Chapter-23

1.Describe how an insertion sort works. What is the time complexity for an insertion sort?

See the text. The time complexity for an insertion sort is O(n^2).

2. Use Figure 23.1 as an example to show how to apply an insertion sort on {45, 11, 50, 59, 60, 2, 4, 7, 10}.

Omitted.

**3.** If a list is already sorted, how many comparisons will the insertionSort method perform?

n - 1 times.

**4.** Describe how a bubble sort works. What is the time complexity for a bubble sort?

See the text. The time complexity for a bubble sort is O(n^2).

5. Use Figure 23.3 as an example to show how to apply a bubble sort on {45, 11, 50, 59, 60, 2, 4, 7, 10}.

Omitted.

5. If a list is already sorted, how many comparisons will the bubbleSort method perform?

n - 1 times.

6. Describe how a merge sort works. What is the time complexity for a merge sort?

See the text. The time complexity for a merge sort is O(nlogn).

7. Use Figure 23.4 as an example to show how to apply a merge sort on {45, 11, 50, 59, 60, 2, 4, 7, 10}.

Omitted.

8. What is wrong if lines 6-15 in Listing 23.6, MergeSort.java, are replaced by the following code?

// Merge sort the first half

**int**[] firstHalf = **new int**[list.length / 2 + 1];System.arraycopy(list, 0, firstHalf, 0, list.length / 2 + 1); mergeSort(firstHalf);

// Merge sort the second half

**int** secondHalfLength = list.length - list.length / 2 - 1; **int**[] secondHalf = **new int**[secondHalfLength];System.arraycopy(list, list.length / 2 + 1,

secondHalf, 0, secondHalfLength);

mergeSort(secondHalf);

Consider a list with two elements. firstHalf will be the entire list.

9. Describe how quick sort works. What is the time complexity for a quick sort?

See the text. The time complexity for a quick sort is O(n^2) in the worst case and O(nlogn) in the average case.

10. Why is quick sort more space efficient than merge sort?

Quick sort does not need to create temporary arrays, while merge sort needs temporary arrrys.

11.Use Figure 23.7 as an example to show how to apply a quick sort on {45, 11, 50, 59, 60, 2, 4, 7, 10}.

Omitted.

12. If lines 37-38 in the QuickSort program is removed, will it still work? Give a counter example to show that it will not work.

Try to sort this list {2, 22, 2, 5, -3} without the code in ines 37-38.

13. What is a complete binary tree? What is a heap? Describe how to remove the root from a heap and how to add a new object to a heap.

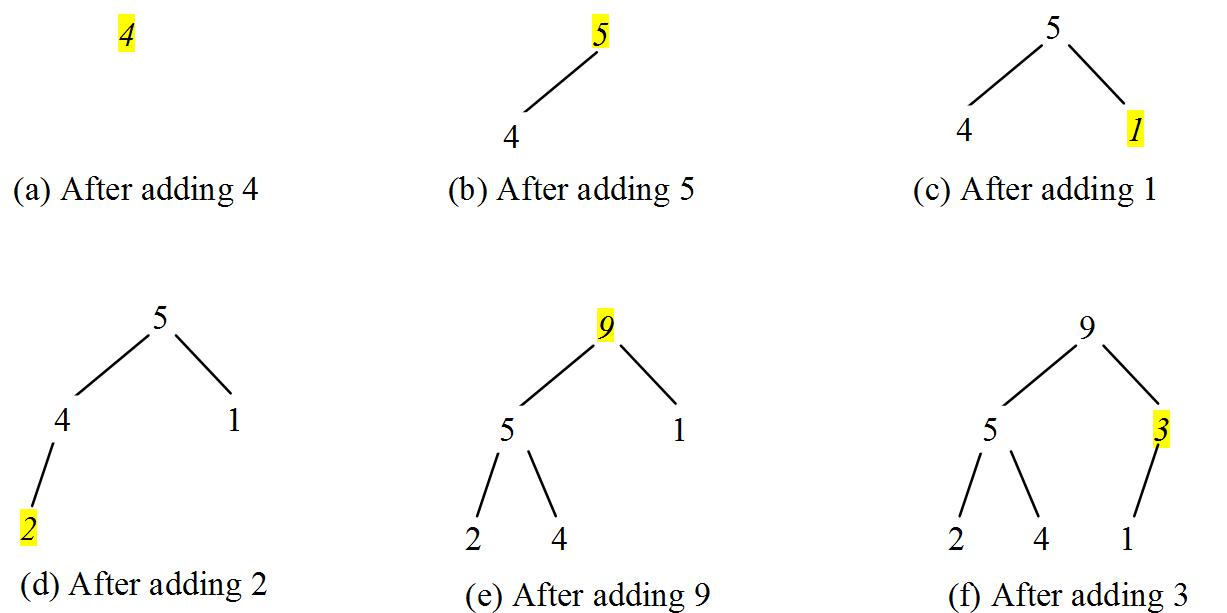
A binary tree is complete if every level of the tree is full except that the last level may not be full and all the leaves on the last level are placed left-most. A heap is a binary tree with the following properties:

