Chapter 20 Questions

1. What is a data structure?

1. A data structure is a collection of data organized in some fashion.
2. In object-oriented thinking, a data structure is an object that stores other objects, referred to as data or elements.
3. some people refer a data structure as a container object or a collection object
4. **All of the above.**

2. Describe the Java Collections Framework under the Collection interface.

A. **The Java Collections Framework defines the Java API for handling common data structures tasks in Java**

B. The Java Collections Framework defines the Java API for handling common Algorithm tasks in Java.

C. Both A and B

D. None of the above

**2. Describe the Java List the interfaces under the Collection interface.**

A. **It defines classes and interfaces for storing and manipulating data in sets, lists, and maps.**

B. A convenience class is an abstract class that partially implements an interface. The Java

C. Collections Framework defines interfaces, convenience abstract classes, and concrete classes.

D. The Java Collections Framework defines the Java API for handling common Algorithm tasks in Java.

**2. Describe the Java convenience abstract classes under the Collection interface.**

A. It defines classes and interfaces for storing and manipulating data in sets, lists,

and maps.

**B. Collections Framework defines interfaces, convenience abstract classes, and concrete classes.**

C. Collections Framework defines interfaces, convenience abstract classes, and concrete classes.

D. The Java Collections Framework defines the Java API for handling common Algorithm tasks in Java.

Q.3. Can a collection object be cloned and serialized?

1. **Yes. The concrete classes of Set, List, and Map implements the clone() method in the Cloneable interface.**
2. No.

Q.4. What method do you use to add all the elements from one collection to another collection?

1. **addAll(Collection c).**
2. add(Collection c).
3. putAll(Collection c).
4. setAll(Collection c).

Q.7. What method do you use to obtain an element in the collection from an iterator?

1. **Use the next() method.**
2. Use the hasNext() method.
3. Use the nextElement() method.
4. Use the get() method.

Q.8. Can you use a foreach loop to traverse the elements in any instance of Collection?

A. No

**B. Yes**

Q.9. When using a foreach loop to traverse all elements in a collection, do you need to use the next() or hasNext() methods in an iterator?

1. **No. They are implicitly used in a foreach loop.**
2. Yes

Q.10. Can you use the forEach method on any instance of Collection? Where is the forEach method defined?

A. No

B. **Yes. It is defined in the Iterable interface which is a super interface for Collection.**

How do you invoke the sort method to sort a list of objects of the type B1?

list.sort(new B());

To sort an array x of objects of the type B1, use

java.util.Arrays.sort(x, new B());

Q.18. Write a lambda expression to create a comparator that compares two Loan objects by their annualInterestRate. Create a comparator using the Comparator.comparing method to compare Loan objects on annualInterestRate. Create a comparator to compare Loan objects first on annualInterestRate then on loanAmount.

(e1, e2) -> e1.getAnnualInterestRate() < e2.getAnnualInterestRate() ? -1 :

e1.getAnnualInterestRate() == e2.getAnnualInterestRate() ? 0 : 1

Comparator.comparing(Loan::getAnnualInterestRate);

Comparator.comparing(Loan::getAnnualInterestRate)

.thenComparing(Loan:getLoanAmount);

Q.19. Create a comparator using a lambda expression and using the Comparator.comparing method, respectively, to compare Collection objects on their size.

(e1, e2) -> e1.size() - e2.size()

Comparator.comparing(Collection::size)

Q.20. Write a statement that sorts an array named points of Point2D objects on their y values and then on their x values.

java.util.sort(points,

Comparator.comparing(Point2D::x).thenComparing(Point2D::y));

Q.21. Write a statement that sorts an ArrayList of strings named list in increasing order of their last character.

list.sort((e1, e2) -> {

if (e1.length() == 0)

return -1;

else (e2.length() == 0)

return 1;

else

return charAt(e1.size() - 1) - charAt(e2.size() - 1);

}

Q.22. Write a statement that sorts a two-dimensional array of double[][] in increasing order of their second column. For example, if the array is double[][] x = {{3, 1}, {2, -1}, {2, 0}}, the sorted array will be {{2, -1}, {2, 0}, {3, 1}}.

java.util.Arrays.sort(x, (e1, e2) -> (int)(e1[1] - e2[1]));

Q.23. Write a statement that sorts a two-dimensional array of double[][] in increasing order of their second column as the primary order and the first column as the secondary order. For

example, if the array is double[][] x = {{3, 1}, {2, -1}, {2, 0}, {1, -1}}, the sorted array will

be {{1, -1}, {2, -1}, {2, 0}, {3, 1}}.

java.util.Arrays.sort(x, (e1, e2) -> {

if (e1[1] - e2[1] != 0)

return (int)(e1[1] - e2[1]);

else

return (int)(e1[0] - e2[0]);

});

Q.24. Are all the methods in the Collections class static?

