كومة (أو، على كومة دقيقة) هي شجرة ثنائية مع اثنين من الممتلكات الخاصة
- فمن شبه كامل
- تمتلئ كافة العقد في ما عدا المستوى الأخير قد يكون بعض العقد في عداد المفقودين نحو اليمين
- شجرة تلبي الممتلكات هيب
- كل القيم مخزن العقد التي هي في معظم كبيرة مثل القيم المخزنة في ذريتهم
- تضمن الملكية كومة أن أصغر عنصر يتم تخزين في الجذر

An Almost Complete Tree

شجرة كاملة تقريبا

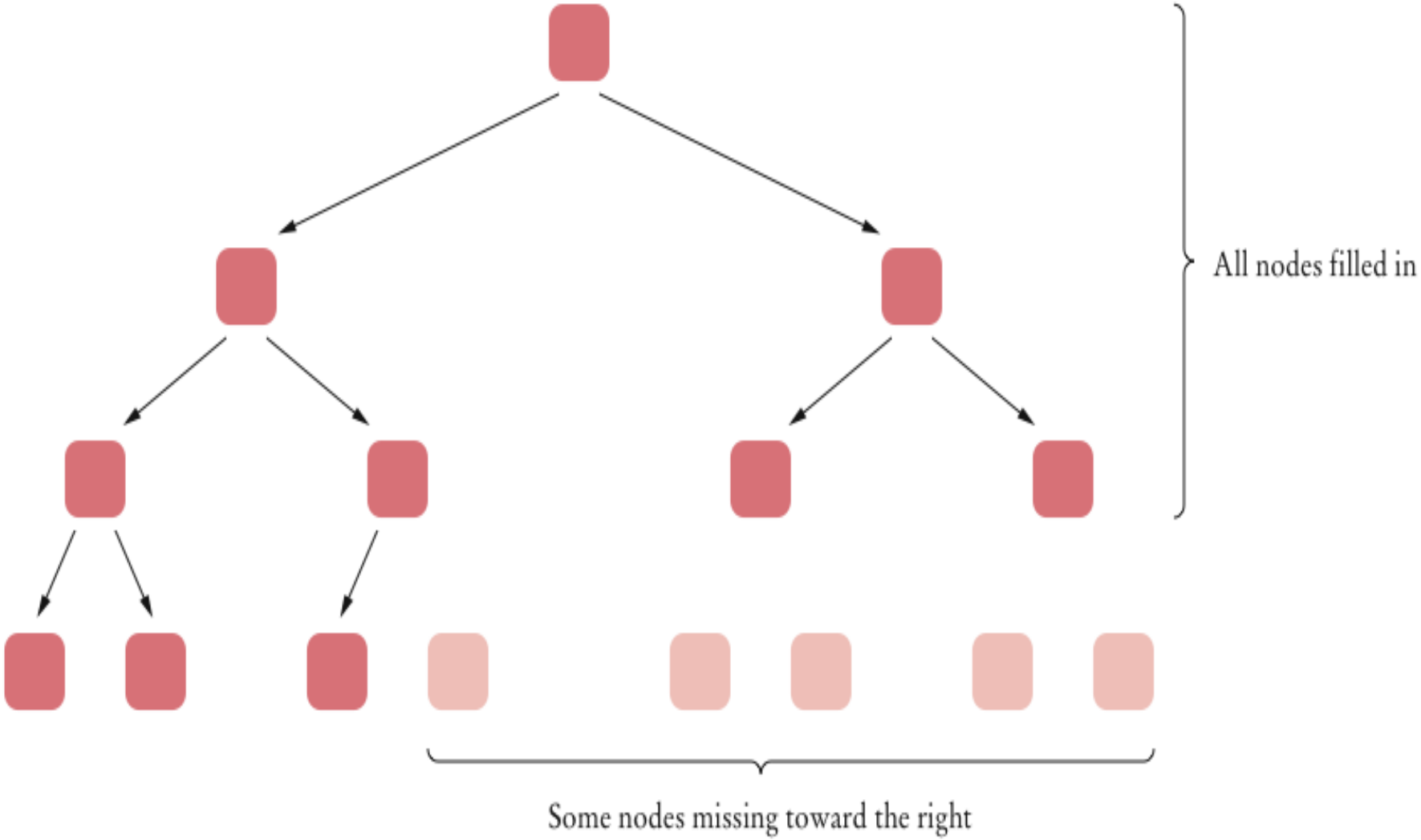
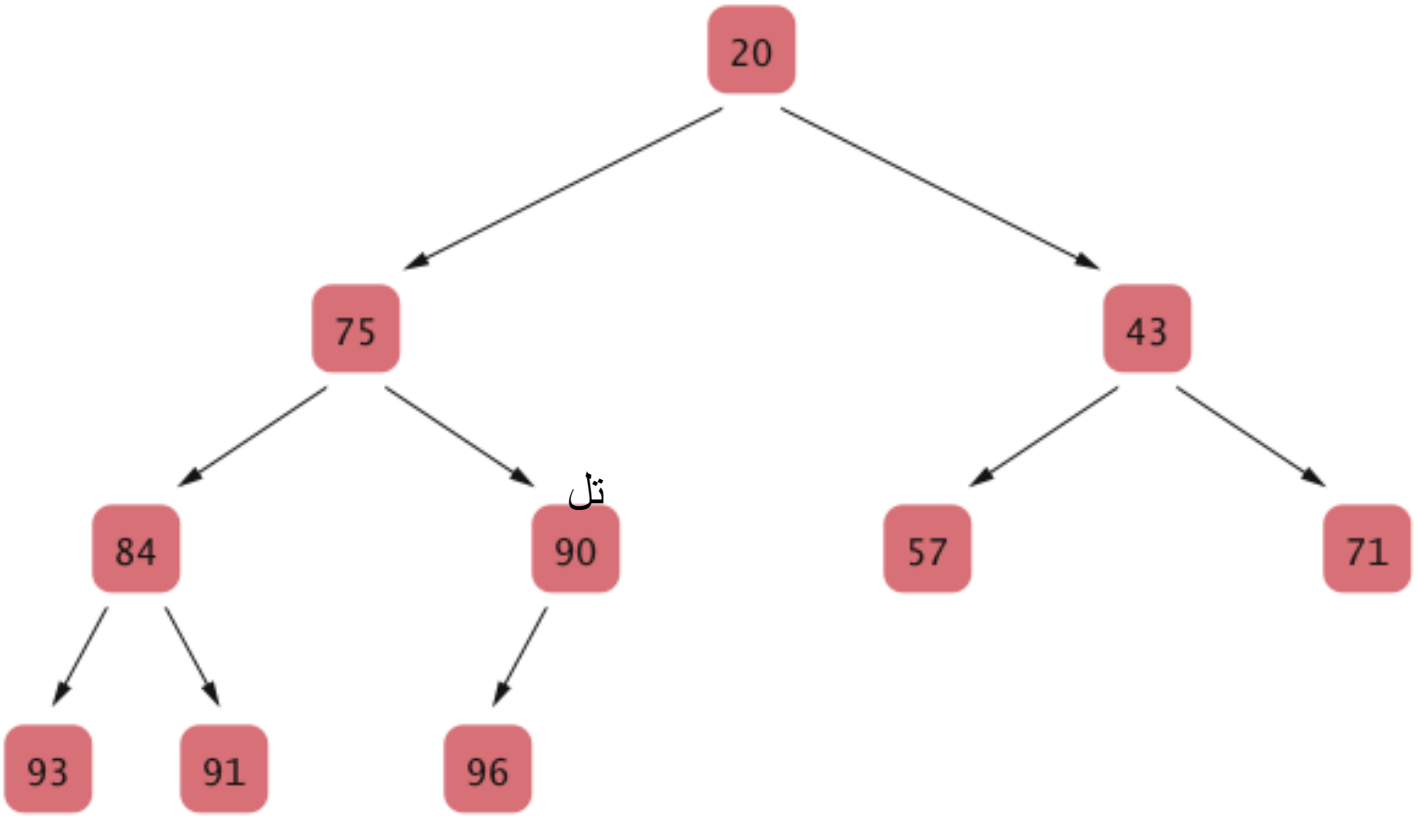


Figure 15 An Almost Completely Filled Tree

A Heap

تل

Figure 16
A Heap



Differences of a Heap with a Binary Search Tree

خلافات في كومة مع ثنائي البحث شجرة

- The shape of a heap is very regular
  - *Binary search trees can have arbitrary shapes*
- In a heap, the left and right subtrees both store elements that are larger than the root element
  - *In a binary search tree, smaller elements are stored in the left subtree and larger elements are stored in the right subtree*

- شكل كومة غير منتظمة جدا
- يمكن البحث الاشجار الثنائية لديها الأشكال التعسفية
- في كومة، اليسار واليمين على حد سواء الأشجار الفرعية عناصر مخزن التي تكون أكبر من العنصر الجذر
- في شجرة البحث الثنائية، يتم تخزين العناصر الصغيرة في الشجرة الفرعية اليسرى ويتم تخزين العناصر أكبر في الشجرة الفرعية الصحيحة