1. It is a complete binary tree.
2. Each node is greater than or equal to any of its children.

14.When a heap in Figure 23.11a is stored in a list, what is index for element 44? What is the index of parent for 44 and what is the index of left and right children of 44?

The index for 44 is 5. The parent index for 44 is 2. The left child index for 44 is 11 and the right child index for 44 is 12.

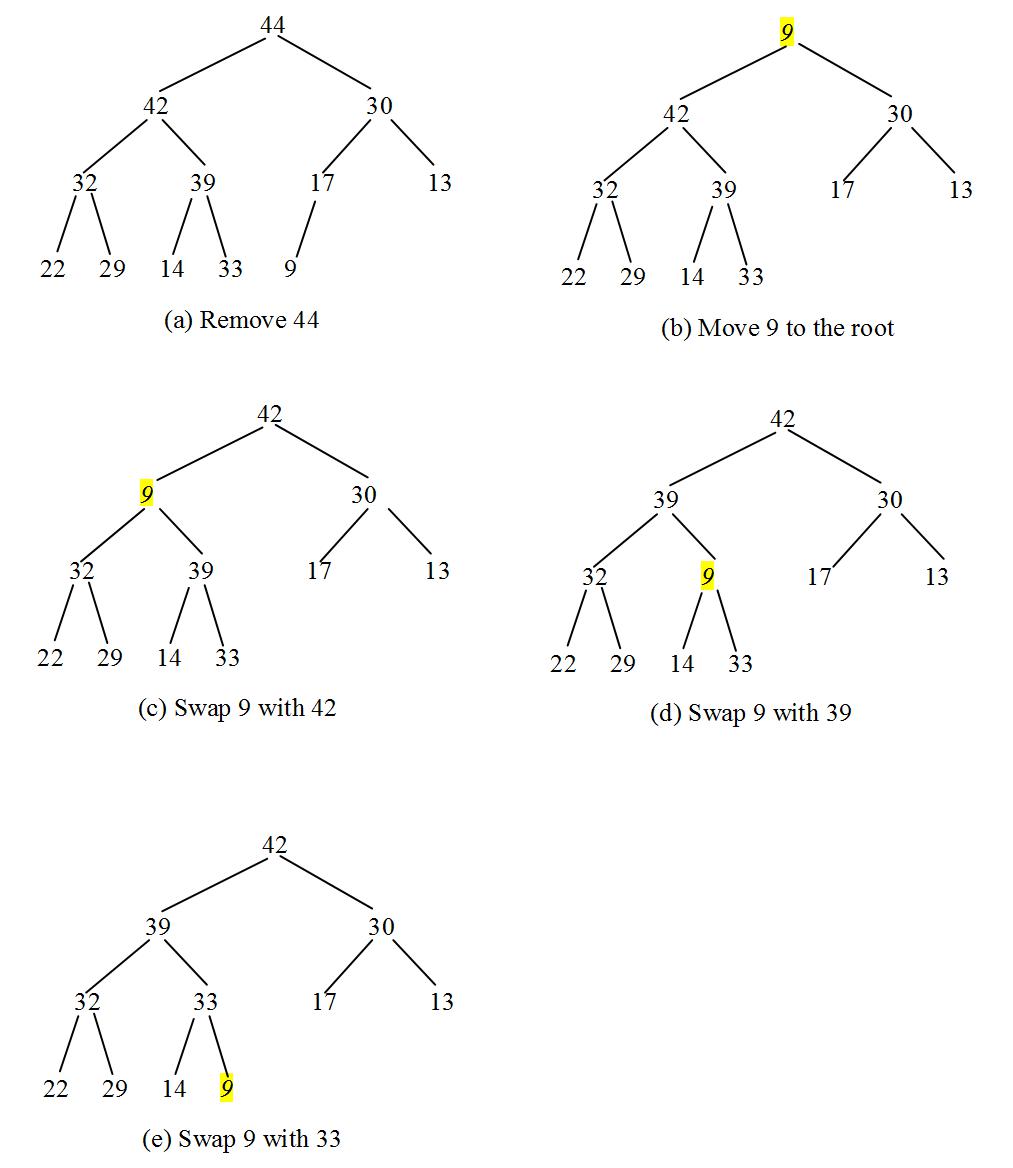
15. Add the elements 4, 5, 1, 2, 9, and 3 into a heap in this order. Draw the diagrams to show the heap after each element is added.



