# Test Bank for Data Structures with Java

*John R. Hubbard*
*Anita Huray*
University of Richmond

# Chapter 1
# Object-Oriented Programming

**Answer “True” or “False”:**

**1.** An analysis of the profitability of a software solution would be done as part of the feasibility study for the project.

**2.** The best time to write a user manual for a software solution is during its maintenance stage.

**3.** The requirements analysis of a software project determines what individual components (classes) will be written.

**4.** In a large software project it is preferable to have the same engineers write the code as did the design.

**5.** In the context of software development, “implementation” refers to the actual writing of the code, using the design provided.

**6.** Software engineers use “OOP” as an acronym for “organized operational programming”.

**7.** The term “Javadoc” refers to the process of “doctoring” Java code so that it runs more efficiently.

**8.** Software engineers use “UML” as an acronym for “Unified Modeling Language”.

**9.** UML diagrams are used to represent classes and the relationships among them.

**10.** The “is-a” relationship between classes is called composition.

**11.** The “contains-a” relationship between classes is called aggregation.

**12.** The “has-a” relationship between classes is called inheritance.

**13.** A “mutable” class is one whose objects can be modified by the objects of other classes.

**14.** An extension of a mutable class is also mutable.

**15.** An extension of a immutable class is also immutable.

## Select the best answer:

**1.** During which stage of a software project are the software components determined?

- **a.** feasibility study
- **b.** requirements analysis
- **c.** design
- **d.** implementation
- **e.** testing
- **f.** maintenance

**2.** User support is part of which stage of a software project?

- **a.** feasibility study
- **b.** requirements analysis
- **c.** design
- **d.** implementation
- **e.** testing
- **f.** maintenance

**3.** During which stage of a software project is the operational definition produced?

- **a.** feasibility study
- **b.** requirements analysis
- **c.** design
- **d.** implementation
- **e.** testing
- **f.** maintenance

**4.** The actual code for a software project is written during which stage?

- **a.** feasibility study
- **b.** requirements analysis
- **c.** design
- **d.** implementation
- **e.** testing
- **f.** maintenance

**5.** During which stage of a software project would a cost-benefit analysis be done?
- **a.** feasibility study
- **b.** requirements analysis
- **c.** design
- **d.** implementation
- **e.** testing
- **f.** maintenance

**6.** The nesting of one class within another is a way to implement:
- **a.** inheritance
- **b.** aggregation
- **c.** composition
- **d.** none of the above

**7.** The specialization of one class by another is called:
- **a.** inheritance
- **b.** aggregation
- **c.** composition
- **d.** immutability
- **e.** none of the above

**8.** The prevention of the components of one class from being modified by another class is called:
- **a.** inheritance
- **b.** aggregation
- **c.** composition
- **d.** immutability
- **e.** none of the above

**9.** Defining a field of one class to be a reference to another, external class is a way to implement:
- **a.** inheritance
- **b.** aggregation
- **c.** composition
- **d.** immutability
- **e.** none of the above

**10.** Extending one class by adding more functionality is called:
- **a.** inheritance
- **b.** aggregation
- **c.** composition
- **d.** immutability
- **e.** none of the above

## Answers to True/False Questions

| | |
|---|---|
| **1.** | True |
| **2.** | False |
| **3.** | False |
| **4.** | False |
| **5.** | True. |
| **6.** | False |
| **7.** | False |
| **8.** | True |
| **9.** | True |
| **10.** | False |
| **11.** | True |
| **12.** | False |
| **13.** | True |
| **14.** | True |
| **15.** | False |

## Answers to Multiple Choice Questions

| | |
|---|---|
| **1.** | **c** |
| **2.** | **f** |
| **3.** | **b** |
| **4.** | **d** |
| **5.** | **a** |
| **6.** | **c** |
| **7.** | **a** |
| **8.** | **d** |
| **9.** | **b** |
| **10.** | **a** |

# Chapter 2
# Abstract Data Types

**Answer “True” or “False”:**

1. Java is a strongly typed programming language.
2. A variable’s data type defines the set of values that the variable may have.
3. An abstract data type defines how a set of operations are to be implemented.
4. An abstract data type can be translated into a Java interface.
5. Preconditions and postconditions are used to define operations.
6. Javadoc can be used to document preconditions and postconditions.
7. In Java, any class or interface defines a data type.
8. In Java, a class may implement several interfaces.
9. In Java, a class may extend several classes.
10. Polymorphism occurs when an object of one type is referenced by a variable of a different type.
11. The software term “information hiding” refers to the technique of naming variables with single letters, like `x`, `y`, and `z`.
12. A mutator is a method that changes the type of an object.
13. An accessor is a method that returns the value of an object.
14. In Java, an interface may be an extension of another interface.
15. In Java, an interface may be an extension of a class.

**16.** In Java, a call to a method that throws an unchecked exception need not be enclosed within a `try` block.

**17.** The `assert` statement in Java throws an unchecked exception when its condition is false.

**18.** The `assert` statement can be used to check preconditions.

**19.** Parametric polymorphism describes the property of one class extending several other classes.

**20.** Every class in Java is an extension of the `Object` class.

## Select the best answer:

**1.** Which of the following is not a primitive type in Java?
- **a.** `int`
- **b.** `double`
- **c.** `String`
- **d.** `boolean`
- **e.** `byte`

**2.** Which of the following is not a type in Java?
- **a.** `int`
- **b.** `null`
- **c.** `String`
- **d.** `Date[]`
- **e.** `Comparable`

**3.** The “state” of an object is:
- **a.** it’s data
- **b.** it’s type
- **c.** it’s methods
- **d.** it’s class
- **e.** none of the above

**4.** An `abstract` class in Java is:
- **a.** an interface
- **b.** a class with no fields
- **c.** a class with no methods
- **d.** a class that has an `abstract` method
- **e.** a class, all of whose methods are `abstract`

**5.** Which is not a class invariant for this class:

```
class Timestamp {
  int year, month, day;
}
```

**a.** $0 < day < 32$
**b.** $0 < month < 13$
**c.** $0 < year < 2500$
**d.** if ($month == 2$) $day < 30$
**e.** a `toString()` method returns a string in the form `"Dec 25 2004"`

**6.** Which of the following cannot be implemented with a Java `assert` statement:

**a.** precondition
**b.** postcondition
**c.** inheritance
**d.** class invariant
**e.** none of the above

**7.** Suppose that `X`, `Y` and `Z` are classes, `Y` extends `X`, `Z` extends `Y`, and variables `x`, `y`, and `z` are defined by:

```
X x = new X();
Y y = new Y();
Z z = new Z();
```

Then which of these statements will compile and run:

**i.** `x = z;`
**ii.** `z = x;`
**iii.** `z = (Z)x;`
**iv.** `y = (Z)x;`

**a.** None of them.
**a.** Only **i**.
**b.** Only **i** and **iii**.
**c.** Only **i**, **ii**, and **iii**.
**d.** Only **i**, **iii**, and **iv**.
**e.** All four statements.

## Answers to True/False Questions

**1.** True

**2.** True

**3.** False

**4.** True

**5.** True

| | |
|---|---|
| **6.** | True |
| **7.** | True |
| **8.** | True |
| **9.** | False |
| **10.** | True |
| **11.** | False |
| **12.** | False |
| **13.** | True |
| **14.** | False |
| **15.** | True |
| **16.** | True |
| **17.** | True |
| **18.** | True |
| **19.** | False |
| **20.** | True |

## Answers to Multiple Choice Questions

| | |
|---|---|
| **1.** | **c** |
| **2.** | **b** |
| **3.** | **a** |
| **4.** | **d** |
| **5.** | **e** |
| **6.** | **c** |
| **7.** | **c** |

# Chapter 3
# Arrays

## Answer "True" or "False":

**1.** An array is a sequence of contiguous elements.

**2.** The last element in an array with 100 elements has index 100.

**3.** The starting element in an array with 10 elements has index 0.

**4.** The following defines an array that will hold words 3 strings:

```
   String[] words = {"Ann", "Bob", "Cal"};
```

**5.** The following defines and allocates space for an array that will contains 10 words:

```
  String[] words = new String("10");
```

For questions 6 through 20 assume that we have these array definitions:

```
double[] aa = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8};
double[] bb ;
bb = new double[] {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8};
double[] cc = new double[8];
double[][]  abc = {aa, bb, cc};
```

**6.** Arrays `aa`, `bb`, and `cc` each have length 8.

**7.** The value of the expression `aa==bb` is true.

**8.** This statement will print the eight values in `bb`

```
  System.out.println(bb);
```

**9.** The statement

```
  System.out.println(aa[1] + "," + bb[3] + "," + cc[5]);
```

will print

```
  2.2, 4.4, 0.0
```

**10.** This statement will render `cc` a separate, independent copy of `aa`:

```
  cc = aa;
```

**11.** This statement will render `cc` a separate, independent copy of `aa`:

```
System.arraycopy(aa, 0, cc, 0, 4);
```

**12.** The Sequential Search Algorithm described on page 80 in section 3.5 will return the value 0 if we use it to find 9.9 in array `aa`.

**13.** The Sequential Search Algorithm described on page 80 in section 3.5 requires 8 comparisons if we use it to search for 9.9 in array `aa`.

**14.** The Binary Search Algorithm described on page 80 in section 3.5 requires exactly 2 comparisons if to search for 9.9 in array `aa`.

**15.** The Sequential Search Algorithm described on page 80 in section 3.5 requires 3 comparisons if to search for 2.8 in array `aa`.

**16.** The Binary Search Algorithm described on page 80 in section 3.5 requires 2 comparisons if to search for 9.9 in `aa`.