Yes.

Q.25. Which of the following static methods in the Collections class are for lists, and which are for collections?

sort, binarySearch, reverse, shuffle, max, min, disjoint, frequency

The methods for lists are: sort, binarySearch, reverse, shuffle

The methods for collections are: max, min, disjoint, frequency

Note that all the methods for collections are also for lists, because lists are collections.

Q.26. Show the output of the following code:

import java.util.\*;

public class Test {

public static void main(String[] args) {

List<String> list =

Arrays.asList("yellow", "red", "green", "blue");

Collections.reverse(list);

System.out.println(list);

List<String> list1 =

Arrays.asList("yellow", "red", "green", "blue");

List<String> list2 = Arrays.asList("white", "black");

Collections.copy(list1, list2);

System.out.println(list1);

Collection<String> c1 = Arrays.asList("red", "cyan");

Collection<String> c2 = Arrays.asList("red", "blue");

Collection<String> c3 = Arrays.asList("pink", "tan");

System.out.println(Collections.disjoint(c1, c2));

System.out.println(Collections.disjoint(c1, c3));

Collection<String> collection =

Arrays.asList("red", "cyan", "red");

System.out.println(Collections.frequency(collection, "red"));

}

}

[blue, green, red, yellow]

[white, black, green, blue]

false

true

2

Q.27. Which method can you use to sort the elements in an ArrayList or a LinkedList? Which method can you use to sort an array of strings?

You can use Collections.sort(list) to sort an ArrayList or a LinkedList and use Arrays.sort

(Object[]) to sort an array of strings. For example,

LinkedList<String> list = new LinkedList<>();

list.add("Java"); list.add("Python"); list.add("C++");

java.util.Collections.sort(list); // Sort the list

String[] languages = {"Java", "Python", "C++"};

java.util.Arrays.sort(languages); // Sort the array

Q.28. Which method can you use to perform binary search for elements in an ArrayList or a LinkedList? Which method can you use to perform binary search for an array of strings?

You can use Collections.binary(list, key) to perform binary search for an ArrayList or a

LinkedList and use Arrays.binary(Object[], key) to sort an array of strings.

Q.29. Write a statement to find the largest element in an array of comparable objects.

Collections.max(Arrays.asList(arrayObject))

Q.30. What is the return value from invoking pane.getChildren() for a pane?

The return value is an ObservableList<Node>, which is a subtype of List<Node>.

Q.31. How do you modify the code in the MutilpleBallApp program to remove the first ball in the list when the button is clicked?

Replace line 75 with the following code:

getChildren().remove(getChildren().size() - 1);

Q.32. How do you modify the code in the MutilpleBallApp program so that each ball will get a random radius between 10 and 20?

Change line 133 to

radius = Math.random\*11 + 10;

Q.33. How do you create an instance of Vector? How do you add or insert a new element into a vector? How do you remove an element from a vector? How do you find the size of a vector?

Vector is the same as ArrayList except that, except that Vector contains the synchronized

methods for accessing and modifying the vector. Since Vector implements List, you can use

the methods in List to add, remove elements from a vector, and use the size() method to find

the size of a vector. To create a vector, use either its constructors.

Q.34. How do you create an instance of Stack? How do you add a new element to a stack? How do you remove an element from a stack? How do you find the size of a stack?

Stack is a subclass of Vector. The Stack class represents a last-in-first-out stack of objects.

The elements are accessed only from the top of the stack. You can retrieve, insert, or remove

an element from the top of the stack. To add a new element to a stack, use the push method.

To remove an element from the top of the stack, use the method pop. To find a stack size,

use the size() method.

Q.35. Does Listing 20.1, TestCollection.java, compile and run if all the occurrences of ArrayList are replaced by LinkedList, Vector, or Stack?

Yes, because these classes are subtypes of the Collection interface.