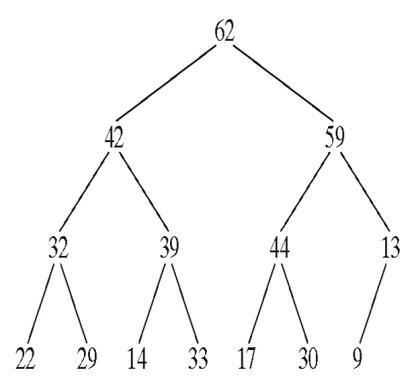
15. Show the steps of creating a heap using {45, 11, 50, 59, 60, 2, 4, 7, 10}.

Omitted.

17. Show the heap after the root in the heap in Figure 23.15c is removed.



Given the following heap, show the steps of removing all nodes from the heap.



Omitted.

18. Which of the following statements are wrong?

1. Heap<Object> heap1 = **new** Heap<>();
2. Heap<Number> heap2 = **new** Heap<>();
3. Heap<BigInteger> heap3 = **new** Heap<>();
4. Heap<Calendar> heap4 = **new** Heap<>();
5. Heap<> heap5 = **new** Heap<>();

Lines 1 and 2 are wrong, because Object and Number do not implement Comparable.

19. Modify line 12 in Listing 23.10 so that the elements are sorted in a non-increasing order.

Change it to

for (int i = 0; i < list.length; i++)

20.

What is the return value from invoking the remove method if the heap is empty?

The return value will be null.

21.

What is the time complexity of inserting a new element into a heap and what is the time complexity of deleting an element from a heap?

O(logn) for both insertion and deletion.

22.

What is the height of an empty heap? What is the height of a heap with only one element? What is the height of a non-empty heap? What is the height of a heap with 16, 17, and 512 elements? If the height of a heap is 5, what is the maximum number of nodes in the heap?

The height of an empty heap is -1 be definition. The height of a heap with only one element is 0. The height of a non-empty heap is the length of a longest path from the root to a leaf. The height of heap with 16 elements is 4. The height of heap with 17 elements is 4. The height of heap with 512 elements is 9. The maximum number of nodes in a heap of height 5 is 63.

23.

Can you sort a list of strings using a bucket sort?

Bucket sort is not suitable for sorting strings.

24.

Show how the radix sort works using the numbers 454, 34, 23, 43, 74, 86, and 76.