**17.** Consider the `IntArrays.resize()` method defined on page 90 in section 3.9. The statement below will have no affect on `cc`:

```
resize(cc, 10);
```

**18.** This statement will construct a `java.util.Vector` with the same elements as those in `aa`:

```
java.util.Vector v = new java.util.Vector(aa);
```

**19.** The two-dimensional array `abc` has 3 rows, each with 8 columns.

**20.** The value of `abc[1][2]` is the same as the value of `abc[2][1]`.

## Select the best answer:

**1.** How many elements of this array

```
int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
```

will be examined when the Sequential Search is used to search for 50:

**a.** 0
**b.** 1
**c.** 2
**d.** 4
**e.** 8

**2.** How many elements of this array

```
int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
```

will be examined when the Binary Search is used to search for 88:

**a.** 0
**b.** 1
**c.** 2
**d.** 4
**e.** 8

**3.** To which of these complexity classes does the function $n^2$ belong:

**a.** $O(n)$
**b.** $\Theta(n^3)$
**c.** $\Omega(n^3)$
**d.** $o(n)$
**e.** $\omega(n)$

**4.** Which complexity class contains $\omega(n)$:

**a.** $O(n)$
**b.** $\Theta(n)$
**c.** $\Omega(n)$
**d.** $o(n)$
**e.** none of the above

**5.** How many constructors does the `java.util.Vector` class have?

**a.** 0
**b.** 1
**c.** 2
**d.** 3
**e.** 4

**6.** What will the contents of the array `a[]` be after this code executes:

```
int[] a = {22, 33, 44, 55, 66, 77};
System.arraycopy(a, 1, a, 2, 3);
```

**a.** `{0, 22, 33, 44, 66, 77}`
**b.** `{22, 22, 33, 44, 55, 77}`
**c.** `{22, 0, 33, 44, 55, 77}`
**d.** `{22, 33, 33, 44, 55, 77}`
**e.** None of the above.

**7.** What will the value of a[1][2] be after this code executes:

```
int[][] a = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
for (int i=0; i<2; i++)
  for (int j=0; j<2; j++)
    IntArrays.swap(a[i], a[j]);  // see Listing 3.8 on page 90
```

**a.** 4
**b.** 5
**c.** 6
**d.** 7
**e.** 8

## Answers to True/False Questions

**1.** True.

**2.** False.

**3.** True.

**4.** True.

**5.** False.

**6.** True.

**7.** False.

**8.** False.

**9.** True.

**10.** True.

**11.** False.

**12.** False.

**13.** True.

**14.** False.

**15.** True.

**16.** True.

**17.** False.

**18.** False.

**19.** True.

**20.** False.

## Answers to Multiple Choice Questions

**1.** **e**

**2.** **c**

**3.** **e**

**4.** **c**

**5.** **e**

**6.** **d**

**7.** **c**

# Chapter 4
# Linked Structures

**Answer "True" or "False":**

**1.** Storing information in a linked list structure is a bad idea if we expect to do a lot queries (searches).

**2.** Storing information in a linked list structure is a bad idea if we plan to do a lot of insertions.

**3.** Storing information in a linked list structure is a bad idea if we plan to do a lot of deletions.

**4.** Storing information in a linked list structure is a bad idea if we plan to print the entire list frequently.

Questions 5-12 use this `Node` class (on page 125):

```
class Node{
  int data;
  Node next;
}
```

Assume we have executed these statements:

```
Node x = new Node();
Node y = new Node();
x.data = 8;
```

**5.** The value of `y.data` is 0.

**6.** The value of `x.next` is 0.

**7.** This statement throws an exception:

```
System.out.println(x);
```

**8.** This statement throws an exception:

```
System.out.println(y.data);
```

**9.** This statement throws an exception:

```
System.out.println(x.next);
```

**10.** This statement throws an exception:

```
System.out.println(y.next.data);
```

**11.** This statement connects the nodes so that the data are in order:

```
y.next = x;
```

**12.** This code constructs a linked list with values 0 through 3:

```
Node start = new Node();
for (int i=1; i<4; i++){
  Node next= new Node();
  next.data = i;
}
```

Questions 13-14 use this `Node` class (Listing 4.10 on page 134):

```
class Node{
  int data;
  Node next;

  Node(int data) {
    this.data = data;
  }

  Node(int data, Node next){
    this.data = data;
    this.next = next;
  }
}
```

**13.** The following will correctly compute the length of the linked list which begins with a node named `start`:

```
int length = 0;
for (Node n = start; n != null; n = n.next) {
  ++length;
}
```

**14.** This prints every other node the linked list beginning at start:

```
for (Node n = start; n != null; n = n.next.next) {
  System.out.println(n.data);
}
```

Questions 15-24 use the `Linked List` class defined in Listing 4.15 on page 141:

**15.** This correctly declares an empty list:

```
LinkedList list = new LinkedList();
```

**16.** This declares a nonempty list of length 1:

```
LinkedList list = new LinkedList( new Node(33, null) );
```

**17.** This will correctly compute the length of a nonempty list:

```
int length = 0;
for (Node n = list.start; n != null; n = n.next) {
  ++length;
}
```

**18.** This code prints the entire contents of a nonempty list :

```
for (Node n = list.start; n.next != null; n = n.next) {
  System.out.println(n.data);
}
```

**19.** After this loop finishes, `n` will refer to the last node in the nonempty list:

```
Node n;
for (n = list.start; n != null; n = n.next);
```

**20.** If `list` has more that one element, then this code will remove its last node:

```
Node n = list.start;
while (n.next.next != null) {
  n = n.next);
}
n.next = null;
```

**21.** This code creates a `String` with all of the data values in list:

```
String listStr = "";
for (n = list.start; n.next != null; n = n.next) {
  listStr += " " + n.data;
}
```

**22.** If `list` has more that one element, then this code will reverse its nodes:

```
Node p = list.start
Node q = p.next;
while (q.next != null) {
  p = q;
  q = q.next);
}
q.next.next = p;
```

**23.** This code makes a copy of the list:

```
LinkedList copy;
for (Node n = list.start; n != null; n = n.next) {
  copy.insert(n.data);
}
```

Questions 24-25 use the `BigInt` class defined on pages 142-143.

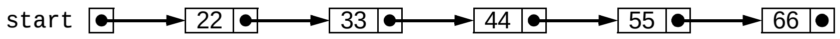
**24.** The most significant digit is the one in the node named `start`.

**25.** This code will print the value 0:

```
BigInt x=new BigInt(12345);
BigInt y=new BigInt(-12345);
BigInt sum = x.plus(y);
System.out.println(sum);
```

## Select the best answer:

In questions 1-4, assume that `start` is an instance of the `Node` class defined in Listing 4.10 on page 134 and it’s current state looks like this :



Also assume that this method is defined:

```
void print(Node n) {
  while (n != null) {
    System.out.print(n.data + “, “);
    n = n.next;
  }
}
```

**1.** What will the output be from this code:

```
start.next = start.next.next;
print(start);
```

**a.** `22, 33, 44, 55, 66`
**b.** `33, 44, 55, 66`
**c.** `22, 44, 55, 66`
**d.** `22, 33, 44, 66`
**e.** `22, 33, 44, 55`

**2.** What will the output be from this code:

```
start = start.next.next;
print(start);
```

**a.** `22, 33, 44, 55, 66`
**b.** `44, 55, 66`
**c.** `22, 55, 66`
**d.** `22, 33, 66`
**e.** `22, 33, 44`

**3.** What will the output be from this code:

```
start.next.next = null;
print(start);
```

**a.** `22, 33`
**b.** `33, 44`
**c.** `33, 22`
**d.** `22, 33, 44`
**e.** None of the above.

**4.** What will the output be from this code:

```
start.next.next = start;
print(start);
```

**a.** `22, 33`
**b.** `33, 44`
**c.** `33, 22`
**d.** `22, 33, 44`
**e.** None of the above.

## Answers to True/False Questions

**1.** True.

**2.** False.

**3.** False.

**4.** False.

**5.** True.

**6.** False.

**7.** False.

**8.** False.

**9.** False.

**10.** True.

**11.** True.

**12.** False.

**13.** True.

**14.** True.

**15.** True.

**16.** False.

**17.** True.

**18.** True.

**19.** False.

**20.** True.

**21.** True.

**22.** False.

**23.** True.

## Answers to Multiple Choice Questions

**1.** **c**

**2.** **b**

**3.** **a**

**4.** **e**

# Chapter 5
# Stacks

## Answer “True” or “False”:

**1.** A stack has at least these three operations: peek, pop, push.

**2.** A stack has two access points: front and back.

**3.** We use an abstract class to specify the operations of a stack.

**4.** The array implementation of a stack is superior to the linked list implementation because the array implementation's pop is faster.

**5.** The array implementation is inferior because we may have to recreate a larger array for the data.

**6.** The linked list implementation requires that an object keep track of both its current size and its capacity.

Questions 7-11 refer to this code:

```
Stack colors = new ArrayStack(8);
colors.push("blue");
colors.push("orange");
colors.push("purple");
colors.push("red");
Stack copy = new ArrayStack(8);
for (int i = 0; i < 4; i++){
  Object x = colors.peek();
  copy.push(x);
}
for (int i = 0; i < 4; i++){
  Object x = colors.pop();
  copy.pushx(x);
}
```

**7.** After line 3 has executed, the `size` of `colors` will be 2.

**8.** After line 10 has executed, the `size` of `copy` will be 4.

**9.** After lines 10 has executed, `copy` will have the same contents as `colors`.

**10.** After line 14 has executed, the `size` of `copy` will be 8.

**11.** After line 14 has executed, the `size` of `colors` will be 4.

**12.** The value of the postfix expression `7 3 + 9 4 – / 3 *` is 6.

**13.** The two postfix expressions `8 4 / 2 /` and `8 4 2 / /` have the same value.

**14.** The infix expression `2*2 - 4*2*24` has same value as the postfix expression: `2 2 * 4 2 * – 24 *`.

**15.** The postfix expression `8 4 * 2 *` has same value as infix `8 + (2 + 4)`.

Questions 16-20 refer to this code:

```
1   java.util.Stack s = new java.util.Stack();
2   System.out.println(s.empty());
3   for (int i = 0; i < 5; i++)
4     s.push(new Integer(i*10));
5   System.out.println(s.peek());
6   for (int i = 0; i < 5; i++)
7     s.push(s.pop());
8   java.util.Stack ss = new java.util.Stack();
9   while (!s.empty())
10    ss.push(s.pop());
11  System.out.println(ss.pop());
12  System.out.println(ss.peek());
13  System.out.println(ss.peek() + " " + s.peek());
```

**16.** The output from line 2 is `0`

**17.** The output from line 5 is `40`

**18.** The output from line 11 is `0`

**19.** The output from line 12 is `10`

**20.** The output from line 13 is `0 40`

## Select the best answer:

Questions 1-3 refer to this code:

```
Stack s = new LinkedStack(8);
s.push("A");
s.push("B");
s.push("C");
s.push("D");
s.push("E");
s.pop();
Object x = s.peek();
s.pop();
s.pop();
s.push(x);
```

**1.** What is the size of the stack `s` after all this code executes?

- **a.** 2
- **b.** 3
- **c.** 4
- **d.** 5
- **e.** None of these.