Q.36. Is java.util.Queue a subinterface of java.util.Collection, java.util.Set, or java.util.List? Does LinkedList implement Queue?

java.util.Queue is a subinterface of java.util.Collection, and LinkedList implements Queue.

Q.37. How do you create a priority queue for integers? By default, how are elements ordered in a priority queue? Is the element with the least value assigned the highest priority in a priority queue?

Use the constructors of PriorityQueue to create priority queues. By default, the elements in a

priority queue are ordered in their natural order using the compareTo method in the

Comparable interface. The element with the least value is assigned the highest priority in

PriorityQueue.

Q.38. How do you create a priority queue that reverses the natural order of the elements?

new PriorityQueue(initialCapacity, Collections.reverseOrder()).

Q.39. Can the EvaluateExpression program evaluate the following expressions "1+2", "1 + 2", "(1) + 2", "((1)) + 2", and "(1 + 2)"?

Yes.

Q.40. Show the change of the contents in the stacks when evaluating "3 + (4 + 5) \* (3 + 5) + 4 \* 5" using the EvaluateExpression program.

Omitted.

Q.41. If you enter an expression "4 + 5 5 5", the program will display 10. How do you fix this problem?

You can fix this problem by throwing an exception if operandStack is not empty after poping the result out of the operandStack stack.

Written Questions:

Q.5. When should a method throw an UnsupportedOperationException?

If a method has no meaning in the subclass, you can implement it in the subclass to throw java.lang.UnsupportedOperationException, a subclass of RuntimeException. This is a good design that you can use in your project. If a method has no meaning in the subclass, you can

implement it as follows:

public void someMethod() {

throw new UnsupportedOperationException

("Method not supported");

}

Q.6. How do you obtain an iterator from a collection object?

The Collection interface extends the Iterable interface. You can obtain an iterator from a

collection using the iterator() method.

Q.11. How do you add and remove elements from a list? How do you traverse a list in both directions?

Use the add or remove method to add or remove elements from a list. Use the listIterator() to obtain an iterator. This iterator allows you to traverse the list bi-directional.

Q.13. What are the differences between ArrayList and LinkedList? Which list should you use to insert and delete elements at the beginning of a list?

ArrayList and LinkedList can be operated similarly. The critical differences between them

are their internal implementation, which impacts the performance. ArrayList is efficient for

retrieving elements, and for adding and removing elements from the end of the list.

LinkedList is efficient for adding and removing elements anywhere in the list.

Q.14. Are all the methods in ArrayList also in LinkedList? What methods are in LinkedList but not in ArrayList?

All the methods in ArrayList are also in LinkedList except the trimToSize() method. The

methods getFirst, getLast, addFirst, addLast are in LinkedList, but not in ArrayList.

Q.15. How do you create a list from an array of objects?

A simple way to create a list from an array of objects is to use

new ArrayList(Arrays.asList(arrayObject))

or

new LinkedList(Arrays.asList(arrayObject)).

Q.16. What are the differences between the Comparable interface and the Comparator interface? In which package is Comparable, and in which package is Comparator?

The Comparable interface contains the compareTo method and Comparator interface

contains the compare method and equals method. Normally, if the objects of a class have

natural order (e.g., String, Date), let the class implement the Comparable interface. The

Comparator interface is more flexible in the sense that it enables you to define a new class

that contains the compare(Object, Object) method to compare two objects of other classes.

The Comparable interface is in the java.lang package, and the Comparator interface is in the

java.util package.

Q.17. How do you define a class A that implements the Comparable interface? Are two instances of class A comparable? How do you define a class B that implements the Comparator interface and override the compare method to compare two objects of type B1? How do you invoke the sort method to sort a list of objects of the type B1 using a comparator?

How do you define a class A that implements the Comparable interface?

public class A implements Comparable<A> {

public int compareTo(A o) {

return an integer;

}

}

Are two instances of class A comparable? Yes.How do you define a class B that implements

the Comparator interface and override the compare method to compare two objects of type

B1?

public class B implements Comparator<B1> {

public int compare(B1 o1, B1 o2) {

return an integer;

}

}

Chapter 21 Questions

Q.1. How do you create an instance of Set? How do you insert a new element in a set? How do

you remove an element from a set? How do you find the size of a set?

The Set is an interface. To create an instance of Set, you need to use HashSet or TreeSet. To

insert an element to a set, use the add method. To remove an element from a set, use the

remove method. To find the size of a set, use the size() method.