Inserting a New Element in a Heap

إدراج عنصر جديد في كومة

1. Add a vacant slot to the end of the tree

إضافة فتحة الشاغرة حتى نهاية الشجرة

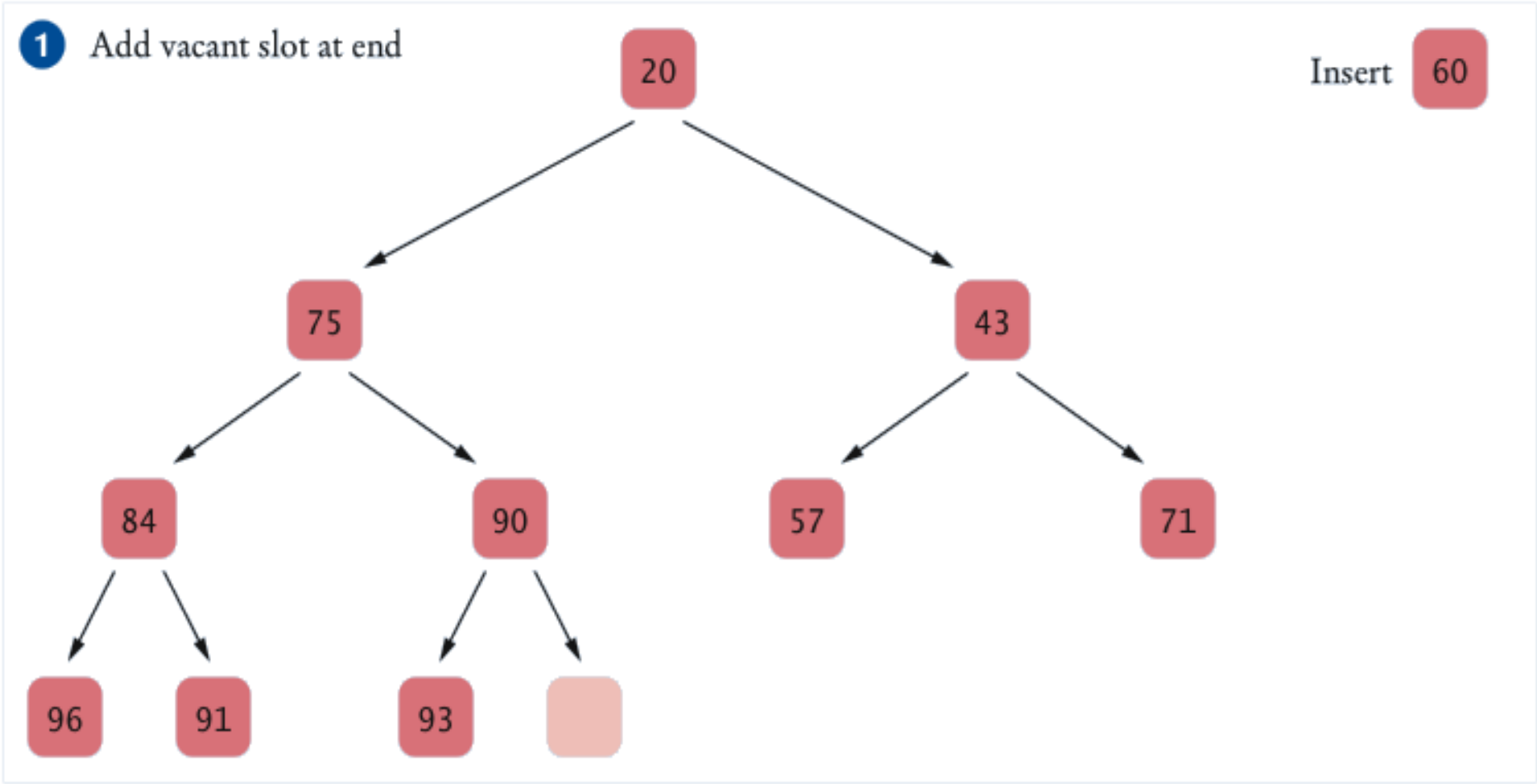


Figure 17 Inserting an Element into a Heap

Continued

Inserting a New Element in a Heap (cont.)

2. Demote the parent of the empty slot if it is larger than the element to be inserted
   - *Move the parent value into the vacant slot, and move the vacant slot up*
   - *Repeat this demotion as long as the parent of the vacant slot is larger than the element to be inserted*

- تخفيض الوالد من الفتحة الفارغة إذا كان أكبر من العنصر المراد إدراجها
- تحريك قيمة الأم في فتحة الشاغرة، ونقل فتحة شاغرا حتى
- كرر هذا التخفيض طالما أن الأم فتحة الشاغرة أكبر من العنصر المراد إدراجها

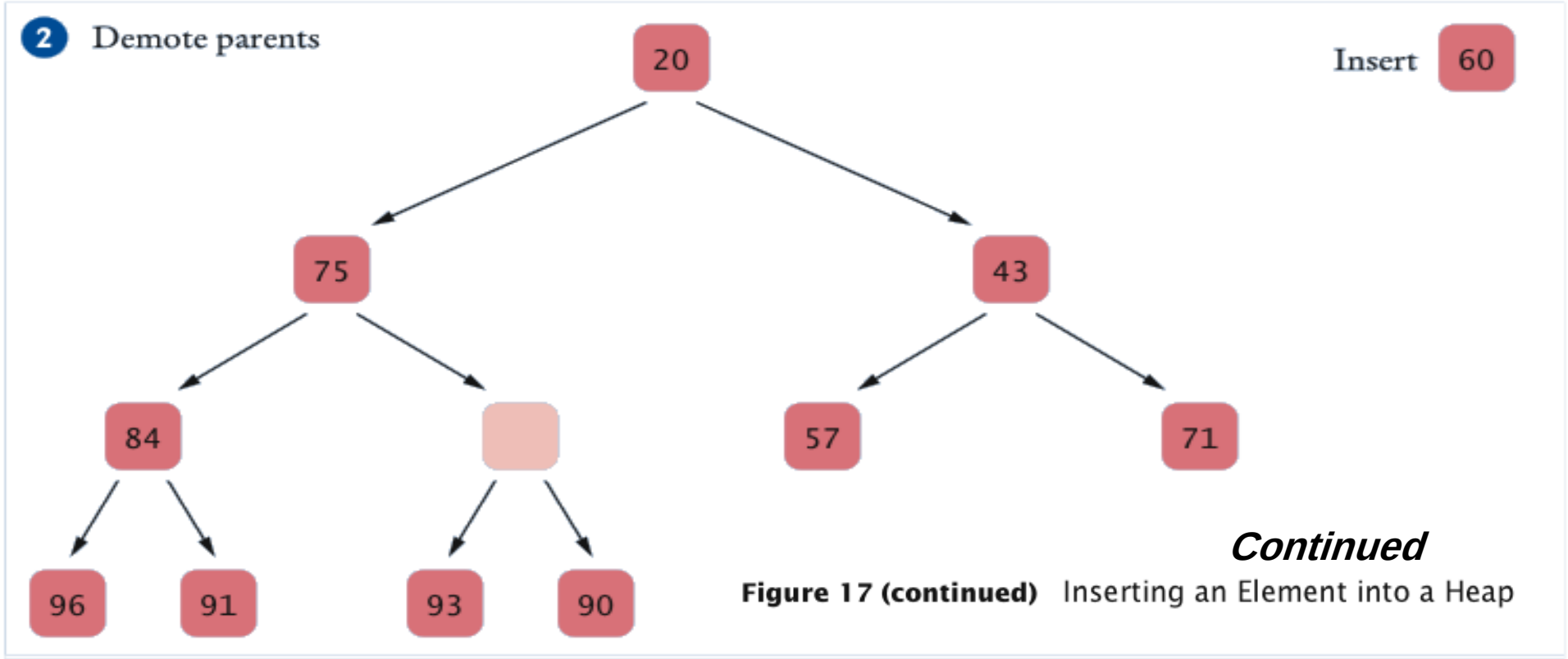
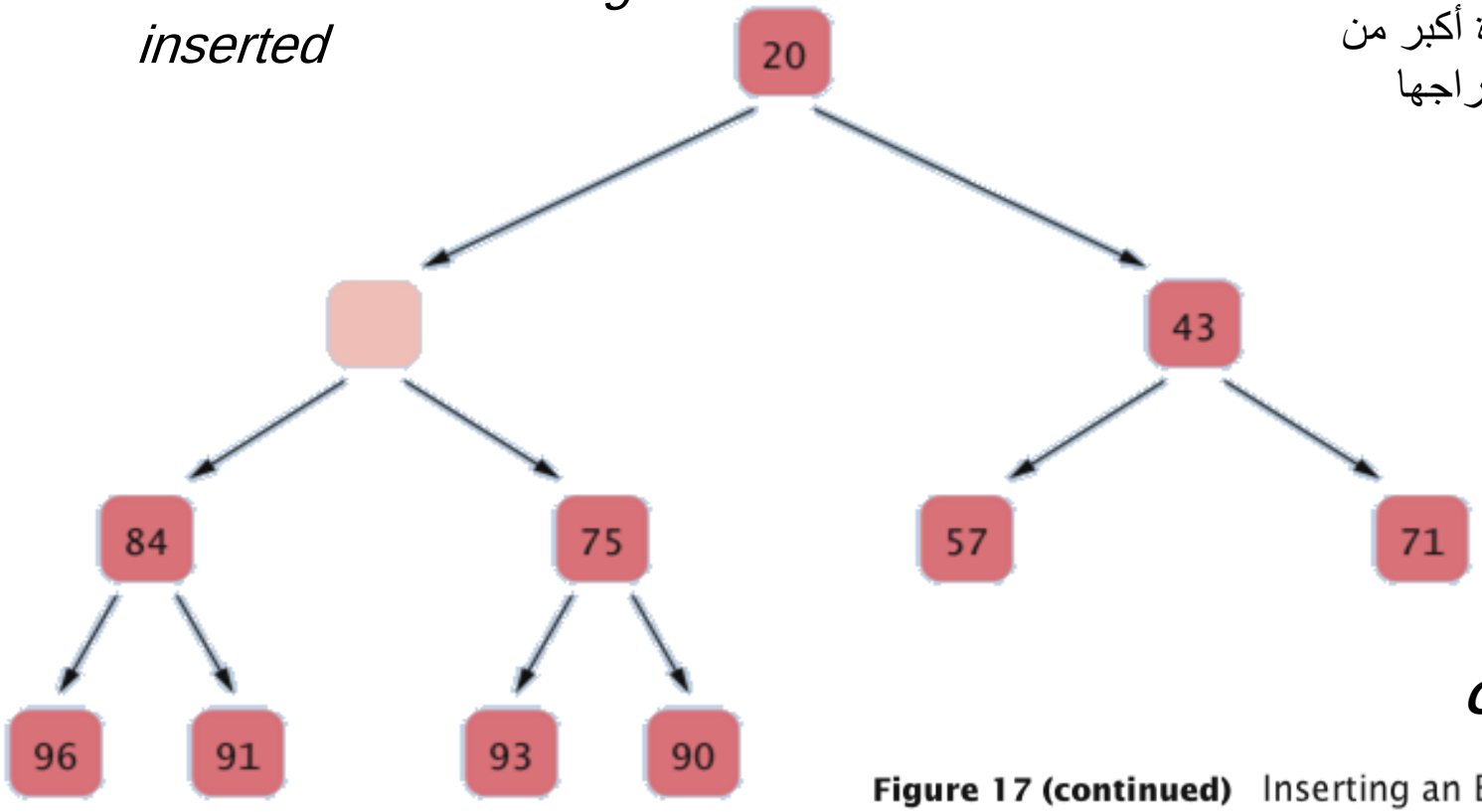