**2.** What is on the top of the stack `s` after all this code executes?

- **a.** `B`
- **b.** `C`
- **c.** `D`
- **d.** `E`
- **e.** None of these.

**3.** What is the second from the top element of the stack `s` after all this code executes?

- **a.** `B`
- **b.** `C`
- **c.** `D`
- **d.** `E`
- **e.** None of these.

Questions 4-8 refer to this code:

```
Stack s1 = new LinkedStack(8);
s1.push("A");
s1.push("B");
s1.push("C");
s1.push("D");
s1.push("E");
Stack s2 = new LinkedStack(8);
for (int i=0; i<3; i++) {
  s2.push(s1.pop());
}
```

```
Stack s3 = new LinkedStack(8);
while (s2.size() > 0) {
  s3.push(s2.pop());
}
while (s3.size() > 0) {
  s1.push(s3.pop());
}
```

**4.** What is the size of the stack `s1` after all of this code executes?

**a.** 0
**b.** 2
**c.** 3
**d.** 5
**e.** None of these.

**5.** What is the size of the stack `s2` after all of this code executes?

**a.** 0
**b.** 1
**c.** 2
**d.** 3
**e.** None of these.

**6.** What is the size of the stack `s3` after all of this code executes?

**a.** 0
**b.** 1
**c.** 2
**d.** 3
**e.** None of these.

**7.** What is on the top of the stack `s1` after after this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

**8.** What is on the top of the stack `s2` after after this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

**9.** What is the value of this postfix expression: `9 4 - 3 + 5 2 * 8 - /`
- **a.** 1
- **b.** 2
- **c.** 3
- **d.** 4
- **e.** None of these.

**10.** What is the value of this postfix expression: `1 2 3 * + 4 2 * 6 5 - - /`
- **a.** 1
- **b.** 2
- **c.** 3
- **d.** 4
- **e.** None of these.

## Answers to True/False Questions

**1.** True.

**2.** False.

**3.** False.

**4.** True.

**5.** True.

**6.** False.

**7.** True.

**8.** True.

**9.** False.

**10.** True.

**11.** False.

**12.** True.

**13.** False.

**14.** False.

**15.** True.

**16.** False.

**17.** True.

**18.** True.

**19.** True.

**20.** False.

## Answers to Multiple Choice Questions

**1.** **b**

**2.** **c**

**3.** **a**

**4.** **d**

**5.** **a**

**6.** **a**

**7.** **b**

**8.** **e**

**9.** **d**

**10.** **a**

# Chapter 6
# Queues

## Answer “True” or “False”:

**1.** A queue has at least these three operations: size, peek, pop, push.

**2.** A queue has two access points,  front and back.

**3.** A queue is a suitable structure to use for reversing the lines in a text file; just insert each line until end-of-file is reached. Then remove each line until the queue is empty.

**4.** The acronym "FIFO" stands for "Fast In Fast Out."

**5.** An array implementation of a queue is more difficult to manage than the array implementation of a stack.

**6.** The linked list implementation of a queue is similar to that for a stack: always insert at the start of the list.

**7.** The head `Node` in the `LinkedQueue` class defined on page 183 always contains the front object, i.e. the first one out.

Problems 8 through 11 refer to the following statements:

```
3    LinkedQueue meals = new LinkedQueue();
4    meals.add("breakfast");
5    meals.add("lunch");
6    meals.add("dinner");
7    System.out.println( meals.size() );
8    System.out.println( meals.first() );
9    meals.add(  meals.remove() );
10   System.out.println( meals.first());
11   for (int i = 0 ; i < meals.size(); i++)
12     System.out.print(" " + meals.first());
```

**8.** The output from line 7 is `4` because the queue will have 4 nodes including head.

**9.** The output from line 8 will be

```
breakfast
```

**10.** The output from line 10 will be

```
breakfast
```

**11.** The output from the loop on lines 11 and 12 will be:

```
lunch dinner breakfast.
```

Problems 12 through 16 refer to this method:

```
static void methdA(LinkedQueue q){
   if (q.isEmpty()) return;
   Object obj = q.remove();
   methdA(q);
   q.add(obj);
 }
```

**12.** The call `methdA(q)` will throw an exception unless `q` is nonempty.

**13.** The call `methdA(q)` reverses the order of the queue it is passed.

**14.** The call `methdA(q)` reverses the order of the queue it is passed except for the original first, whose place remains unchaged.

**15.** If parameter `q` has size 3, then the call `methdA(q)` will generate 2 more calls to `methdA`.

**16.** If parameter q has size 3, then the call `methdA(q)` will generate 3 more calls to `methdA`.

Problems 17 through 20 refer to this method:

```
static LinkedQueue methdB(LinkedQueue q){
  LinkedQueue qB = new LinkedQueue();
  int n = q.size();
  for (int i = 0; i < n; i++){
    Object obj = q.remove();
    qB.add( obj );
    q.add(obj);
  }
  return qB;
}
```

**17.** The call `methdB(q)` returns a `LinkedQueue` which equals `q` in size and contents.

**18.** The call `methdB(q)` returns a  whose size is the same as that of  but whose objects are in reverse order.

**19.** The call `methdB(q)` returns a `LinkedQueue` whose size and content are the same as original , but it reverses the objects in .

**20.** We can replace lines 22 and 23 with this single line without changing the effect of `methB()`:

```
  for(int i=0; i<q.size(); i++)
```

## Select the best answer:

Questions 1-3 refer to this code:

```
Queue q = new LinkedQueue(8);
q.add("A");
q.add("B");
q.add("C");
q.add("D");
q.add("E");
q.remove();
Object x = q.first();
q.remove();
q.remove();
q.add(x);
```

**1.** What is the size of the queue `q` after all this code executes?

**a.** 2
**b.** 3
**c.** 4
**d.** 5
**e.** None of these.

**2.** What is the first element of the queue `q` after all this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

**3.** What is the last element of the queue `q` after all this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

Questions 4-7 refer to this code:

```
Queue q = new LinkedQueue(8);
q.add("A");
```

```
q.add("B");
q.add("C");
q.add("D");
q.add("E");
Stack s = new LinkedStack(8);
for (int i=0; i<3; i++) {
  s.push(q.remove());
}
while (s.size() > 0) {
  q.add(pop());
}
```

**4.** What is the size of the queue `q` after all of this code executes?

- **a.** 0
- **b.** 2
- **c.** 3
- **d.** 5
- **e.** None of these.

**5.** What is the size of the stack `s` after all of this code executes?

- **a.** 0
- **b.** 1
- **c.** 2
- **d.** 3
- **e.** None of these.

**6.** What is the first element of the queue `q` after all this code executes?

- **a.** `B`
- **b.** `C`
- **c.** `D`
- **d.** `E`
- **e.** None of these.

**7.** What is the last element of the queue `q` after all this code executes?

- **a.** `B`
- **b.** `C`
- **c.** `D`
- **d.** `E`
- **e.** None of these.

Questions 8-10 refer to this code:

```
Queue q1 = new LinkedQueue(8);
q1.add("A");
q1.add("B");
q1.add("C");
q1.add("D");
```

```
q1.add("E");
Queue q2 = new LinkedQueue(8);
q2.add("X");
q2.add("Y");
q2.add("Z");
Queue q3 = new LinkedQueue(8);
while (q2.size() > 0) {
  q3.add(q1.remove());
  q3.add(q2.remove());
}
for (int i=0; i<3; i++) {
  q1.add(q3.remove());
}
```

**8.** What is the size of the queue `q1` after all of this code executes?

**a.** 0
**b.** 2
**c.** 3
**d.** 5
**e.** None of these.

**9.** What is the first element of the queue `q1` after all this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

**10.** What is the last element of the queue `q1` after all this code executes?

**a.** `B`
**b.** `C`
**c.** `D`
**d.** `E`
**e.** None of these.

## Answers to True/False Questions

**1.** False.

**2.** True.

**3.** False.

**4.** True.

**5.** True.

**6.** False.

7. False.
8. False.
9. True.
10. False.
11. True.
12. False.
13. True.
14. False.
15. False.
16. True.
17. True.
18. False.
19. False.
20. True.

## Answers to Multiple Choice Questions

1. b
2. c
3. a
4. d
5. a
6. c
7. e
8. d
9. c
10. b

# Chapter 7 Collections

**Answer "True" or "False":**

**1.** The type `java.util.Collection` is an interface.

**2.** A `Collection` object may have more than one iterator acting on it.

**3.** A prefix representation of an arithmetic expression can be ambiguous.

**4.** An infix representation of an arithmetic expression can be ambiguous.

**5.** An arithmetic expression in pastfix notation can be evaluated using a stack.

**6.** If `x` and `y` are references to objects of the same type, then the expression `x == y` will evaluate to `true` if and only if `x` and `y` refer to the same object.

**7.** If `x` and `y` are references to objects of the same type, then the expression `x.equals(y)` will evaluate to `true` if and only if `x` and `y` refer to the same object.

**8.** If `x` and `y` are references to objects of the same type, then the assignment `y = x` will result in `x` and `y` referring to the same object.