Q.2. If two objects o1 and o2 are equal, what is o1.equals(o2) and o1.hashCode() == o2.hashCode

()?

If two objects o1 and o2 are equal, o1.equals(o2) is true and o1.hashCode() == o1.hashCode

() is true.

Q.3. What are the differences between HashSet, LinkedHashSet, and TreeSet?

HashSet is unsorted, but TreeSet is sorted. HashSet is more efficient than TreeSet if you

don't want the elements in a set to be sorted. If you create a TreeSet using its default

constructor, the compareTo method is used to compare the elements in the set, assuming that

the class of the elements implements the Comparable interface. To use a comparator, you

have to use the constructor TreeSet(Comparator comparator) to create a sorted set that uses

the compare method in the comparator to order the elements in the set. A runtime error

would occur if you add an element that cannot be compared with the existing elements in the

tree set?

Q.4. How do you traverse the elements in a set?

To traverse a set, use the iterator method to obtain an iterator. You can then traverse the set

through the iterator. Using an iterator, you can traverse only sequentially from the beginning

to the end.

Q.5. How do you sort the elements in a set using the compareTo method in the Comparable

interface? How do you sort the elements in a set using the Comparator interface? What

would happen if you added an element that could not be compared with the existing elements

in a tree set? How does the TreeeSet test if two elements are equal?

To sort the elements in a set using the Comparable interface, there are two ways: (1) create a

TreeSet using new TreeSet(), (2) create a TreeSet from a set using new TreeSet(set). To sort

the elements in a set using the Comparator interface, create a TreeSet using new TreeSet

(Comparator), then add the elements to the tree set.

How does the TreeeSet test if two elements are equal? two elements are considered equal in

a TreeSet if e1.compare(e2) is 0 using the comparator. If two elements are equal, only one

can be stored in a TreeSet. Note that two elements e1 and e2 in a HashSet and

LinkedHashSet are considered equal, if e1.equals(e2) is true and their hashCode() are the

same.

Q.6. Suppose that set1 is a set that contains the strings red, yellow, and green, and that set2 is

another set that contains the strings red, yellow, and blue. Answer the following questions:

(a) What are in set1 and set2 after executing set1.addAll(set2)?

(b) What are in set1 and set2 after executing set1.add(set2)?

(c) What are in set1 and set2 after executing set1.removeAll(set2)?

(d) What are in set1 and set2 after executing set1.remove(set2)?

(e) What are in set1 and set2 after executing set1.retainAll(set2)?

(f) What is in set1 after executing set1.clear()?

Ans:

Q.7. Show the output of the following code:

import java.util.\*;

public class Test {

public static void main(String[] args) {

LinkedHashSet<String> set1 = new LinkedHashSet<>();

set1.add("New York");

LinkedHashSet<String> set2 = set1;

LinkedHashSet<String> set3 =

(LinkedHashSet<String>)(set1.clone());

set1.add("Atlanta");

System.out.println("set1 is " + set1);

System.out.println("set2 is " + set2);

System.out.println("set3 is " + set3);

set1.forEach(e -> System.out.print(e + " "));

}

}

Ans:

Q.8. Show the output of the following code:

Set<String> set = new LinkedHashSet<>();

set.add("ABC");

set.add("ABD");

System.out.println(set);

[ABC, ABD]

Q.9. What will the output be if lines 6-7 in Listing 21.5 is replaced by the following code:

Set<GeometricObject> set = new HashSet<>();

It will display the area of four geometric objects in random order. Note that the eqauls

method is not overidden in the Circle class. Therefore, both new Circle(40) in lines 9-10 are

stored in the hash set.

Q.10. Show the output of the following code:

Set<String> set = new TreeSet<>(

Comparator.comparing(String::length));

set.add("ABC");

set.add("ABD");

System.out.println(set);

[ABC]

Q.11. (This is not in the printed book) Show the output of the following code:

import java.util.\*;

import java.io.\*;

public class Test {

public static void main(String[] args) throws Exception {

ObjectOutputStream output = new ObjectOutputStream(

new FileOutputStream("c:\\test.dat"));

LinkedHashSet<String> set1 = new LinkedHashSet<>();

set1.add("New York");

LinkedHashSet<String> set2 =

(LinkedHashSet<String>)set1.clone();

set1.add("Atlanta");

output.writeObject(set1);

output.writeObject(set2);

output.close();

ObjectInputStream input = new ObjectInputStream(

new FileInputStream("c:\\test.dat"));

set1 = (LinkedHashSet<String>)input.readObject();

set2 = (LinkedHashSet<String>)input.readObject();

System.out.println(set1);

System.out.println(set2);

input.close();

}

}

[New York, Atlanta]

[New York]

Q.12. Suppose you need to write a program that stores unordered, non-duplicate elements, what

data structure should you use?