Figure 17 (continued) Inserting an Element into a Heap

Inserting a New Element in a Heap (cont.)

2. Demote the parent of the empty slot if it is larger than the element to be inserted
   - *Move the parent value into the vacant slot, and move the vacant slot up*
   - *Repeat this demotion as long as the parent of the vacant slot is larger than the element to be inserted*

- تخفيض الوالد من الفتحة الفارغة إذا كان أكبر من العنصر المراد إدراجها
- تحريك قيمة الأم في فتحة الشاغرة، ونقل فتحة شاغرا حتى
- كرر هذا التخفيض طالما أن الأم فتحة الشاغرة أكبر من العنصر المراد إدراجها



Continued

Figure 17 (continued) Inserting an Element into a Heap

Inserting a New Element in a Heap (cont.)

3. At this point, either the vacant slot is at the root, or the parent of the vacant slot is smaller than the element to be inserted. Insert the element into the vacant slot

- عند هذه النقطة، إما فتحة الشاغرة هي في الجذر، أو الوالد من فتحة الشاغرة أصغر من عنصر لاحقا. إدراج عنصر في فتحة الشاغرة

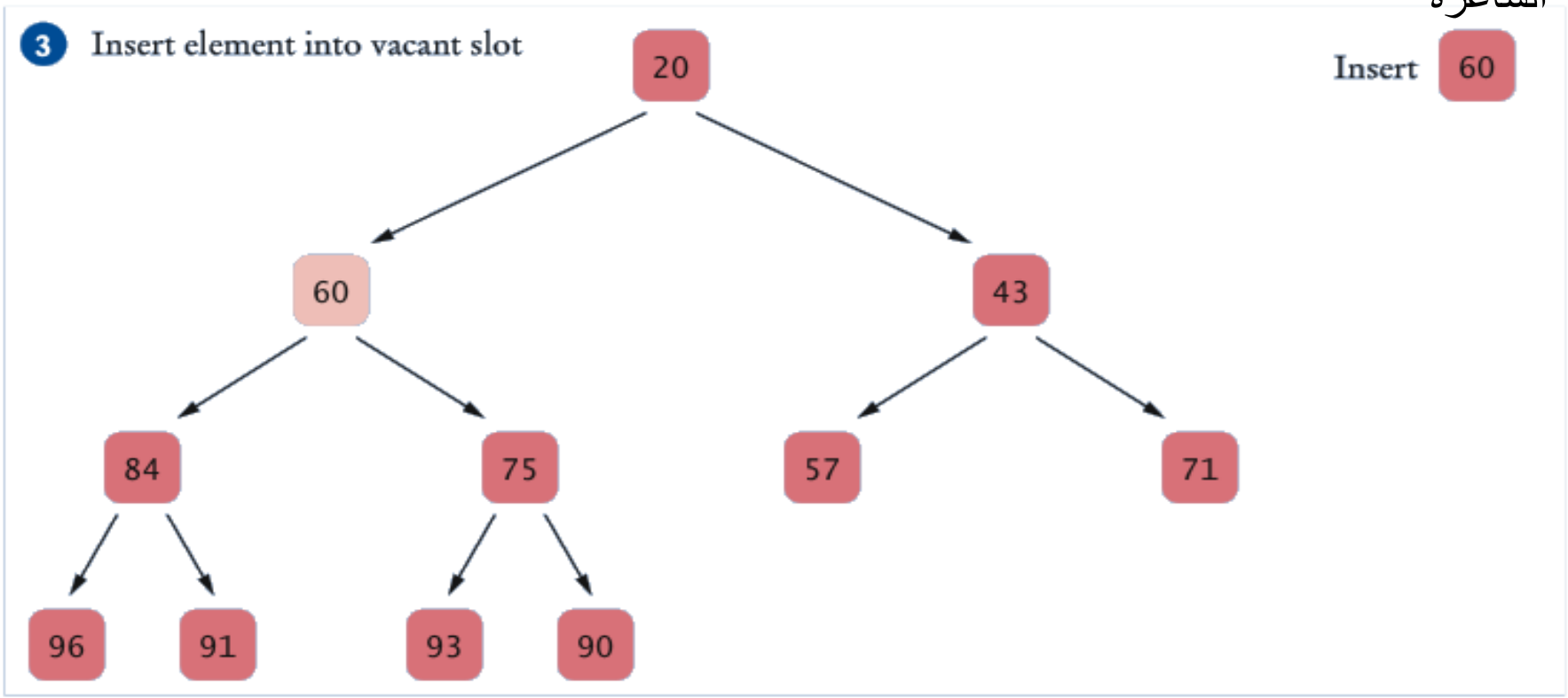
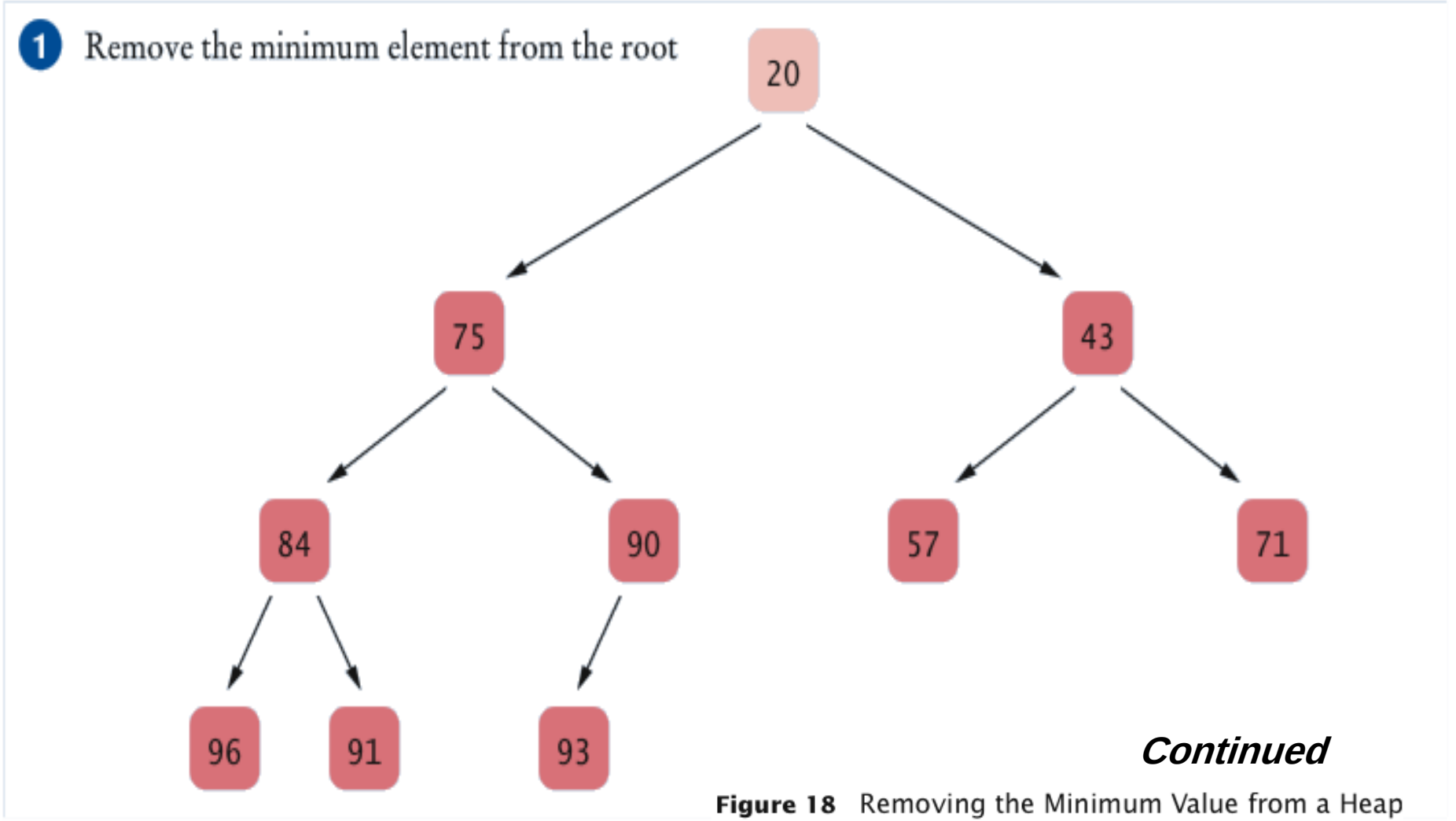