**9.** A `List` object can contain duplicate elements.

**10.** A `Set` object can contain duplicate elements.

**11.** A `Map` object can contain duplicate keys.

**12.** Iterators can be used in insert and remove elements in a collection.

**13.** An abstract class cannot be instantiated.

**14.** The `addAll()` method in the `java.util.Collection` interface represents the union operation for sets.

**15.** The `removeAll()` method in the `java.util.Collection` interface represents the intersection operation for sets.

**16.** The `retainAll()` method in the `java.util.Collection` interface represents the relative complement operation for sets.

## Select the best answer:

For each of questions 1-4, assume that this code has already executed:

```
String[] states = {"NY", "CA", "TX", "VA", "MI", "FL"};
ArrayList list = new ArrayList(Arrays.asList(stgates));
Iterator it = list.iterator();
```

**1.** What will this output:

```
it.next();
System.out.println(it.next());
```

**a.** `NY`
**b.** `CA`
**c.** `MI`
**d.** `FL`
**e.** None of these.

**2.** What will this output:

```
it.next();
System.out.println(it.remove());
```

**a.** `NY`
**b.** `CA`
**c.** `MI`
**d.** `FL`
**e.** None of these.

**3.** What will this output:

```
it.remove();
System.out.println(it.next());
```

**a.** `NY`
**b.** `CA`
**c.** `MI`
**d.** `FL`
**e.** None of these.

**4.** What will this output:

```
it.next();
System.out.println(remove("TX"));
```

**a.** `TX`
**b.** `CA`
**c.** `true`
**d.** `false`
**e.** None of these.

**5.** Which method in the `java.util.Collection` interface represents the union operation for sets?

**a.** `addAll()`
**b.** `removeAll()`
**c.** `retainAll()`
**d.** `containsll()`
**e.** None of these.

**6.** Which method in the `java.util.Collection` interface represents the intersection operation for sets?

**a.** `addAll()`
**b.** `removeAll()`
**c.** `retainAll()`
**d.** `containsll()`
**e.** None of these.

**7.** Which method in the `java.util.Collection` interface represents the subset operation for sets?

**a.** `addAll()`
**b.** `removeAll()`
**c.** `retainAll()`
**d.** `containsll()`
**e.** None of these.

**8.** Which method in the `java.util.Collection` interface represents the relative complements operation for sets?

**a.** `addAll()`
**b.** `removeAll()`
**c.** `retainAll()`
**d.** `containsll()`
**e.** None of these.

## Answers to True/False Questions

1. True.
2. True.
3. False.
4. True.
5. True.
6. True.
7. False.
8. True.
9. True.
10. False.
11. False.
12. True.
13. True.
14. True.
15. False.
16. False.

## Answers to Multiple Choice Questions

1. b
2. e
3. b
4. c
5. a
6. c
7. d
8. b

# Chapter 8
# Lists

**Answer "True" or "False":**

**1.** The `java.util.List` type can be implemented with either a linked list or an array.

**2.** The `java.util.List` type is an interface that extends the `java.util.Collection` interface.

**3.** The `java.util.ListIterator` type is an interface that extends the `java.util.List` interface.

**4.** The `java.util.ListIterator` type specifies bi-directional access for accessing and modifying `java.util.List` objects.

**5.** If `list` is any object of type `java.util.List`, then `x.listIterator()` will return an iterator of `java.util.ListIterator` type that can access the elements of `list`.

**6.** If `it` has type `java.util.ListIterator`, then the statement

```
  x = it.next();
```

will assign to `x` the current element located by `it` and then move `it` to the next element (if it exists).

**7.** If `it` has type `java.util.ListIterator`, then the statement

```
  x = it.previous();
```

will assign to `x` the current element located by `it` and then move `it` to the previous element (if it exists).

**8.** Objects of type `java.util.ArrayList` perform lookups faster than objects of type `java.util.LinjkedList`.

**9.** Objects of type `java.util.ArrayList` perform insertions faster than objects of type `java.util.LinjkedList`.

**10.** Objects of type `java.util.ArrayList` perform deletions faster than objects of type `java.util.LinjkedList`.

## Select the best answer:

For each of questions 1-4, assume that this code has already executed:

```
String[] states = {"NY", "CA", "MI", "FL"};
ArrayList list = new ArrayList(Arrays.asList(states));
ListIterator it = list.listIterator();
```

**1.** What will this output:

```
System.out.println(it.previous());
```

**a.** `NY`
**b.** `CA`
**c.** `MI`
**d.** `FL`
**e.** None of these.

**2.** What will this output:

```
it.next();
System.out.println(it.previous());
```

**a.** `NY`
**b.** `CA`
**c.** `MI`
**d.** `FL`
**e.** None of these.

**3.** What will this output:

```
it.next();
it.next();
it.previous();
it.set("XX");
System.out.println(list);
```

**a.** `[NY, CA, MI, FL]`
**b.** `[XX, CA, MI, FL]`
**c.** `[NY, XX, MI, FL]`
**d.** `[NY, CA, XX, FL]`
**e.** None of these.

**4.** What will this output:

```
it.next();
it.next();
it.previous();
it.set("XX");
```

```
System.out.println(list);
```

**a.** `[NY, CA, MI, FL]`
**b.** `[XX, CA, MI, FL]`
**c.** `[NY, XX, MI, FL]`
**d.** `[NY, CA, XX, FL]`
**e.** None of these.

For each of questions 5-7, assume that all of this code has executed:

```
String[] args = {"A", "B", "C", "D", "E", "F", "G"};
ArrayList list = new ArrayList(Arrays.asList(args));
list.add(list.get(2), 4);
list.remove(6);
```

**5.** What will the size of the list be?
**a.** 0
**b.** 6
**c.** 7
**d.** 8
**e.** None of these.

**6.** What would `list.get(4)` return?
**a.** `C`
**b.** `D`
**c.** `E`
**d.** `F`
**e.** `G`

**7.** What would `list.get(6)` return?
**a.** `C`
**b.** `D`
**c.** `E`
**d.** `F`
**e.** `G`

For each of questions 8-10, assume that all of this code has executed:

```
String[] args = {"A", "B", "C", "D", "E", "F", "G"};
ArrayList list = new ArrayList(Arrays.asList(args));
ListIterator it = list.listIterator();
it.next();
it.next();
it.next();
it.next();
it.previous();
it.set("X");
it.previous();
it.add("Y");
it.next();
it.set("Z");
it.previous();
it.previous();
```

**8.** What would `it.next()` return?
**a.** `C`
**b.** `D`
**c.** `X`
**d.** `Y`
**e.** None of these.

**9.** What would `list.get(4)` return?
**a.** `C`
**b.** `D`
**c.** `X`
**d.** `Y`
**e.** None of these.

**10.** What would `list.get(4)` return?
**a.** `C`
**b.** `D`
**c.** `X`
**d.** `Y`
**e.** None of these.

## Answers to True/False Questions

**1.** True.

**2.** True.

**3.** False.

**4.** True.

**5.** True.

**6.** True.

**7.** False.

**8.** True.

**9.** False.

**10.** False.

## Answers to Multiple Choice Questions

| | |
|---|---|
| 1. | e |
| 2. | a |
| 3. | c |
| 4. | e |
| 5. | c |
| 6. | a |
| 7. | e |
| 8. | d |
| 9. | c |
| 10. | e |

# Chapter 9
# Hash Tables

**Answer “True” or “False”:**

1. Hash tables are almost as efficient as arrays in performing lookups in almost constant time.
2. Hash tables are almost as efficient as linked lists in performing insertions and deletions in almost constant time.
3. Hash tables have the advantage of maintaining the order of its elements’ keys.
4. A hash function works like an array index.
5. A hash function should be independent of the capacity of the hash table.
6. A hash table *collision* is where two equal objects have different hash values.
7. Linear probing, quadratic probing, and double hashing are all forms of open addressing in hash tables.
8. The run-time of linear probing is independent of the hash table’s load factor.
9. Double hashing requires defining two independent hash functions.
10. Double hashing is used to overcome secondary clustering.
11. The `java.util.HashMap` class uses double hashing.
12. Rehashing requires rebuilding the hash table with a larger backing array.
13. Separate chaining describes a hash table whose backing array contains linked lists.
14. A perfect hash function is one that has no collisions.
15. A minimal perfect hash function has a 100% load factor.

## Select the best answer:

For each of questions 1-9, assume that keys from this table (taken from Table 9.3 on page 271) are being inserted into a hash table with capacity 9, using the hash values `h` shown here, computed by the hash function.

```
h = key.hashCode() % 9
```

| key | key.hashCode() | h |
|---|---|---|
| AT | 2099 | 2 |
| FR | 2252 | 2 |
| DE | 2177 | 8 |
| GR | 2283 | 6 |
| IT | 2347 | 7 |
| PT | 2564 | 8 |
| SE | 2642 | 5 |
| SG | 2644 | 7 |

**1.** Using linear probing, at which index will `SG` be put, after having inserted `IT` and `PT`?

- **a.** `7`
- **b.** `8`
- **c.** `0`
- **d.** `1`
- **e.** None of these.

**2.** Using linear probing, at which index will `DE` be put, after having inserted `IT`, `PT`, and `SG`?

- **a.** `7`
- **b.** `8`
- **c.** `0`
- **d.** `1`
- **e.** None of these.

**3.** Using linear probing, at which index will `FR` be put, after having inserted `IT`, `PT`, `SG`, and `DE`?

- **a.** `7`
- **b.** `8`
- **c.** `0`
- **d.** `1`
- **e.** None of these.

**4.** Using quadratic probing, at which index will `SG` be put, after having inserted `IT` and `PT`?

- **a.** `0`
- **b.** `1`
- **c.** `2`
- **d.** `3`
- **e.** None of these.