Omitted.

25.Describe how external sort works. What is the complexity of the external sort algorithm?

See the text. The time complexity for an external sort is O(nlogn) on disk I/O of blocks of data.

26. Ten numbers {2, 3, 4, 0, 5, 6, 7, 9, 8, 1} are stored in the external file largedata.dat.

Trace the SortLargeFile program by hand with MAX\_ARRAY\_SIZE 2.

Omitted.

Chapter-24

01. Suppose list is an instance of MyList, can you get an iterator for list using list.iterator()?

Yes. Because MyList is Iterable.

02. Can you create a list using new MyList()?

No. Because MyList is an interface.

03. What methods in Collection are overridden as default methods in MyList?

The methods isEmpty(), contains(Object e), add(E e), remove(Object e), size(), containsAll(Collection<?> c), addAll(Collection<? extends E> c),removeAll(Collection<?> c), retainAll(Collection<?> c), toArray(), and toArray(T[] are overridden in MyList.

04. What are the benefits of overriding the methods in Collection as default methods in MyList?

Default methods is a new feature in Java 8. The default methods providedefualt implementation for some methods in the interface.

05. What are the limitations of the array data type?

An array is a fixed-size data structure. Once an array is created, its size cannot be changed.

06. MyArrayList is implemented using an array, and an array is a fixed-size data structure. Why is MyArrayList considered a dynamic data structure?

MyArrayList is implemented using an array and an array is a fixed-size data structure. But MyArrayList is considered as a dynamic data structure, because its storage size changes behind the scene and hidden from the user.

07. Show the length of the array in MyArrayList after each of the following statements is executed.

1. MyArrayList<Double> list = **new** MyArrayList<>();
2. list.add(1.5);
3. list.trimToSize();
4. list.add(3.4);
5. list.add(7.4);
6. list.add(17.4);

After line 1, list's length is 16. After line 2, list's length is 16. After line 3, list's length is 1. After line 4, list's length is 3. After line 5, list's length is 3. After line 6, list's length is 7.

07. What is wrong if lines 11-12 in Listing 24.2, MyArrayList.java,

**for** (**int** i = 0; i < objects.length; i++)

add(objects[i]);

are replaced by

data = objects;

size = objects.length;

If

for (int i = 0; i < objects.length; i++)

add(objects[i]);

are replaced by

super(objects);

When constructing an ArrayList using new ArrayList(objects), the super class' constructor is invoked first to add element in objects to data. However, data has not been initialized yet. data will be initialized after the body of the superclass' constructor is executed. So you will get a NullPointerException when attempting to add an element to data. See Supplement III.I, "Initialization Block," for reference.If

for (int i = 0; i < objects.length; i++)

add(objects[i]);

are replaced by

data = objects;

size = objects.length;

Then data and objects refer to the same array. This is a security hole. You may change ArrayList by directing changing the array elements through objects.

08. If you change the code in line 33 in Listing 24.2, MyArrayList.java, from

E[] newData = (E[])(**new** Object[size \* 2 + 1]);

to

E[] newData = (E[])(**new** Object[size \* 2]);

the program is incorrect. Can you find the reason?

(Hint: To find the bug, perform trimToSize() on an empty list, then add a new element to the list.)When an empty array list is trimmed, its size becomes 0. If you create a new array by doubling its size, the new array size is still 0. Adding a new element now would cause an ArrayIndexOutOfBounds exception.

09. Will the MyArrayList class have memory leak if the following code in line 41 is deleted?

data = (E[])**new** Object[INITIAL\_CAPACITY];

Yes.

10. The get(index) method invokes the checkIndex(index) method (lines 59-63 in Listing 24.2) to throw an IndexOutOfBoundsException if the index is out of bounds. Suppose the add(index, e) method is implemented as follows:

**public void** add(**int** index, E e) {

checkIndex(index);

// Same as lines 23-33 in Listing 24.2 MyArrayList.java

}

What will happen if you run the following code?