Suppose you need to write a program that stores non-duplicate elements, you should use a

HashSet.

Q.13. Suppose you need to write a program that stores non-duplicate elements in the order of

insertion, what data structure should you use?

Use a LinkedHashSet.

Q.14. Suppose you need to write a program that stores non-duplicate elements in increasing order

of the element values, what data structure should you use?

Use a TreeSet.

Q.15. Suppose you need to write a program that stores a fixed number of the elements (possibly

duplicates), what data structure should you use?

Use an array.

Q.16. Suppose you need to write a program that stores the elements in a list with frequent

operations to append and delete elements at the end of the list, what data structure should

you use?

Use an ArrayList.

Q.17. Suppose you need to write a program that stores the elements in a list with frequent

operations to insert and delete elements at the beginning of the list, what data structure

should you use?

Use a LinkedList.

Q.18. Will the CountKeywords program work if lines 33-34 are changed to

Set<String> keywordSet =

new LinkedHashSet<>(Arrays.asList(keywordString));

Yes.

Q.19. Will the CountKeywords program work if lines 33-34 are changed to

List<String> keywordSet =

new ArrayList<String>(Arrays.asList(keywordString));

Yes.

Q.20. How do you create an instance of Map? How do you add an entry to a map consisting of a

key and a value? How do you remove an entry from a map? How do you find the size of a

map? How do you traverse entries in a map?

Map is an interface. To create an instance of Map, you need to use the HashMap class or the

TreeMap class. The HashMap and TreeMap have various constructors that you can use to

create a HashMap or a TreeMap. You can use the put method to add an entry to a map, and

the remove method to remove an entry with the specified key from the map. Use the size

method to find the size of a map.

Q.21. Describe and compare HashMap, LinkedHashMap, and TreeMap.

The HashMap, LinkedHashMap, and TreeMap classes are three concrete implementations of

the Map interface. The HashMap class is efficient for locating a value, inserting a mapping,

and deleting a mapping. The entries in a HashMap are not ordered, but the entries in a

LinkedHashMap can be retrieved in the order in which they were inserted into the map

(known as the insertion order), or the order in which they were last accessed, from least

recently accessed to most recently (access order). The TreeMap class, implementing

SortedMap, is efficient for traversing the keys in a sorted order.

Q.22. Show the printout of the following code:

import java.util.\*;

public class Test {

public static void main(String[] args) {

Map<Integer, String> map = new LinkedHashMap<>();

map.put("123", "John Smith");

map.put("111", "George Smith");

map.put("123", "Steve Yao");

map.put("222", "Steve Yao");

System.out.println("(1) " + map);

System.out.println("(2) " + new TreeMap<, String>(map));

map.forEach((k, v) -> {

if (k.equals("123")) System.out.println(v);});

}

}

(1) {123=Steve Yao, 111=George Smith, 222=Steve Yao}

(2) {111=George Smith, 123=Steve Yao, 222=Steve Yao}

Q.23. Will the CountOccurrenceOfWords program work if line 10 is changed to

Map<String, int> map = new TreeMap<>();

No. A concrete type must of an object type.

Q.24. Will the CountOccurrenceOfWords program work if line 17 is changed to

if (map.get(key) == null) {

Yes. It will work.

Q.25. Will the CountOccurrenceOfWords program work if lines 32-33 are changed to

for (String key: map)

System.out.println(key + "\t" + map.getValue(key));

No. You cannot iterate on keys in a map in Java.

Q.26. Replace the code in lines 17-24 in one line of code using a conditional expression.

map.put(key, map.containsKey(key) ? map.get(key) + 1 : 1);

Q.27. What is wrong in the following code?

Set<String> set = Collections.singleton("Chicago");

set.add("Dallas");

The singleton set is immutable. You cannot add an element into the singleton set.

Q.28. What happens when you run the following code?