**5.** Using linear probing, at which index will `DE` be put, after having inserted `IT`, `PT`, and `SG`?

**a.** `0`
**b.** `1`
**c.** `2`
**d.** `3`
**e.** None of these.

**6.** Using linear probing, at which index will `FR` be put, after having inserted `IT`, `PT`, `SG`, and `DE`?

**a.** `0`
**b.** `1`
**c.** `2`
**d.** `3`
**e.** None of these.

For each of questions 7-9, assume that keys from this table, using the second hash function `h2`, defined by:

```
h2 = key.hashCode() % 11
```

| key | key.hashCode() | h | h2 |
|---|---|---|---|
| AT | 2099 | 2 | 9 |
| FR | 2252 | 2 | 8 |
| DE | 2177 | 8 | 10 |
| GR | 2283 | 6 | 6 |
| IT | 2347 | 7 | 4 |
| PT | 2564 | 8 | 1 |
| SE | 2642 | 5 | 2 |
| SG | 2644 | 7 | 4 |

**7.** Using double hashing, at which index will `SG` be put, after having inserted `IT` and `PT`?

**a.** `0`
**b.** `1`
**c.** `2`
**d.** `3`
**e.** None of these.

**8.** Using double hashing, at which index will `DE` be put, after having inserted `IT`, `PT`, and `SG`?

**a.** `0`
**b.** `1`
**c.** `2`
**d.** `3`
**e.** None of these.

**9.** Using double hashing, at which index will `FR` be put, after having inserted `IT`, `PT`, `SG`, and `DE`?

**a.** `0`
**b.** `1`
**c.** `2`
**d.** `3`
**e.** None of these.

## Answers to True/False Questions

1. True.
2. True.
3. False.
4. True.
5. False.
6. False.
7. True.
8. False.
9. True.
10. True.
11. False.
12. True.
13. True.
14. True.
15. True.

## Answers to Multiple Choice Questions

1. c
2. d
3. e
4. c
5. a
6. d
7. c
8. a
9. b

# Chapter 10
# Recursion

**Answer "True" or "False":**

1. A recursive method is one that invokes itself. iterative method.
2. A recursive method is likely to be more succinct than its equivalent iterative method.
3. A recursive method is likely to use more variables than its equivalent iterative method.
4. A recursive method is likely to use more space than its equivalent iterative method.
5. A recursive method is likely to be easier to debug than its equivalent iterative method.
6. A recursive method is likely to run faster than its equivalent iterative method.
7. A recursive method must include a base case to stop the recursion.
8. The Fibonacci method given Listing 10.3 on page 304 is so slow because it makes two direct recursive calls.
9. The recursive Fibonacci method (Listing 10.3 on page 304) runs in exponential time.
10. De Moivre's formula for computing the Fibonacci numbers is only approximately correct.
11. The recursive exponentiation method (page 315) uses the "divide and conquer" strategy..
12. The recursive exponentiation method (page 315) runs in logarithmic time.
13. The recursive permutation method (Listing 10.8 on page 316) runs in linear time.
14. The method method for printing permutations is simpler than the iterative method.

**15.** A calling tree is a graphical representation that shows all the recursive calls made by a recursive method.

## Select the best answer:

**1.** Under which circumstances should a recursive solution be preferred to an iterative solution?

- **i.** When the recursive version is more easily understood.
- **ii.** When the running time is critical.
- **iii.** When space (memory) is critical.

**a.** None of the above.
**b.** Only **i**.
**c.** Only **i** and **ii**.
**d.** Only **i** and **iii**.
**e.** All of the above.

**2.** What is the asymptotic run-time complexity for the recursive factorial method (Listing 10.1 on page 299)?

**a.** logarithmic
**b.** linear
**c.** quadratic
**d.** exponential
**e.** None of these.

**3.** How many recursive calls would `factorial(5)` make using the implementation given in Listing 10.1 on page 299?

**a.** 3
**b.** 4
**c.** 5
**d.** 8
**e.** 25

**4.** What is the asymptotic run-time complexity for the recursive Towers of Hanoi method (Listing 10.2 on page 302)?

**a.** logarithmic
**b.** linear
**c.** quadratic
**d.** exponential
**e.** None of these.

**5.** How many recursive calls would `print(5, 'A', 'B', 'C')` make using the implementation given in Listing 10.2 on page 302?

**a.** 5
**b.** 25
**c.** 30
**d.** 120
**e.** 125

**6.** What is the asymptotic run-time complexity for the recursive Fibonacci method (Listing 10.3 on page 304)?

**a.** logarithmic
**b.** linear
**c.** quadratic
**d.** exponential
**e.** None of these.

**7.** How many recursive calls would `f(5)` make using the implementation given in Listing 10.3 on page 304?

**a.** 3
**b.** 4
**c.** 5
**d.** 8
**e.** 25

**8.** What is the asymptotic run-time complexity for the recursive Binary Search (Algorithm 10.1 on page 313)?

**a.** logarithmic
**b.** linear
**c.** quadratic
**d.** exponential
**e.** None of these.

**9.** How many recursive calls would the Binary Search (Algorithm 10.1 on page 313) make in searching for the first element in a sequence of 100 elements?

**a.** 3
**b.** 4
**c.** 5
**d.** 8
**e.** 25

**10.** What is the asymptotic run-time complexity for the recursive exponentiation method (on page 315)?

- **a.** logarithmic
- **b.** linear
- **c.** quadratic
- **d.** exponential
- **e.** None of these.

**11.** How many recursive calls would `exp(100)` make using the implementation given on page 315?

- **a.** 3
- **b.** 4
- **c.** 5
- **d.** 8
- **e.** 25

**12.** What is the asymptotic run-time complexity for the recursive permutation method (Listing 10.8 on page 316)?

- **a.** logarithmic
- **b.** linear
- **c.** quadratic
- **d.** exponential
- **e.** None of these.

**13.** How many recursive calls would be made from `print("ABCDE")` using the implementation given in Listing 10.8 on page 316?

- **a.** 5
- **b.** 25
- **c.** 30
- **d.** 120
- **e.** 125

**14.** What is the asymptotic run-time complexity for the mutually recursive sine method (Listing 10.9 on page 320)?

- **a.** logarithmic
- **b.** linear
- **c.** quadratic
- **d.** exponential
- **e.** None of these.

**15.** How many multiplications would be made from the call `sin(0.032)` using the implementation given in Listing 10.9 on page 320?

- **a.** 6
- **b.** 9
- **c.** 12
- **d.** 24
- **e.** 32

## Answers to True/False Questions

**1.** True.

**2.** True.

**3.** False.

**4.** True.

**5.** False.

**6.** False.

**7.** True.

**8.** True.

**9.** True.

**10.** False.

**11.** True.

**12.** True.

**13.** False.

**14.** False.

**15.** True.

## Answers to Multiple Choice Questions

**1.** **b**

**2.** **b**

**3.** **b**

**4.** **d**

| | |
|---|---|
| 5. | c |
| 6. | d |
| 7. | d |
| 8. | a |
| 9. | c |
| 10. | a |
| 11. | d |
| 12. | d |
| 13. | d |
| 14. | e |
| 15. | b |

# Chapter 11
# Trees

**Answer "True" or "False":**

**1.** A tree is a linear data structure.

**2.** The depth of a node in a tree is equal to the number of its ancestors.

**3.** The size of a subtree is equal to the number of descendants of the root of the subtree.

**4.** If $x$ is a descendant of $y$ in a tree, then the depth of $x$ is greater than the depth of $y$.

**5.** If the depth of $x$ is greater than the depth of $y$ in a tree, then $x$ is a descendant of $y$.

**6.** A tree is a singleton if and only if its root is a leaf.

**7.** Every leaf of a subtree is also a leaf of its supertree.

**8.** The root of a subtree is also the root of its supertree.

**9.** If $R$ is a subtree of $S$ and $S$ is a subtree of $T$, then $R$ is a subtree of $T$.

**10.** A node is a leaf if and only if it has degree 0.

**11.** In any tree, the number of internal nodes must be less than the number of leaf nodes.

**12.** A tree is full if and only if all of its leaves are at the same level.

**13.** Every subtree of a full binary tree is full.

**14.** Every subtree of a complete binary tree is complete.

**15.** The height of a tree is the length of its longest root-to-leaf path.

**16.** The height of a tree is its deepest level.

**17.** If $x$ is a descendant of $y$, and $y$ is a descendant of $z$, then $x$ is a descendant of $z$.

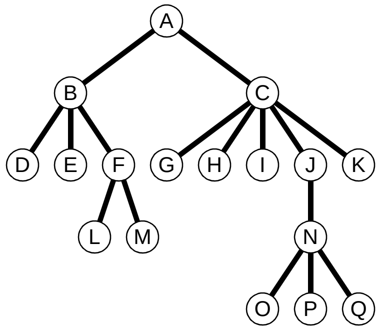
**18.** If *x* is a descendant of *y*, and *x* is a descendant of *z*, then *y* is a descendant of *z*.