MyArrayList<> list = **new** MyArrayList<>(); list.add("New York");

list.add(e) invokes list.add(list.size(),e), which will throw an exception, because size is now 0.

11. If a linked list does not contain any nodes, what are the values in head and tail? head and tail are null.

12. If a linked list has only one node, is head == tail true? List all cases in which head == tail is true.

Yes. When the list is empty, head == tail is also true.

13. Draw a diagram to show the linked list after each of the following statements is executed.

MyLinkedList<Double> list = **new** MyLinkedList<>(); list.add(1.5);

list.add(6.2);

list.add(3.4);

list.add(7.4);

list.remove(1.5);

list.remove(2);

Omitted.

15. When a new node is inserted to the head of a linked list, will the head and the tail be changed?

The head will be changed. The tail will also be changed if the list is empty before the insertion.

16. When a new node is appended to the end of a linked list, will the head and the tail be changed?

The head will be changed if the list is empty before the insertion. The tail will always be changed.

17. Simplify the following code in Listing 24.5 using a conditional expression.

**if** (current != **null**) {

result.append(", "); // Separate two elements with a comma

}

**else** {

result.append("]"); // Insert the closing ] in the string

}

result.append((current != null) ? ", " : "]");

18. Simplify the code for the removeLast() method by invoking the removeFirst() method when the size is less than or equal to 1. Is the new code more efficient in execution time?

public E removeLast() {

if (size <= 1) {

return removeFirst();

}

else {

Node<E> current = head;

for (int i = 0; i < size - 2; i++) { current = current.next;

}

Node<E> temp = tail;

tail = current;

tail.next = null;

size--;

return temp.element;

}

}

This new code is simpler and better. It reuses the existing code and makes the code easy to maintain.

The time complexity for the new code is O(1), which is the same as before. (Thanks to the UT Dallas students for their contribution. 10/14/2015)

19 What is the time complexity of the addFirst(e) and removeFirst() methods in MyLinkedList?

O(1)

20. What would be the time complexity for the size() method if the size data field is not used in MyLinkedList?

O(n)

21. Suppose you need to store a list of elements. If the number of elements in the program is fixed, what data structure should you use? If the number of elements in the program changes, what data structure should you use?

If the number of elements is fixed in the program, use array is more efficient. If the number of elements changes in the program, you may use MyArrayList or MyLinkedList.

22. If you have to add or delete the elements at the beginning of a list, should you use MyArrayList or MyLinkedList? If most of the operations on a list involve retrieving an element at a given index, should you use MyArrayList or MyLinkedList?

If you have to add or delete the elements anywhere in a list, use MyLinkedList. If most of operations on a list involve retrieving an element at a given index, use MyArrayList.

23. Both MyArrayList and MyLinkedList are used to store a list of objects. Why do we need both types of lists?

Both MyArrayList and MyLinkedList are used to store a list of objects. Why do we need two? MyLinkedList is more efficient for deletion and insertion at the beginning of the list. MyArrayList is more efficient for all other operations.

24. Implement the removeLast() method in a doubly linked list in O(1) time.

Note that in a singly linked list, to find the second-to-last node, you have loop all the way from head to the second-to-last node, which takes O(n) time. Here is the copy of the code from the text:

1. public E removeLast() {
2. if (size == 0) return null; // Nothing to remove
3. else if (size == 1) { // Only one element in the list
4. Node<E> temp = head;
5. head = tail = null; // list becomes empty
6. size = 0;
7. return temp.element;
8. }
9. else {
10. Node<E> current = head;

11

1. for (int i = 0; i < size - 2; i++)
2. current = current.next;

14

1. Node<E> temp = tail;
2. tail = current;
3. tail.next = null;
4. size--;
5. return temp.element;
6. }
7. }

In a doubly linked list, the second-to-last node can be found in O(1) time using tail.previous. Here is the complete code:

1. public E removeLast() {
2. if (size == 0) return null; // Nothing to remove
3. else if (size == 1) { // Only one element in the list
4. Node<E> temp = head;
5. head = tail = null; // list becomes empty
6. size = 0;
7. return temp.element;
8. }
9. else {
10. Node<E> temp = tail;
11. tail = tail.previous;
12. tail.next = null;
13. size--;
14. return temp.element;
15. }

25. You can use inheritance or composition to design the data structures for stacks and queues. Discuss the pros and cons of these two approaches.