Figure 17 (continued) Inserting an Element into a Heap

Removing an Arbitrary Node from a Heap إزالة عقدة التعسفي من كومة

1. Extract the root node value

استخراج قيمة عقدة الجذر



Continued

Figure 18 Removing the Minimum Value from a Heap

Removing an Arbitrary Node from a Heap (cont.)

2. Move the value of the last node of the heap into the root node, and remove the last node. Heap property may be violated for root node (one or both of its children may be smaller).

- نقل قيمة العقدة الأخيرة من كومة إلى عقدة الجذر، وإزالة العقدة الأخيرة. قد يتم انتهاك الملكية هيب لعقدة الجذر (أحدهما أو كلاهما من أبنائها قد يكون أصغر).

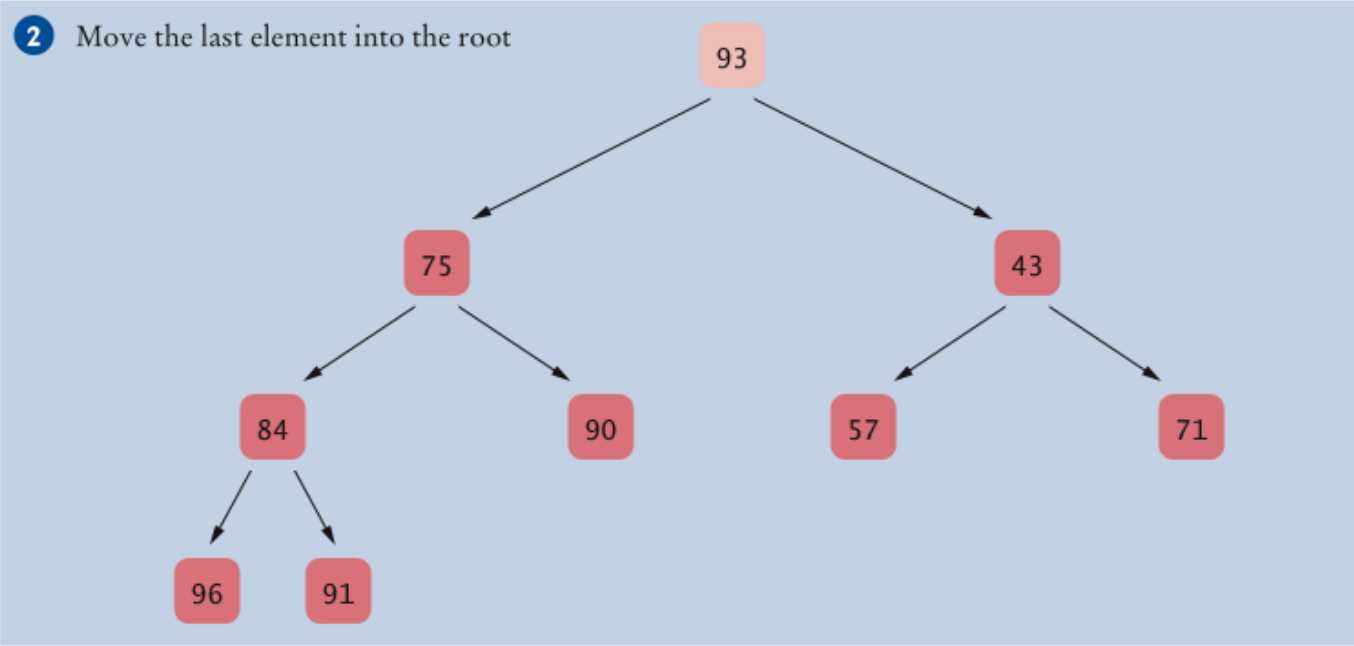


Figure 18 Removing the Minimum Value from a Heap

Continued

Removing an Arbitrary Node from a Heap (cont.)

3. Promote the smaller child of the root node. Root node again fulfills the heap property. Repeat process with demoted child. Continue until demoted child has no smaller children. Heap property is now fulfilled again. This process is called “fixing the heap”.

- تعزيز أصغر من عقدة الجذر الطفل. عقدة الجذر تلبي مرة أخرى الممتلكات الكومة. كرر العملية مع الطفل رتبته. تستمر حتى الطفل تخفيض ديه أطفال لا أصغر. هو الوفاء الملكية كومة الآن مرة أخرى. وتسمى هذه العملية "تحديد كومة".

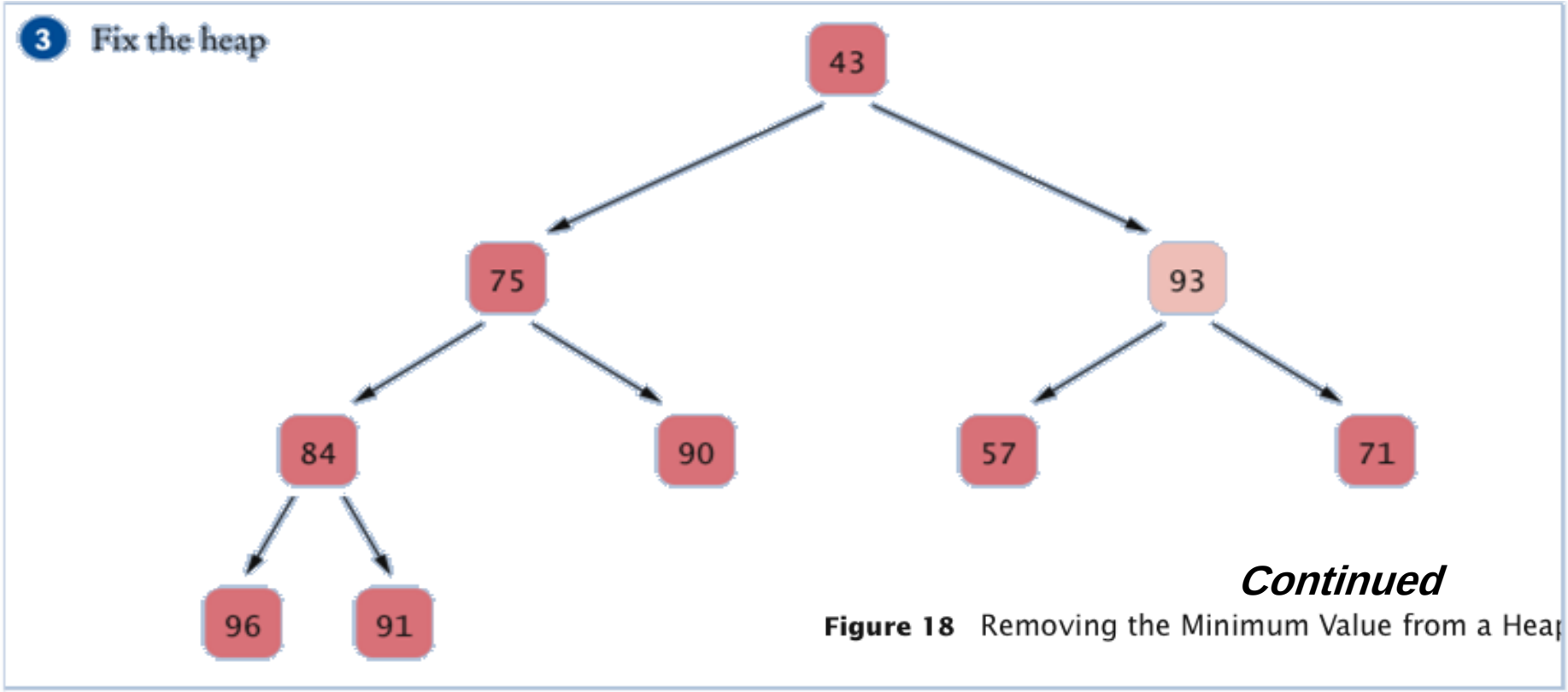


Figure 18 Removing the Minimum Value from a Heap

Removing an Arbitrary Node from a Heap (cont.)

3. Promote the smaller child of the root node. Root node again fulfills the heap property. Repeat process with demoted child. Continue until demoted child has no smaller children. Heap property is now fulfilled again. This process is called “fixing the heap”.