**19.** If *x* is a descendant of *y*, and *z* is a descendant of *y*, then *x* is a descendant of *z*.

**20.** The degree of a tree is the same as the degree of its root node.

## Select the best answer:

Questions 1–8 refer to the tree shown here:



**1.** What is the size of this tree?
- **a.** 4
- **b.** 5
- **c.** 9
- **d.** 15
- **e.** 17

**2.** What is the height of this tree?
- **a.** 4
- **b.** 5
- **c.** 9
- **d.** 15
- **e.** 17

**3.** What is the degree of this tree?
- **a.** 4
- **b.** 5
- **c.** 9
- **d.** 15
- **e.** 17

**4.** What is the size of the subtree rooted at J?
- **a.** 1
- **b.** 3
- **c.** 4
- **d.** 5
- **e.** 9

**5.** How many descendants does node J have?
- **a.** 1
- **b.** 3
- **c.** 4
- **d.** 5
- **e.** 9

**6.** How many nodes are at level 2?

**a.** 1
**b.** 2
**c.** 3
**d.** 5
**e.** 8

**7.** What is the width of this tree?

**a.** 1
**b.** 2
**c.** 3
**d.** 5
**e.** 8

**8.** How long is the root path to node L?

**a.** 1
**b.** 2
**c.** 3
**d.** 4
**e.** 5

**9.** How many nodes are in the full tree of degree 3 and height 4?

**a.** 21
**b.** 40
**c.** 81
**d.** 83
**e.** 121

**10.** What is the height of a complete tree of degree 3 and size 100?

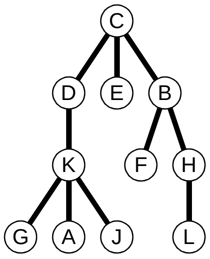
**a.** 3
**b.** 4
**c.** 5
**d.** 9
**e.** 33

**11.** What is the smallest possible height of any tree of degree 3 and size 100?

**a.** 3
**b.** 4
**c.** 5
**d.** 9
**e.** 33

**12.** What is the largest possible height of any tree of degree 3 and size 100?
- **a.** 4
- **b.** 33
- **c.** 96
- **d.** 97
- **e.** 99

Questions 13–15 refer to the tree shown here:



**13.** What is the level order traversal of this tree?
- **a.** C D K G A J E B F H L
- **b.** G A J K D E F L H B C
- **c.** C D E B K G A J F H L
- **d.** C D E B K F H G A J L
- **e.** G A J L K F H D E B C

**14.** What is the preorder traversal of this tree?
- **a.** C D K G A J E B F H L
- **b.** G A J K D E F L H B C
- **c.** C D E B K G A J F H L
- **d.** C D E B K F H G A J L
- **e.** G A J L K F H D E B C

**15.** What is the postorder traversal of this tree?
- **a.** C D K G A J E B F H L
- **b.** G A J K D E F L H B C
- **c.** C D E B K G A J F H L
- **d.** C D E B K F H G A J L
- **e.** G A J L K F H D E B C

**16.** Which traversal algorithm(s) always visit the root first?
- **a.** Level Order
- **b.** Preorder
- **c.** Postorder
- **d.** Level Order and Preorder
- **e.** All three algorithms

**17.** Which traversal algorithm(s) are recursive?
- **a.** Level Order
- **b.** Preorder
- **c.** Postorder
- **d.** Preorder and Postorder
- **e.** All three algorithms

**18.** In an ordered tree of degree 3, what is the index number of the parent of the node numbered 30?

**a.** 8
**b.** 9
**c.** 10
**d.** 20
**e.** 29

**19.** In an ordered tree of degree 3, what is the index number of the first child of the node numbered 30?

**a.** 31
**b.** 60
**c.** 90
**d.** 91
**e.** 92

**20.** In an ordered tree of degree 3, what is the index number of the last child of the node numbered 30?

**a.** 33
**b.** 63
**c.** 91
**d.** 92
**e.** 93

## Answers to True/False Questions

**1.** False

**2.** True.

**3.** False.

**4.** True.

**5.** False.

**6.** True.

**7.** True.

**8.** False.

**9.** True.

**10.** True.

**11.** False.

| | |
|---|---|
| **12.** | False. |
| **13.** | True. |
| **14.** | True. |
| **15.** | True. |
| **16.** | True. |
| **17.** | True. |
| **18.** | False. |
| **19.** | False. |
| **20.** | False. |

## Answers to Multiple Choice Questions

| | |
|---|---|
| **1.** | **e** |
| **2.** | **a** |
| **3.** | **b** |
| **4.** | **d** |
| **5.** | **c** |
| **6.** | **e** |
| **7.** | **e** |
| **8.** | **c** |
| **9.** | **e** |
| **10.** | **b** |
| **11.** | **b** |
| **12.** | **d** |
| **13.** | **d** |
| **14.** | **a** |
| **15.** | **b** |
| **16.** | **d** |
| **17.** | **d** |

**18.** c

**19.** d

**20.** e

# Chapter 12
# Binary Trees

**Answer “True” or “False”:**

**1.** There are five different binary trees of size 3.

**2.** The number of internal nodes of a binary tree cannot be less than its number of leaves.

**3.** The number of internal nodes of a binary tree cannot be more than its number of leaves.

**4.** The number of internal nodes of a binary tree cannot equal number of leaves.

**5.** The number of leaves in a binary tree cannot be less than its height.

**6.** The number of leaves in a binary tree cannot be greater than its height.

**7.** The number of leaves in a binary tree cannot equal its height.

**8.** The number of leaves in a full binary tree of height $h$ is $2^h$.

**9.** The number of internal nodes in a full binary tree of height $h$ is $2^h$.

**10.** The length of each root-to-leaf path in a complete binary tree of height $h$ is either $h$ or $h - 1$.

**11.** In the prefix representation of an arithmetic expression, each operator precedes its operands.

**12.** In the postfix representation of an arithmetic expression, each operator follows its operands.

**13.** In the preorder traversal of any binary tree, the root is always visited first.

**14.** In the inorder traversal of any binary tree, the root is never visited last.

**15.** In the postorder traversal of any binary tree, the root is always visited last.

**16.** In the inorder traversal of any binary tree, the node that is farthest from the root is always visited last.

**17.** If *T* is the binary tree that represents an ordered forest *F*, then the number of disjoint trees in *F* is equal to the height of *T*.

**18.** If *T* is the binary tree that represents an ordered forest *F*, then the number of disjoint trees in *F* is equal to the length of the path from the root of *T* to its right-most node.

**19.** If *T* is the binary tree that represents an ordered forest *F*, then the number of siblings of a node in *F* is equal to the shortest path from the corresponding node in *T* to a leaf.

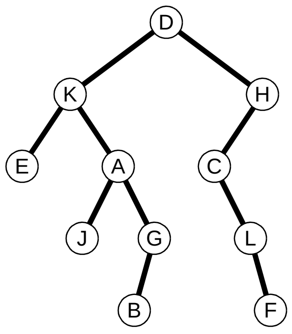
**20.** If *T* is the binary tree that represents an ordered forest *F*, then the number of disjoint trees in *F* is equal to the length of the root-path in *T* to its right-most node.

## Select the best answer:

**1.** How many different binary trees of size 4 are there?
- **a.** 5
- **b.** 9
- **c.** 14
- **d.** 15
- **e.** 17

**2.** What is the size of the full binary tree of height 3?
- **a.** 5
- **b.** 9
- **c.** 14
- **d.** 15
- **e.** 17

Questions 3–6 refer to the tree shown here:



**3.** What is the level order traversal of this tree?
- **a.** D K E A J G B H C L F
- **b.** E J B G A K F L C H D
- **c.** E K J A B G D H C L F
- **d.** D K H E A C J G L B F
- **e.** E K J A B G D C L F H

**4.** What is the preorder traversal of this tree?

**a.** D K E A J G B H C L F
**b.** E J B G A K F L C H D
**c.** E K J A B G D H C L F
**d.** D K H E A C J G L B F
**e.** E K J A B G D C L F H

**5.** What is the inorder traversal of this tree?

**a.** D K E A J G B H C L F
**b.** E J B G A K F L C H D
**c.** E K J A B G D H C L F
**d.** D K H E A C J G L B F
**e.** E K J A B G D C L F H

**6.** What is the postorder traversal of this tree?

**a.** D K E A J G B H C L F
**b.** E J B G A K F L C H D
**c.** E K J A B G D H C L F
**d.** D K H E A C J G L B F
**e.** E K J A B G D C L F H

**7.** In a complete binary tree, what is the index number of the parent of the node numbered 20?

**a.** 8
**b.** 9
**c.** 10
**d.** 11
**e.** 19

**8.** In a complete binary tree, what is the index number of the left child of the node numbered 20?

**a.** 21
**b.** 40
**c.** 41
**d.** 42
**e.** 43

**9.** In a complete binary tree, what is the index number of the right child of the node numbered 20?

- **a.** 21
- **b.** 22
- **c.** 41
- **d.** 42
- **e.** 43

**10.** How many nodes are in the full binary tree of height 5?

- **a.** 6
- **b.** 10
- **c.** 31
- **d.** 63
- **e.** 64

**11.** Which of these could not be the size of any binary tree of height 5?

- **a.** 6
- **b.** 10
- **c.** 31
- **d.** 63
- **e.** 64

**12.** What is the smallest possible size of a complete binary tree of height 5?

- **a.** 32
- **b.** 33
- **c.** 63
- **d.** 64
- **e.** 65

**13.** What is the smallest possible height of a binary tree of size 100?

- **a.** 4
- **b.** 5
- **c.** 6
- **d.** 7
- **e.** 10

**14.** What is the largest possible height of a binary tree of size 100?

- **a.** 6
- **b.** 7
- **c.** 15
- **d.** 99
- **e.** 100

**15.** What is the smallest possible height of a complete binary tree of size 100?

**a.** 5
**b.** 6
**c.** 7
**d.** 8
**e.** 10

**16.** What is the largest possible height of a complete binary tree of size 100?

**a.** 6
**b.** 7
**c.** 8
**d.** 99
**e.** 100

**17.** Evaluate this prefix expression: `- * 3 + 2 1 * 2 - 5 1`

**a.** 1
**b.** 2
**c.** 3
**d.** 4
**e.** 6

**18.** Evaluate this postfix expression: `2 4 1 + * 3 7 4 - * -`

**a.** 1
**b.** 2
**c.** 3
**d.** 4
**e.** 6

**19.** What is the prefix expression for `(7 - 5)*2 + 9/(4 - 1)` ?

**a.** `+ - 7 5 * 2 / 9 - 4 1`
**b.** `+ * - 7 5 2 / 9 - 4 1`
**c.** `7 5 - 2 * 9 4 1 - / +`
**d.** `7 5 - 2 * 9 / 4 1 - +`
**e.** `7 - 5 * 2 + 0 / 4 - 1`

**20.** What is the postfix expression for `(7 - 5)*2 + 9/(4 - 1)` ?

**a.** `+ - 7 5 * 2 / 9 - 4 1`
**b.** `+ * - 7 5 2 / 9 - 4 1`
**c.** `7 5 - 2 * 9 4 1 - / +`
**d.** `7 5 - 2 * 9 / 4 1 - +`
**e.** `7 - 5 * 2 + 0 / 4 - 1`

## Answers to True/False Questions

1. True.
2. False.
3. False.
4. False.
5. False.
6. False.
7. False.
8. True.
9. False.
10. True.
11. True.
12. True.
13. True.
14. False.
15. True.
16. False.
17. False.
18. True.
19. False.
20. False.