Using inheritance: You can declare the stack class by extending the array list class, and the queue class by extending the linked list class. Using composition: You can declare an array list as a data field in the stack class, and a linked list as a data field in the queue class. Both designs are fine, but using composition is better because it enables you to declare a complete new stack class and queue class without inheriting the unnecessary and inappropriate methods from the array list and linked list.

26. If LinkedList is replaced by ArrayList in lines 2-3 in Listing 24.6 GenericQueue.java, what will be the time complexity for the enqueue and dequeue methods?

The time complexity for enqueue will be O(1) and for dequeue will be O(n).

27. Which lines of the following code are wrong?

1. List<> list = **new** ArrayList<>();
2. list.add("Tom");
3. list = **new** LinkedList<>();
4. list.add("Tom");
5. list = **new** GenericStack<>();
6. list.add("Tom");

Line 5 will be wrong, because GenericStack is not a subtype of MyList.

28. What is a priority queue?

In a priority queue, elements are assigned with priorities. When accessing elements, the element with the highest priority is removed first.

29. What are the time complexity of the enqueue, dequeue , and getSize methods in MyProrityQueue?

For enqueue and dequeue in a priority queue, the complexity is O(logn). For getSize(), the complexity is O(1).

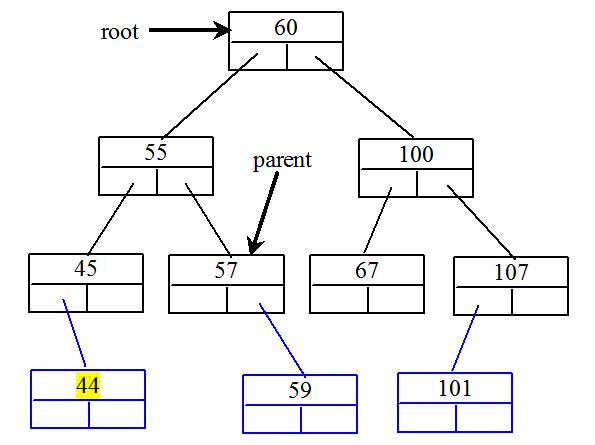
30. Which of the following statements are wrong?

1. MyPriorityQueue<Object> q1 = **new** MyPriorityQueue<>();
2. MyPriorityQueue<Number> q2 = **new** MyPriorityQueue<>();
3. MyPriorityQueue<Integer> q3 = **new** MyPriorityQueue<>();
4. MyPriorityQueue<Date> q4 = **new** MyPriorityQueue<>();
5. MyPriorityQueue<> q5 = **new** MyPriorityQueue<>();

Lines 1 and 2 are wrong, because Object and Number don't implement Comparable.

Chapter-25

01. Show the result of inserting 44 into Figure 25.4b.



02. Show the inorder, preorder, and postorder of traversing the elements in the binary tree shown in Figure 25.1b.

Inorder: A F G M R T

Preorder: G F A R M T

Postorder: A F M T R G

Hide Answer

**03.** If a set of elements is inserted into a BST in two different orders, will the two corresponding BSTs look the same? Will the inorder traversal be the same? Will the postorder traversal be the same? Will the preorder traversal be the same?

If a set of the same elements is inserted into a binary tree in two different orders, will the two corresponding binary trees look the same? No. Will the inorder traversal be the same? Yes. Will the postorder traversal be the same? No. Will the preorder traversal be the same? No.

**04.** What is the time complexity of inserting an element into a BST?

The time complexity of inserting an element to a binary tree is O(n).