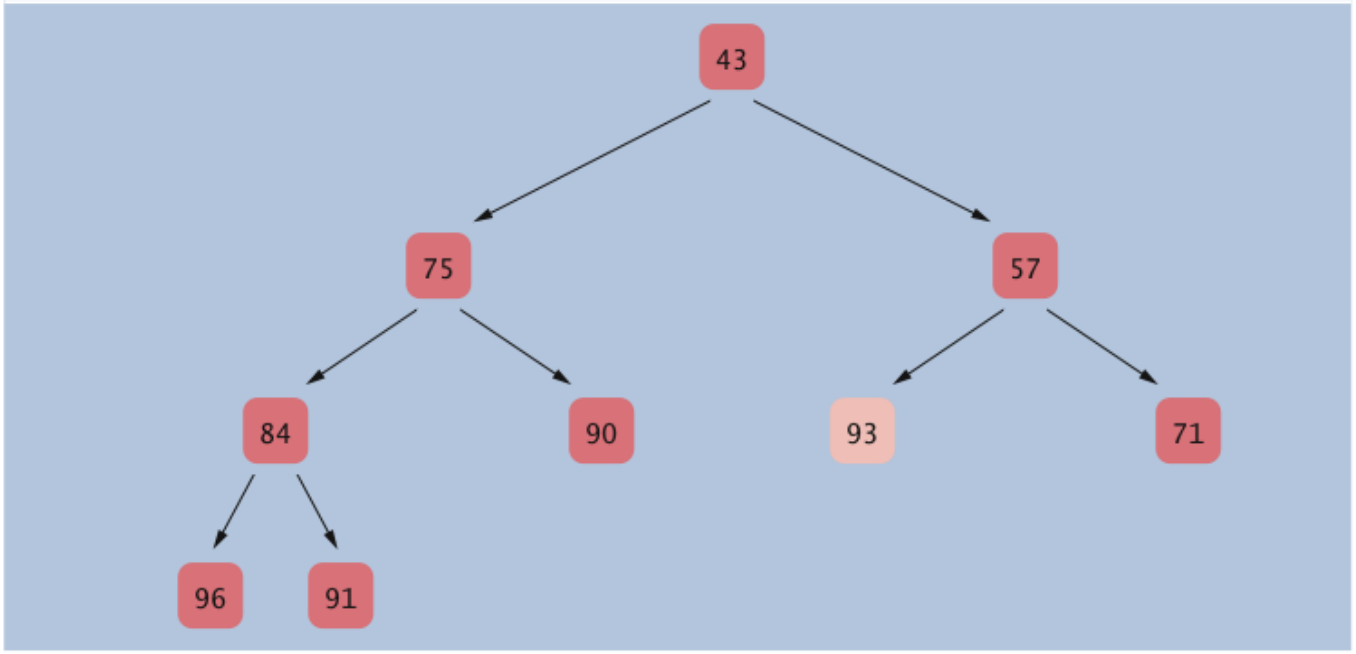


Figure 18 Removing the Minimum Value from a Heap

Heap Efficiency

- Insertion and removal operations visit at most h nodes
- h: Height of the tree
- If n is the number of elements, then
  $2^{h-1} \leq n < 2^h$
  or
  $h-1 \leq \log_2(n) < h$
- Thus, insertion and removal operations take $O(\log(n))$ steps
- Heap's regular layout makes it possible to store heap nodes efficiently in an array

- زيارة عمليات الإدخال والإخراج في معظم العقد ح
- ح: ارتفاع الشجرة
- إذا كان nهو عدد من العناصر، ثم

$2^{h-1} \leq n < 2^h$
or
$h-1 \leq \log_2(n) < h$

- هكذا، وعمليات الإدخال والإخراج تأخذ) Oسجل (ن)) خطوات
- تخطيط منتظم كومة يجعل من الممكن لتخزين العقد كومة بكفاءة في مجموعة