## Answers to Multiple Choice Questions

1. c
2. d
3. d
4. a

| | |
|---|---|
| 5. | e |
| 6. | b |
| 7. | b |
| 8. | c |
| 9. | d |
| 10. | d |
| 11. | e |
| 12. | a |
| 13. | c |
| 14. | d |
| 15. | b |
| 16. | a |
| 17. | a |
| 18. | a |
| 19. | b |
| 20. | c |

# Chapter 13
# Search Trees

**Answer “True” or “False”:**

**1.** In a search tree, the elements’ key field is an instance variable that uniquely identifies the elements.

**2.** In a search tree, the key value at each node is greater than the keys in its subtrees.

**3.** In a multiway search tree, the keys in each node are in increasing order.

**4.** Binary search trees are efficient because they guarantee $\Theta(\lg n)$ access time.

**5.** AVL trees are efficient because they guarantee $\Theta(\lg n)$ access time.

**6.** Red-black trees are efficient because they guarantee $\Theta(\lg n)$ access time.

**7.** Every subtree of a BST is another BST.

**8.** If both the left and right subtrees of a binary tree $T$ are BSTs, then $T$ must be a BST.

**9.** If the same keys are inserted in a different order to build a BST, the resulting BST will be the same.

**10.** Every subtree of an AVL tree is another AVL tree.

**11.** If $T$ is a BST and both the left and right subtrees of $T$ are AVL trees, then $T$ must be an AVL tree.

**12.** Fibonacci trees are worst-case AVL trees.

**13.** All the leaves of a B-tree are at the same level.

**14.** A 2-3-4 tree is a search tree of degree 4.

**15.** A red-black tree is a search tree of degree 4.

**16.** Every red-black tree is naturally represented by a unique 2-3-4 tree.

**17.** Every 2-3-4 tree is uniquely represented by a unique red-black tree.

**18.** The black height of a red-black tree is equal to the height of the 2-3-4 tree that it represents.

**19.** In a B-tree of order 8, the root node must have at least 4 subtrees.

**20.** The height of a B-tree of order 8 that contains 100,000 keys is less than 8.

## Select the best answer:

**1.** How many different binary search trees could result from inserting the three keys `A`, `B`, and `C` in different orders?

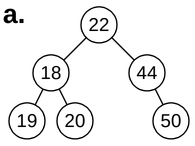
**a.** 2
**b.** 3
**c.** 4
**d.** 5
**e.** 6

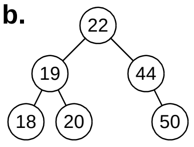
**2.** If the seven keys `A`, `B`, `C`, `D`, `E`, `F`, and `G` have been inserted into a binary search tree and `C` is in node *x*, then where could `D` be?

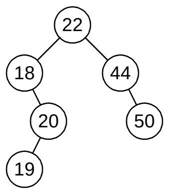
**i.** The parent of *x*.
**ii.** The grandparent of *x*.
**iii.** The right child of *x*.
**iv.** The left child of the right child of *x*.
**v.** The right child of the right child of *x*.

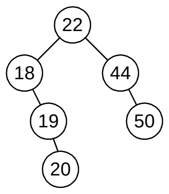
**a.** Only **i**.
**b.** Only **i** and **ii**.
**c.** Only **i**, **ii**, and **iii**.
**d.** Only **i**, **ii**, **iii**, and **iv**.
**e.** All of the above.

**3.** Which of these is not a binary search tree?

**a.**


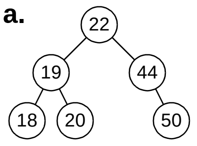
**b.**


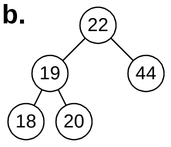
**c.**


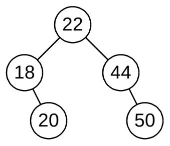
**d.**


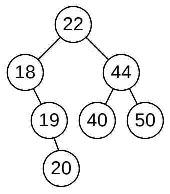
**e.** None of these.

**4.** Which of these is not an AVL tree?

**a.**


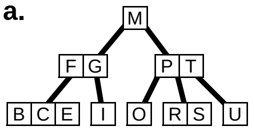
**b.**


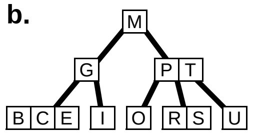
**c.**


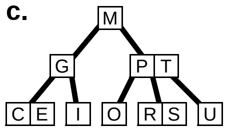
**d.**


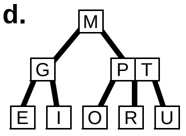
**e.** None of these.

**5.** Which of these is not a 2-3-4 tree?

**a.**


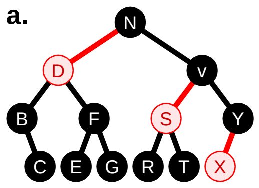
**b.**


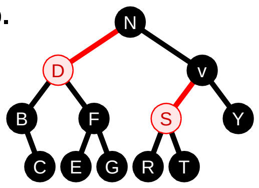
**c.**


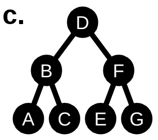
**d.**


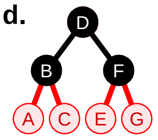
**e.** None of these.

**6.** Which of these is not a red-black tree?

**a.**


**b.**


**c.**


**d.**


**e.** None of these.

**7.** In a B-tree of degree 8 and height 3, which of these could not be the number of keys:

**i.** 50
**ii.** 100
**iii.** 100
**iv.** 4000

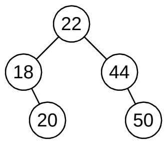
**a.** Only **i**.
**b.** Only **i** and **ii**.
**c.** Only **i** and **iv**.
**d.** Only **iv**.
**e.** All of these.

**8.** In a B-tree of degree 8 and size 1000, which of these could not be the height:

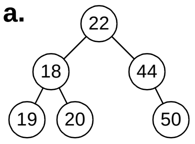
**i.** 2
**ii.** 3
**iii.** 4
**iv.** 5

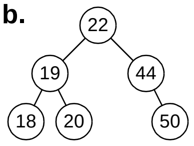
**a.** Only **i**.
**b.** Only **i** and **ii**.
**c.** Only **i** and **iv**.
**d.** Only **iv**.
**e.** All of these.

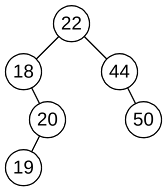
Questions 9-10 refer to this binary search tree:

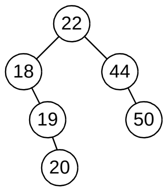


**9.** How will the tree look after the key 19 is inserted?

**a.**


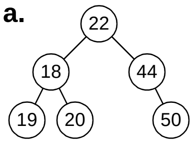
**b.**


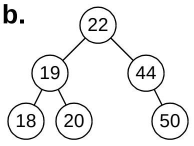
**c.**


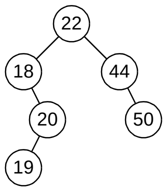
**d.**


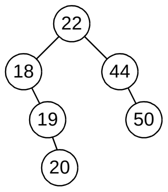
**e.** None of these.

**10.** If it is an AVL tree, then how will the tree look after the key 19 is inserted?

**a.**


**b.**


**c.**


**d.**


**e.** None of these.

## Answers to True/False Questions

1. True
2. False.
3. True.
4. True.
5. True.
6. True.
7. True.
8. False.
9. False
10. True
11. False.
12. True.
13. True.
14. True.
15. False.
16. True.
17. False.
18. True.
19. False.
20. True.

## Answers to Multiple Choice Questions

1. d
2. d
3. a
4. d

| | |
|---|---|
| 5. | a |
| 6. | e |
| 7. | b |
| 8. | c |
| 9. | c |
| 10. | b |

# Chapter 14
# Heaps and Priority Queues

**Answer “True” or “False”:**

**1.** In a heap, if $x$ is the parent of $y$, then $x \geq y$.

**2.** In a heap, if $x$ is the right sibling of $y$, then $x \geq y$.

**3.** If the sequence $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ is a heap, then $a_3 \geq a_7$.

**4.** The Heapify algorithm moves down a root-to-leaf path, promoting the larger child at each level.

**5.** The Heapify algorithm runs in logarithmic time.

**6.** The Build Heap algorithm runs in logarithmic time.

**7.** With the heap implementation of the `PriorityQueue` interface, the `best()` method runs in logarithmic time.

**8.** With the heap implementation of the `PriorityQueue` interface, the `add()` method runs in logarithmic time.

**9.** With the heap implementation of the `PriorityQueue` interface, the `remove()` method runs in logarithmic time.

**10.** The Huffman code for a document assigns a unique binary code to each character used in the document.

**Select the best answer:**

**1.** Which of these is not a root-to leaf subsequence of $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$?

- **i.** $(a_0, a_1, a_3, a_7)$
- **ii.** $(a_0, a_1, a_3, a_8)$
- **iii.** $(a_0, a_1, a_4, a_8)$

**a.** Only **i**.
**b.** Only **ii**.
**c.** Only **iii**.
**d.** Only **i** and **ii**.
**e.** Only **ii** and **iii**.

**2.** The root-to-leaf index to $a_{50}$ is:

**a.** $(a_0, a_2, a_5, a_{11}, a_{24}, a_{50})$
**b.** $(a_0, a_1, a_2, a_5, a_{11}, a_{24}, a_{50})$
**c.** $(a_0, a_1, a_3, a_6, a_{12}, a_{24}, a_{50})$
**d.** $(a_0, a_1, a_3, a_6, a_{12}, a_{25}, a_{50})$
**e.** $(a_0, a_1, a_2, a_5, a_{11}, a_{24}, a_{50})$
**f.** $(a_0, a_2, a_5, a_{11}, a_{25}, a_{50})$

**3.** If the sequence $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ is a heap, then which element is the parent of $a_4$?