**05.** Implement the search(element) method using recursion.

@Override

/\*\* Returns true if the element is in the tree \*/ public boolean search(E e) {

return search(root, e);

}

public boolean search(TreeNode<E> root, E e) {

if (root == null)

return false;

else if (e.compareTo(root.element) < 0)

return search(root.left, e);

else if (e.compareTo(root.element) > 0)

return search(root.right, e);

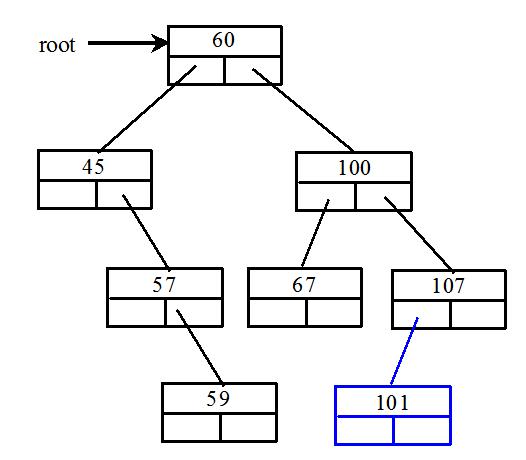
else

return true;

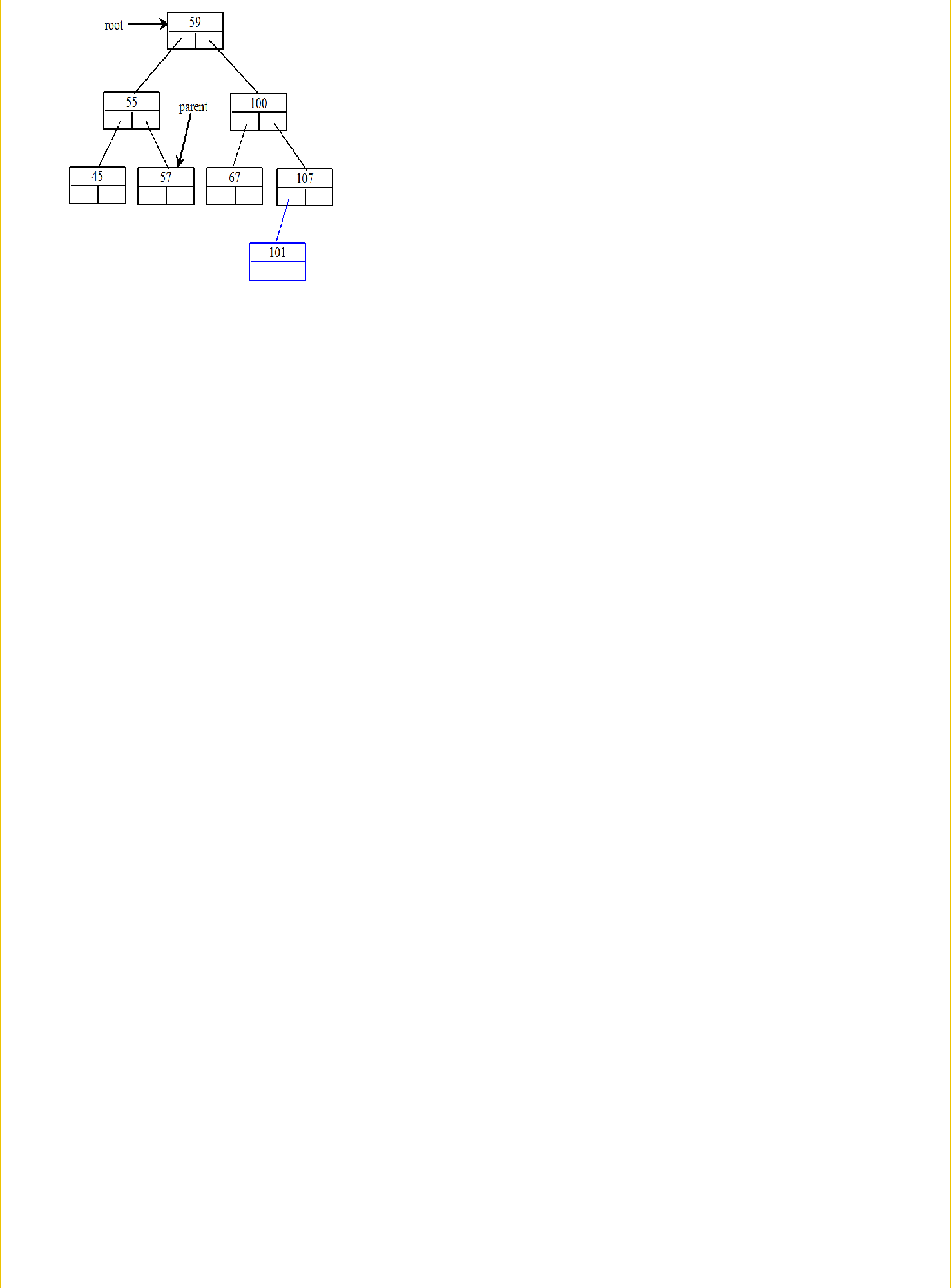
}

Hide Answer

06. Show the result of deleting 55 from the tree in Figure 25.4b.



**07.** Show the result of deleting 60 from the tree in Figure 25.4b.



**08.** What is the time complexity of deleting an element from a BST?

The time complexity of deleting an element from a binary tree is O(n).

**09.** Is the algorithm correct if lines 203-207 in Listing 25.4 in Case 2 of the delete() method are replaced by the following code?

parentOfRightMost.right = rightMost.left;

No. Consider the case when current is parentOfRightMost and current.left is rightMost. You have to assign rightMost.left to parentOfRightMost.left.

**10.** How many times will the displayTree method be invoked if the tree is empty? How many times will the displayTree method be invoked if the tree has 100 nodes?

The displayTree method will be invoked 0 time if the tree is empty. The displayTree method will be invoked 100 times if the tree has 100 nodes.

**11.** In what order are the nodes in the tree visited by the displayTree method: inorder, preorder, or postorder?

The nodes in the tree are visited in preorder.

**12.** What would happen if the code in lines 47-52 in BTView.java is moved to line 33?

You will see the line that connecting the nodes to be displayed on top of nodes starting from the center of the node, not from the edge of the node, because the line would be displayed after the node.