Storing a Heap in an Array

تخزين كومة في صفيف

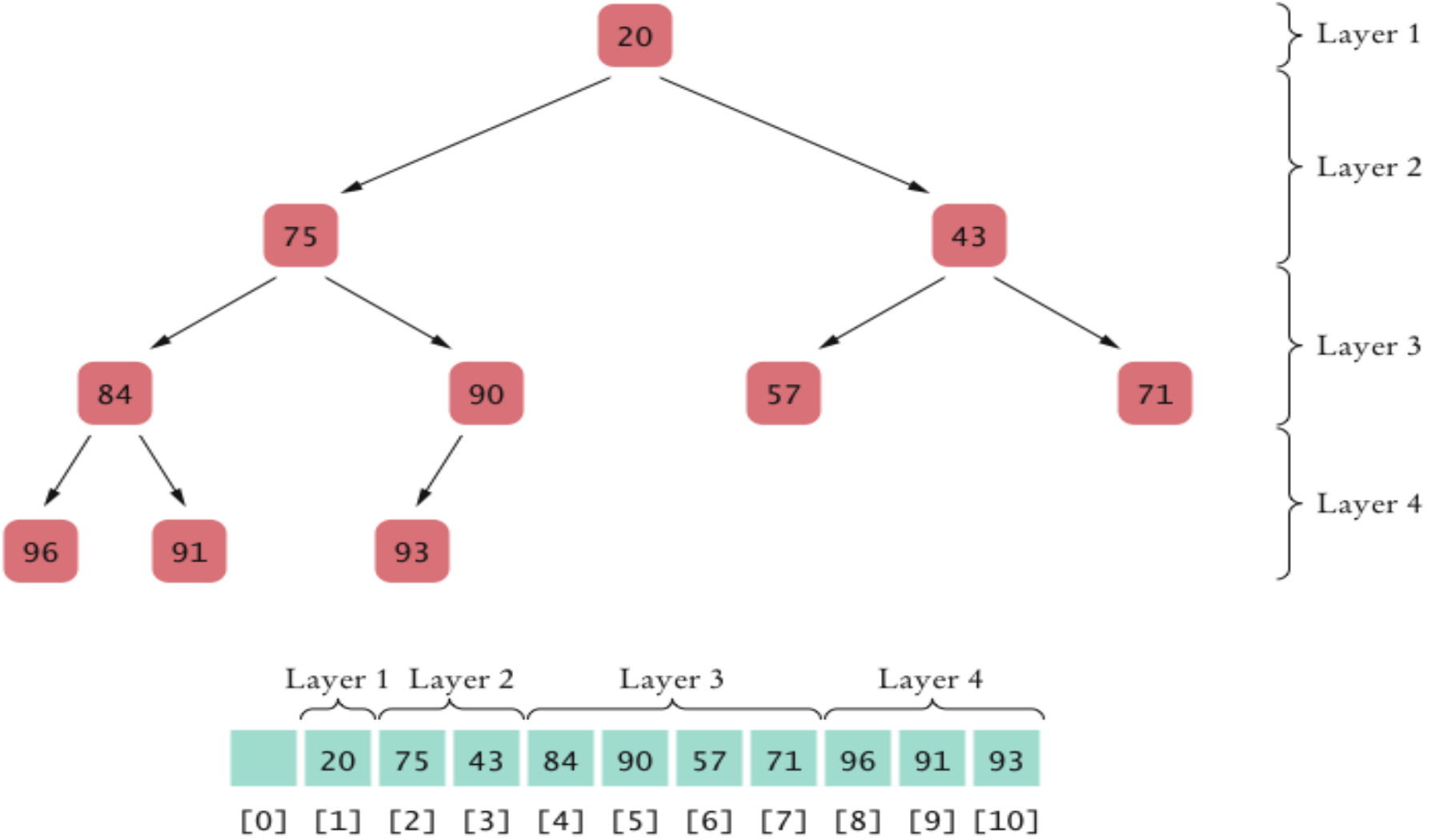


Figure 19 Storing a Heap in an Array

ch16/pqueue/MinHeap.java

```
1  import java.util.*;
2
3  /**
4     This class implements a heap.
5  */
6  public class MinHeap
7  {
8     private ArrayList<Comparable> elements;
9
10    /**
11       Constructs an empty heap.
12    */
13    public MinHeap()
14    {
15       elements = new ArrayList<Comparable>();
16       elements.add(null);
17    }
18
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
19      /**
20         Adds a new element to this heap.
21         @param newElement the element to add
22      */
23      public void add(Comparable newElement)
24      {
25         // Add a new leaf
26         elements.add(null);
27         int index = elements.size() - 1;
28
29         // Demote parents that are larger than the new element
30         while (index > 1
31               && getParent(index).compareTo(newElement) > 0)
32         {
33            elements.set(index, getParent(index));
34            index = getParentIndex(index);
35         }
36
37         // Store the new element into the vacant slot
38         elements.set(index, newElement);
39      }
40
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
41      /**
42         Gets the minimum element stored in this heap.
43         @return the minimum element
44      */
45      public Comparable peek()
46      {
47         return elements.get(1);
48      }
49
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
50      /**
51         Removes the minimum element from this heap.
52         @return the minimum element
53      */
54      public Comparable remove()
55      {
56         Comparable minimum = elements.get(1);
57
58         // Remove last element
59         int lastIndex = elements.size() - 1;
60         Comparable last = elements.remove(lastIndex);
61
62         if (lastIndex > 1)
63         {
64            elements.set(1, last);
65            fixHeap();
66         }
67
68         return minimum;
69      }
70
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
71      /**
72         Turns the tree back into a heap, provided only the root
73         node violates the heap condition.
74      */
75      private void fixHeap()
76      {
77         Comparable root = elements.get(1);
78
79         int lastIndex = elements.size() - 1;
80         // Promote children of removed root while they are smaller than last
81
82         int index = 1;
83         boolean more = true;
84         while (more)
85         {
86            int childIndex = getLeftChildIndex(index);
87            if (childIndex <= lastIndex)
88            {
89               // Get smaller child
90
91               // Get left child first
92               Comparable child = getLeftChild(index);
93
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
94                // Use right child instead if it is smaller
95                if (getRightChildIndex(index) <= lastIndex
96                 && getRightChild(index).compareTo(child) < 0)
97                {
98                   childIndex = getRightChildIndex(index);
99                   child = getRightChild(index);
100               }
101
102               // Check if larger child is smaller than root
103               if (child.compareTo(root) < 0)
104               {
105                  // Promote child
106                  elements.set(index, child);
107                  index = childIndex;
108               }
109               else
110               {
111                  // Root is smaller than both children
112                  more = false;
113               }
114            }
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
115             else
116             {
117                 // No children
118                 more = false;
119             }
120         }
121
122         // Store root element in vacant slot
123         elements.set(index, root);
124     }
125
126     /**
127         Returns the number of elements in this heap.
128     */
129     public int size()
130     {
131         return elements.size() - 1;
132     }
133
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
134     /**
135        Returns the index of the left child.
136        @param index the index of a node in this heap
137        @return the index of the left child of the given node
138     */
139     private static int getLeftChildIndex(int index)
140     {
141        return 2 * index;
142     }
143
144     /**
145        Returns the index of the right child.
146        @param index the index of a node in this heap
147        @return the index of the right child of the given node
148     */
149     private static int getRightChildIndex(int index)
150     {
151        return 2 * index + 1;
152     }
153
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
154     /**
155        Returns the index of the parent.
156        @param index the index of a node in this heap
157        @return the index of the parent of the given node
158     */
159     private static int getParentIndex(int index)
160     {
161        return index / 2;
162     }
163
164     /**
165        Returns the value of the left child.
166        @param index the index of a node in this heap
167        @return the value of the left child of the given node
168     */
169     private Comparable getLeftChild(int index)
170     {
171        return elements.get(2 * index);
172     }
173
```

Continued

ch16/pqueue/MinHeap.java (cont.)

```
174      /**
175         Returns the value of the right child.
176         @param index the index of a node in this heap
177         @return the value of the right child of the given node
178      */
179      private Comparable getRightChild(int index)
180      {
181         return elements.get(2 * index + 1);
182      }
183
184      /**
185         Returns the value of the parent.
186         @param index the index of a node in this heap
187         @return the value of the parent of the given node
188      */
189      private Comparable getParent(int index)
190      {
191         return elements.get(index / 2);
192      }
193   }
```

ch16/pqueue/HeapDemo.java

```
1  /**
2     This program demonstrates the use of a heap as a priority queue.
3  */
4  public class HeapDemo
5  {
6     public static void main(String[] args)
7     {
8        MinHeap q = new MinHeap();
9        q.add(new WorkOrder(3, "Shampoo carpets"));
10       q.add(new WorkOrder(7, "Empty trash"));
11       q.add(new WorkOrder(8, "Water plants"));
12       q.add(new WorkOrder(10, "Remove pencil sharpener shavings"));
13       q.add(new WorkOrder(6, "Replace light bulb"));
14       q.add(new WorkOrder(1, "Fix broken sink"));
15       q.add(new WorkOrder(9, "Clean coffee maker"));
16       q.add(new WorkOrder(2, "Order cleaning supplies"));
17
18       while (q.size() > 0)
19          System.out.println(q.remove());
20    }
21 }
```

ch16/pqueue/WorkOrder.java

```
1  /**
2     This class encapsulates a work order with a priority.
3  */
4  public class WorkOrder implements Comparable
5  {
6     private int priority;
7     private String description;
8
9     /**
10       Constructs a work order with a given priority and description.
11       @param aPriority the priority of this work order
12       @param aDescription the description of this work order
13    */
14    public WorkOrder(int aPriority, String aDescription)
15    {
16       priority = aPriority;
17       description = aDescription;
18    }
19
```

Continued

ch16/pqueue/WorkOrder.java (cont.)

```
20      public String toString()
21      {
22         return "priority=" + priority + ", description=" +
description;
23      }
24
25      public int compareTo(Object otherObject)
26      {
27         WorkOrder other = (WorkOrder) otherObject;
28         if (priority < other.priority) return -1;
29         if (priority > other.priority) return 1;
30         return 0;
31      }
32   }
```

Program Run:

```
priority=1, description=Fix broken sink
priority=2, description=Order cleaning supplies
priority=3, description=Shampoo carpets
priority=6, description=Replace light bulb
priority=7, description=Empty trash
priority=8, description=Water plants
priority=9, description=Clean coffee maker
priority=10, description=Remove pencil sharpener shavings
```

Self Check 16.15

The software that controls the events in a user interface keeps the events in a data structure. Whenever an event such as a mouse move or repaint request occurs, the event is added. Events are retrieved according to their importance. What abstract data type is appropriate for this application?

Answer: A priority queue is appropriate because we want to get the important events first, even if they have been inserted later.

- البرنامج الذي يتحكم في الأحداث في واجهة المستخدم يحتفظ الأحداث في بنية بيانات. عند حدوث حدث مثل تحرك الماوس أو إعادة رسم الطلب، إضافة الحدث. تم استردادها الأحداث وفقا لأهميتها. ما هو نوع البيانات مجردة غير مناسبة لهذا الطلب؟
- الجواب: طابور الأولوية هو المناسب لأننا نريد للحصول على الأحداث الهامة أولا، حتى إذا كان قد تم إدراجها في وقت لاحق.

Self Check 16.16

Could we store a binary search tree in an array so that we can quickly locate the children by looking at array locations `2 * index` and `2 * index + 1`?

Answer: Yes, but a binary search tree isn't almost filled, so there may be holes in the array. We could indicate the missing nodes with `null` elements.

- يمكن أن نقوم بتخزين شجرة البحث الثنائية في مجموعة حتى نتمكن من العثور بسرعة على الأطفال من خلال النظر في مواقع مجموعة ٢ * ٢ * مؤشر ومؤشر + ١؟
- الجواب: نعم، ولكن شجرة البحث الثنائية لا شغل تقريبا، لذلك قد يكون هناك ثقوب في مجموعة. نحن يمكن أن يشير العقد المفقود مع عناصر فارغة.

The Heapsort Algorithm

- Based on inserting elements into a heap and removing them in sorted order
- This algorithm is an $O(n \log(n))$ algorithm:
  - *Each insertion and removal is O(log(n))*
  - *These steps are repeated n times, once for each element in the sequence that is to be sorted*

- وبناء على إدراج العناصر في كومة وإزالتها من أجل فرزها
- هذه الخوارزمية هي $O(n \log(n))$ الخوارزمية:
- كل الإدراج وإزالة هو $O(n \log(n))$
- وتتكرر هذه الخطوات ن مرات، مرة واحدة لكل عنصر في تسلسل التي سيتم فرزها