**a.** $a_0$
**b.** $a_1$
**c.** $a_2$
**d.** $a_7$
**e.** $a_8$

**4.** If the sequence $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ is a heap, then which element is the right child of $a_3$?

**a.** $a_0$
**b.** $a_1$
**c.** $a_6$
**d.** $a_7$
**e.** $a_8$

**5.** Which of these arrays does not have the heap property?

**a.**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 99 | 88 | 66 | 44 | 77 | 55 | 22 | 33 |

**b.**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 99 | 88 | 55 | 77 | 22 | 33 | 44 | 66 |

**c.**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 99 | 88 | 44 | 55 | 77 | 22 | 33 | 66 |

**d.**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 99 | 88 | 44 | 77 | 55 | 22 | 33 | 66 |

**e.**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 99 | 88 | 44 | 77 | 55 | 33 | 22 | 66 |

## Answers to True/False Questions

1. True.
2. False.
3. True.
4. True.
5. True.
6. False.
7. False.
8. True.
9. True.
10. True.

## Answers to Multiple Choice Questions

1. c
2. a
3. b
4. e
5. c

# Chapter 15
# Sorting

**Answer “True” or “False”:**

**1.** No comparison sort can do better than $\Theta(n \lg n)$.

**2.** On a sequence that is nearly sorted, the Insertion Sort is faster than the Merge Sort.

**3.** On a sequence that is nearly sorted, the Merge Sort is faster than the Quick Sort.

**4.** On a sequence that is nearly sorted in reverse order, the Insertion Sort runs faster than the Merge Sort.

**5.** On a sequence that is nearly sorted in reverse order, the Heap Sort runs faster than the Merge Sort.

**6.** If the length of a sequence is doubled, the Selection Sort will take about twice as long to run.

**7.** The Selection Sort will take the same time to run on a sequence of length *n*, regardless of the original order of the elements.

**8.** The Insertion Sort will take the same time to run on a sequence of length *n*, regardless of the original order of the elements.

**9.** The Quick Sort will take the same time to run on a sequence of length *n*, regardless of the original order of the elements.

**10.** The Heap Sort will take the same time to run on a sequence of length *n*, regardless of the original order of the elements.

**Select the best answer:**

**1.** Which sorting algorithm runs in linear time in the best case?

**a.** Selection Sort
**b.** Insertion Sort
**c.** Merge Sort
**d.** Quick Sort
**e.** Heap Sort

**2.** Which sorting algorithm runs faster on a sequence that is in reverse order than on a random sequence?

**a.** Selection Sort
**b.** Insertion Sort
**c.** Merge Sort
**d.** Quick Sort
**e.** Heap Sort

**3.** Which sorting algorithm makes the fewest number of comparisons on a sequence whose elements are all equal?

**a.** Selection Sort
**b.** Insertion Sort
**c.** Merge Sort
**d.** Quick Sort
**e.** Heap Sort

**4.** Which sorting algorithm uses the most amount of space to sort a sequence on *n* elements?

**a.** Selection Sort
**b.** Insertion Sort
**c.** Merge Sort
**d.** Quick Sort
**e.** Heap Sort

**5.** Which sorting algorithm is not stable?

**a.** Quick Sort
**b.** Selection Sort
**c.** Insertion Sort
**d.** Merge Sort
**e.** Quick Sort

**6.** How many comparisons would the Selection Sort make on an array of 8 elements that is already in ascending order?

**a.** 7
**b.** 8
**c.** 24
**d.** 28
**e.** 64

**7.** How many comparisons would the Insertion Sort make on an array of 10 elements that is already in ascending order?

**a.** 7
**b.** 8
**c.** 24
**d.** 28
**e.** 64

**8.** How many comparisons would the Insertion Sort make on an array of 10 elements that is already in descending order?

**a.** 7
**b.** 8
**c.** 24
**d.** 28
**e.** 64

**9.** How many comparisons would the Quick Sort make on an array of 10 elements that is already in ascending order?

**a.** 7
**b.** 8
**c.** 24
**d.** 28
**e.** 64

**10.** The Radix Sort applies the Counting Sort on each digit (step 2 of Algorithm 15.10 on page 467). Which of these alternatives to the Counting Sort would cause the Radix Sort to fail?

**a.** Bubble Sort
**b.** Insertion Sort
**c.** Selection Sort
**d.** Shell Sort
**e.** Quick Sort

## Answers to True/False Questions

1. True
2. True
3. True
4. False
5. True
6. False
7. True.
8. False.
9. False.
10. False.

## Answers to Multiple Choice Questions

1. b
2. e
3. b
4. c
5. e
6. d
7. a
8. d
9. d
10. e

# Chapter 16
# Graphs

## Answer “True” or “False”:

**1.** A graph is connected if every vertex has a path to every other vertex.

**2.** A cycle is a path that ends at the same vertex where it begins.

**3.** The implementation of the Breadth-First Search algorithm uses a queue.

**4.** The implementation of the Depth-First Search algorithm uses a queue.

**5.** A spanning tree for a graph includes all the edges of the graph.

**6.** A spanning tree for a graph includes all the vertices of the graph.

**7.** Both the Breadth-First Search and the Depth-First Search generate a spanning tree for their graph.

**8.** The complete graph with $n$ vertices has $n^2$ edges..

**9.** The adjacency matrix for a weighted graph has two entries for each edge.

**10.** The adjacency list for a weighted graph has two entries for each edge.

## Select the best answer:

**1.** How many 1s are there in the adjacency matrix of a graph with 6 edges?

**a.** 0
**b.** 1
**c.** 2
**d.** 6
**e.** 12

**2.** How many 1s are there in the adjacency matrix of the complete graph $K_n$?

**a.** 0
**b.** $n$
**c.** $2n$
**d.** $(n^2 - n)/2$
**e.** $n^2 - n$

**3.** What is the output from this program, using the `Graph` class from Listing 16.1 on page 483:

```
class TestGraph {
  public static void main(String[] args) {
    Graph g = new Graph(new String[]{"A", "B", "C", "D", "E"});
    g.add("A", "B");
    g.add("B", "C");
    g.add("C", "D");
    g.add("D", "A");
    System.out.println(g);
  }
}
```

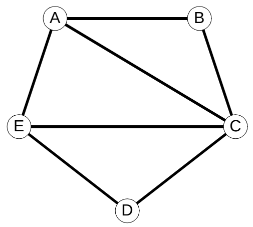
**a.** `{A:BC,B:AC,C:BD,D:AC}`
**b.** `{A:BC,B:ACD,C:BD,D:ABC}`
**c.** `{A:BCD,B:AC,C:ABD,D:AC}`
**d.** `{A:BCD,B:ACD,C:ABD,D:ABC}`
**e.** None of these.

**4.** What is the output from this program, using the `Graph` class from Listing 16.1 on page 483:

```
class TestGraph {
  public static void main(String[] args) {
    Graph g = new Graph(new String[]{"A", "B", "C", "D", "E"});
    g.add("A", "B");
    g.add("B", "C");
    g.add("B", "D");
    g.add("C", "D");
    g.add("D", "E");
    g.add("E", "A");
    System.out.println(g);
  }
}
```

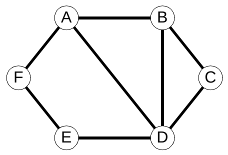
**a.** `{A:B,B:CD,C:D,D:E,E:A}`
**b.** `{A:B,B:ACD,C:D,D:BCE,E:A}`
**c.** `{A:BE,B:ACD,C:BD,D:BCE,E:AD}`
**d.** `{A:BCE,B:ACD,C:ABD,D:BCE,E:ABD}`
**e.** None of these.

**5.** What is the Breadth First Search of this graph?
- **a.** ABCDE
- **b.** ABCED
- **c.** ABEDC
- **d.** ABECD
- **e.** None of these.



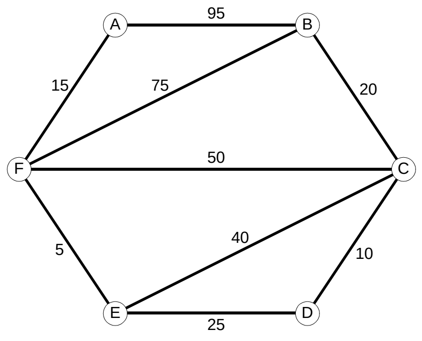
**6.** What is the Depth First Search of the graph in Question 5?
- **a.** ABCDE
- **b.** ABCED
- **c.** ABEDC
- **d.** ABECD
- **e.** None of these.

**7.** What is the Breadth First Search of thIs graph?
- **a.** ABCDEF
- **b.** ABDCEF
- **c.** ABDFEC
- **d.** ABDFCE
- **e.** None of these.



**8.** What is the Depth First Search of the graph in Question 7?
- **a.** ABCDEF
- **b.** ABDCEF
- **c.** ABDFEC
- **d.** ABDFCE
- **e.** None of these.

**9.** When Dijkstra’s algorithm (Algorithm 16.5 on page 495) is applied to this graph, what values does *v.dist* have for vertex B?
- **a.** 95, 90, 85, 80, 70
- **b.** 95, 90, 85, 80, 75
- **c.** 95, 90, 85, 80
- **d.** 95, 90, 85, 75
- **e.** None of these.



**10.** When Dijkstra's algorithm (Algorithm 16.5 on page 495) is applied to the graph in Question 9, what values does *v.prev* have for vertex C?

- **a.** BFE
- **b.** BFED
- **c.** FED
- **d.** FE
- **e.** None of these.

## Answers to True/False Questions

**1.** True.

**2.** True.

**3.** True.

**4.** False.

**5.** False.

**6.** True.

**7.** True.

**8.** False.

**9.** True.

**10.** True.

## Answers to Multiple Choice Questions

**1.** **e**

**2.** **e**

**3.** **d**

**4.** **c**

**5.** **e**

**6.** **a**

**7.** **d**

**8.** **a**

**9.** **b**

**10.** **c**