13. What is MVC? What are the benefits of the MVC?

See the text.

**14.** What is an iterator?

An iterator is an object that provides a uniform way for traversing the elements in a container such as a set, list, binary tree, etc.

**15.** What method is defined in the java.lang.Iterable<E> interface?

The iterator() method is defined in the java.lang.Iterable interface.

**15.** Suppose you delete implements Collection<E> from line 3 in Listing 25.3, Tree.java. Will Listing 25.10 still compile?

No. Since Tree is not iterable now, the foreach loop cannot be used.

**16.** What is the benefit of being a subtype of Iterable<E>?

Being a subtype of Iterable, the elements of the container can be traversed using a for-each loop.

**17.** Write one statement that displays the maximum and minimum element in a BST object named tree.

System.out.println("Max elememt in the tree is " + java.util.Collections.max(tree) +

* and the min elememt in the tree is " + java.util.Collections.min(tree));

**18.** Every internal node in a Huffman tree has two children. Is it true?

Yes.

**18.** What is a greedy algorithm? Give an example.

A greedy algorithm is often used in solving optimization problems. The algorithm makes the choice that is optimal locally in the hope that this choice will lead to a globally optimal solution.

**19.** If the Heap class in line 50 in Listing 25.9 is replaced by java.util.PriorityQueue, will the program still work?

Yes, except that you also have to change heap.getSize() to heap.size().

**20.** How do you replace lines 94-99 in Listing 25.11 using one line?

return root.weight < t.weight ? 1 : root.weight == t.root.weight ? 0 : -1