The Heapsort Algorithm

خوارزمية Heapsort و

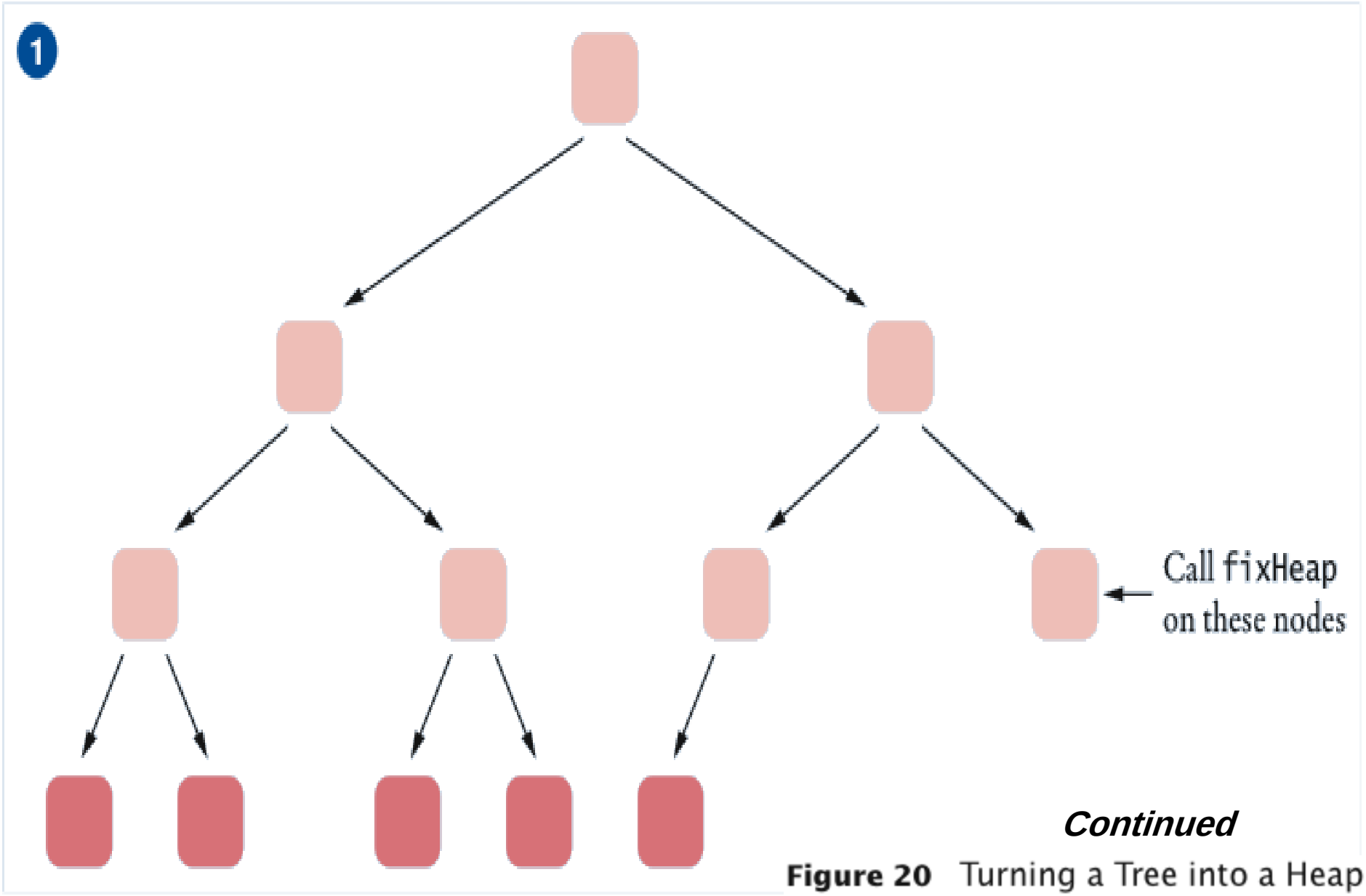
- Can be made more efficient
  - *Start with a sequence of values in an array and "fixing the heap" iteratively*
- First fix small subtrees into heaps, then fix larger trees
- Trees of size 1 are automatically heaps
- Begin the fixing procedure with the subtrees whose roots are located in the next-to-lowest level of the tree
- Generalized `fixHeap` method fixes a subtree with a given root index:

```
void fixHeap(int rootIndex, int
lastIndex)
```

يمكن أن تكون أكثر كفاءة تبدأ سلسلة من القيم في مجموعة و"تحديد كومة" تكراري
إصلاح أولا الأشجار الفرعية الصغيرة إلى أكوام، ثم حل الأشجار الكبيرة
أشجار من حجم ١ أكوام تلقائيا
بدء عملية إصلاح مع الأشجار الفرعية التي تقع في جذور المقبل من أدنى مستوى للشجرة
تعميم طريقة fixHeap بإصلاح الشجرة مع مؤشر الجذر معينة:

Turning a Tree into a Heap

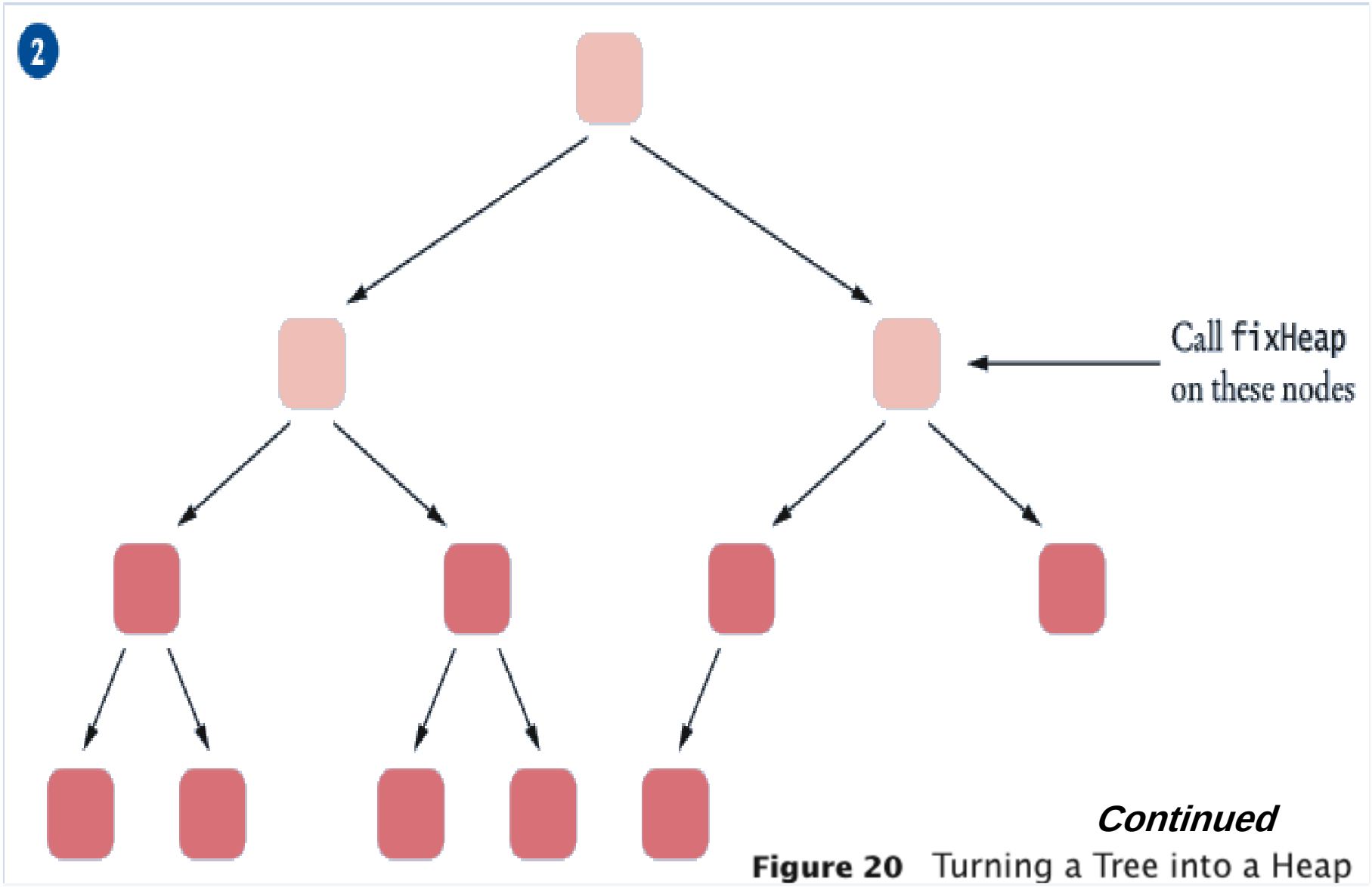
تحول شجرة إلى كومة



Continued

Figure 20 Turning a Tree into a Heap

Turning a Tree into a Heap (cont.)



Continued

Figure 20 Turning a Tree into a Heap

Turning a Tree into a Heap (cont.)

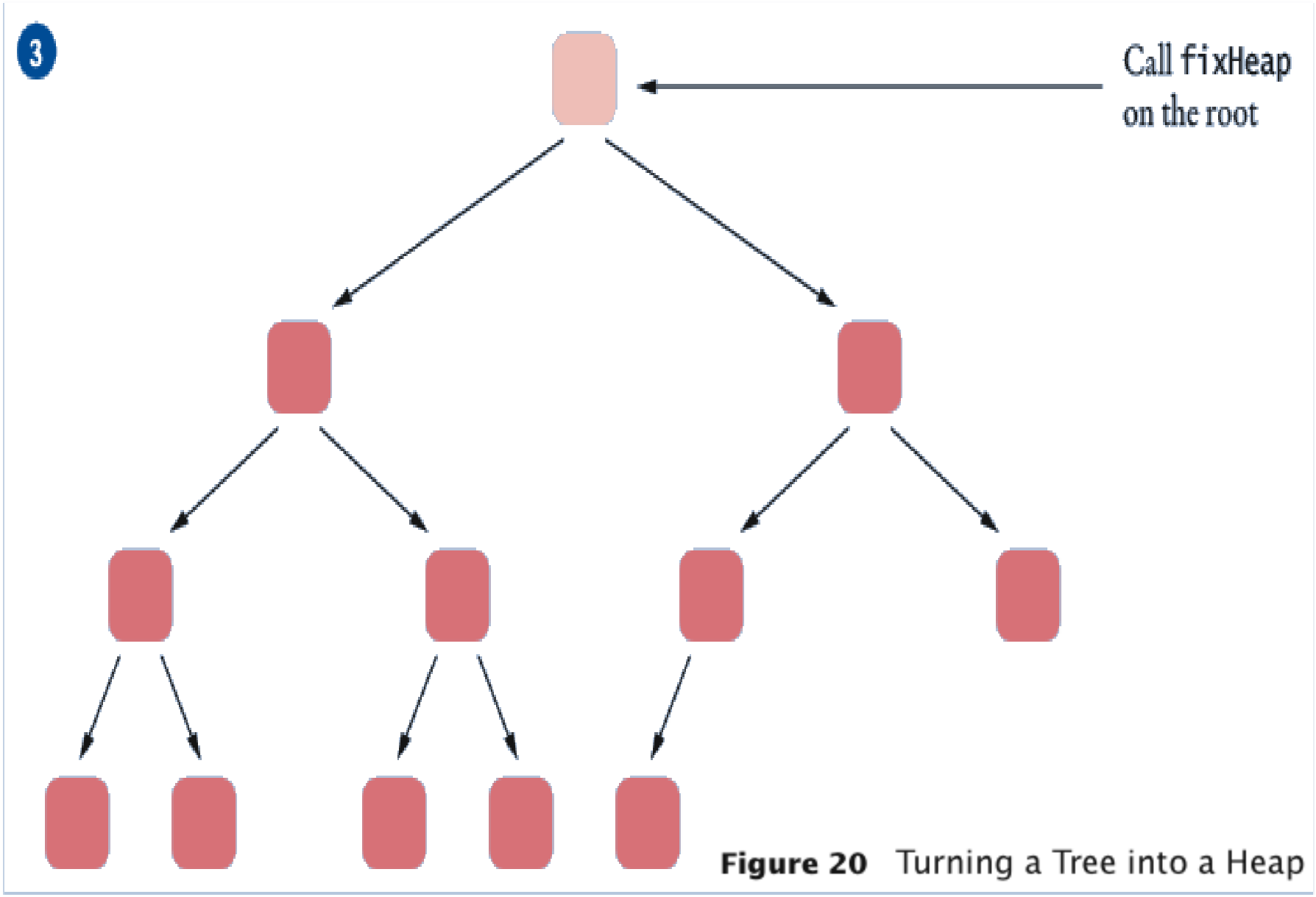


Figure 20 Turning a Tree into a Heap

The Heapsort Algorithm

خوارزمية Heapsortو

- After array has been turned into a heap, repeatedly remove the root element
  - *Swap root element with last element of the tree and then reduce the tree length*
- Removed root ends up in the last position of the array, which is no longer needed by the heap
- We can use the same array both to hold the heap (which gets shorter with each step) and the sorted sequence (which gets longer with each step)
- Use a max-heap rather than a min-heap so that sorted sequence is accumulated in the correct order

- بعد أن تحولت مجموعة في كومة، وإزالة مرارا العنصر الجذر
- العنصر الجذر مبادلة مع العنصر الأخير من الشجرة ومن ثم تقلل من طول شجرة
- الجذر إزالة ينتهي في المركز الأخير في المصفوفة، والتي لم يعد هناك حاجة من قبل كومة
- يمكننا استخدام نفس مجموعة على حد سواء لعقد كومة (الذي يحصل أقصر مع كل خطوة) وتسلسل مرتبة (الذي يحصل لفترة أطول مع كل خطوة)
- استخدام أقصى كومة بدلا من كومة مين بحيث يتم المتراكمة تسلسل فرزها في الترتيب الصحيح

Using Heapsort to Sort an Array

لفرز صفيف Heapsortباستخدام

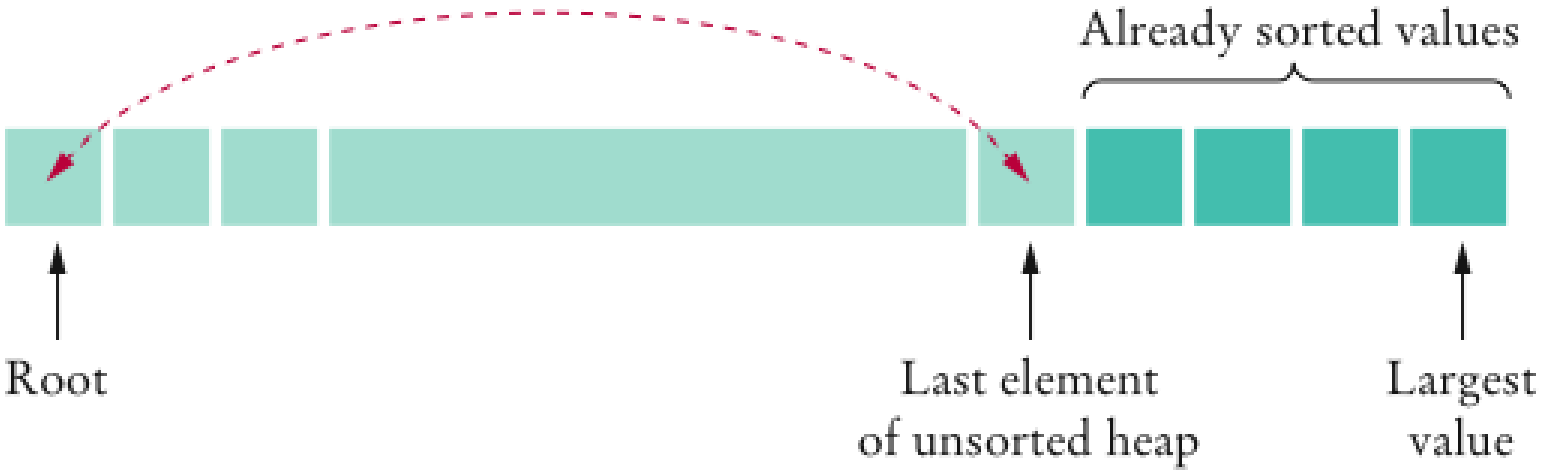


Figure 21 Using Heapsort to Sort an Array

ch16/heapsort/HeapSorter.java

```
1  /**
2     This class applies the heapsort algorithm to sort an array.
3  */
4  public class HeapSorter
5  {
6     private int[] a;
7
8     /**
9        Constructs a heap sorter that sorts a given array.
10       @param anArray an array of integers
11    */
12    public HeapSorter(int[] anArray)
13    {
14       a = anArray;
15    }
16
```

Continued

ch16/heapsort/HeapSorter.java (cont.)

```
17      /**
18         Sorts the array managed by this heap sorter.
19      */
20      public void sort()
21      {
22         int n = a.length - 1;
23         for (int i = (n - 1) / 2; i >= 0; i--)
24            fixHeap(i, n);
25         while (n > 0)
26         {
27            swap(0, n);
28            n--;
29            fixHeap(0, n);
30         }
31      }
32
```

Continued

ch16/heapsort/HeapSorter.java (cont.)

```
33      /**
34         Ensures the heap property for a subtree, provided its
35         children already fulfill the heap property.
36         @param rootIndex the index of the subtree to be fixed
37         @param lastIndex the last valid index of the tree that
38         contains the subtree to be fixed
39      */
40      private void fixHeap(int rootIndex, int lastIndex)
41      {
42         // Remove root
43         int rootValue = a[rootIndex];
44
45         // Promote children while they are larger than the root
46
47         int index = rootIndex;
48         boolean more = true;
49         while (more)
50         {
51            int childIndex = getLeftChildIndex(index);
52            if (childIndex <= lastIndex)
53            {
54               // Use right child instead if it is larger
55               int rightChildIndex = getRightChildIndex(index);
56               if (rightChildIndex <= lastIndex
57                     && a[rightChildIndex] > a[childIndex])
58               {
59                  childIndex = rightChildIndex;
60               }
```

Continued

ch16/heapsort/HeapSorter.java (cont.)

```
61
62              if (a[childIndex] > rootValue)
63              {
64                 // Promote child
65                 a[index] = a[childIndex];
66                 index = childIndex;
67              }
68              else
69              {
70                 // Root value is larger than both children
71                 more = false;
72              }
73           }
74           else
75           {
76              // No children
77              more = false;
78           }
79        }
80
81        // Store root value in vacant slot
82        a[index] = rootValue;
83     }
84
```

Continued

ch16/heapsort/HeapSorter.java (cont.)

```
85      /**
86         Swaps two entries of the array.
87         @param i the first position to swap
88         @param j the second position to swap
89      */
90      private void swap(int i, int j)
91      {
92         int temp = a[i];
93         a[i] = a[j];
94         a[j] = temp;
95      }
96
97      /**
98         Returns the index of the left child.
99         @param index the index of a node in this heap
100        @return the index of the left child of the given node
101     */
102     private static int getLeftChildIndex(int index)
103     {
104        return 2 * index + 1;
105     }
106
```

Continued

ch16/heapsort/HeapSorter.java (cont.)

```
107       /**
108          Returns the index of the right child.
109          @param index the index of a node in this heap
110          @return the index of the right child of the given node
111       */
112       private static int getRightChildIndex(int index)
113       {
114          return 2 * index + 2;
115       }
116    }
```

Self Check 16.17

Which algorithm requires less storage, heapsort or mergesort?

Answer: Heapsort requires less storage because it doesn't need an auxiliary array.

- الذي خوارزمية يتطلب تخزين أقل، heapsortأو mergesort؟
- الجواب: Heapsortيتطلب أقل التخزين لأنها لا تحتاج إلى مجموعة المساعدة.

Self Check 16.18

Why are the computations of the left child index and the right child index in the `HeapSorter` different than in `MinHeap`?

Answer: The `MinHeap` wastes the 0 entry to make the formulas more intuitive. When sorting an array, we don't want to waste the 0 entry, so we adjust the formulas instead.

- لماذا هي الحسابية للمؤشر الطفل الأيسر والمؤشر حق الطفل في HeapSorterمختلفة مما كانت عليه في MinHeap؟
- الجواب: MinHeapيهدر دخول ٠ لجعل صيغ أكثر سهولة. عندما يكون الترتيب صفيف، نحن لا نريد أن نضيع دخول ٠، لذلك علينا أن نعدل الصيغ بدلا من ذلك.