List<String> list = Collections.unmodifiableList(

Arrays.asList("Chicago", "Boston"));

list.remove("Dallas");

The list created from the unmodifiableList method is immutable is an unmodifiable view of

the list. You cannot modify the list through this view.

Chapter 22 Questions

Q.1. Why is a constant factor ignored in the Big O notation? Why is a nondominating term

ignored in the Big O notation?

The constant factor is ignored in big O notation, because it has no impact on the growth rate

of the time complexity function. A nondominating term is ignored in Big O notation,

because as the input size grows, the dominating term grows much faster than the

nondominating term.

Q.2. What is the order of each of the following functions?

(a) (n2 + 1)2/n

(b) (n2 + log2n)2 / n

(c) n3 + 100n2 + n

(d) 2n + 100n2 + 45n

(e) n2n + n22n

(a) (n2 + 1)2/n = O(n3)

(b) (n2 + log2n)2 / n = O(n3)

(c) n3 + 100n2 + n = O(n3)

(d) 2n + 100n2 + 45n = O(2n)

(e) n2n + n22n = O(n22n)

Q.3. Count the number of iterations in the following loops.

(a)

int count = 1;

while (count < 30) {

count = count \* 2;

}

(b)

int count = 15;

while (count < 30) {

count = count \* 3;

}

(c)

int count = 1;

while (count < n) {

count = count \* 2;

}

(d)

int count = 15;

while (count < n) {

count = count \* 3;

}

(A) 5

(B) 1

(C) The ceiling of log2n times

(D) The ceiling of log3(n/15) times

Q.4. How many stars are displayed in the following code if n is 10? How many if n is 20? Use the

Big O notation to estimate the time complexity.

(a)

for (int i = 0; i < n; i++) {

System.out.print('\*');

}

(b)

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

System.out.print('\*');

}

}

(c)

for (int k = 0; k < n; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

System.out.print('\*');

}

}

}

(d)

for (int k = 0; k < 10; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

System.out.print('\*');

}

}

}

if n is 10: (a) 10 (b) 10^2 (c) 10^3 (d) 10\*10^2

if n is 20: (a) 20 (b) 20^2 (c) 20^3 (d) 20\*20^2

Using Big-O notation: O(n), O(n^2), O(n^3), O(n^2)

Q.5. Use the Big O notation to estimate the time complexity of the following methods:

(a)

public static void mA(int n) {

for (int i = 0; i < n; i++) {

System.out.print(Math.random());

}

}

(b)

public static void mB(int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < i; j++)

System.out.print(Math.random());

}

}

(c)

public static void mC(int[] m) {

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

System.out.print(m[i]);

}

for (int i = m.length - 1; i >= 0; )

{

System.out.print(m[i]);

i--;

}

}

(d)

public static void mD(int[] m) {

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

for (int j = 0; j < i; j++)

System.out.print(m[i] \* m[j]);

}

}

(a): O(n)

(b): O(n^2)

(c): O(n)

(d): O(n^2)

Q.6. Design an O(n) time algorithm for computing the sum of numbers from n1 to n2 for (n1 <

n2). Can you design an O(1) for performing the same task?

An O(n) time algorithm for this is

int sum = 0;

for (int i = n1; i <= n2; i++)

sum += i;

An O(1) time algorithm for this is

int sum = n2 \* (n2 + 1) / 2 - n1 \* (n1 - 1) / 2;

Q.7. Example 7 in Section 22.3 assumes n = 2k Revise the algorithm for an arbitrary n and prove

that the complexity is still O(logn).

result = a;

i = 2;

while (i <= n) {

result = result \* result;

i \*= 2;

}

for (int j = i / 2 + 1; j <= n; j++)

result = result \* a;

Assume that 2k-1 <= n < 2k. The while loop is executed k-1 times. The for loop is executed at

most 2k-2k-1=2k-1 times. So, the total complexity is O(n). Consider another implementation:

public static int f(int a, int n) {

if (n == 1) {

return a;

}

else {

int temp = f(a, n / 2);

if (n % 2 == 0) {

return temp \* temp;

}

else {

return a \* temp \* temp;

}

}

}

This implementation results in O(logn) complexity.

Q.8. Put the following growth functions in order: 5n3/4032, 44logn, 10nlogn, 500, 2n2, 2n/45, 3n

500,44logn,3n,10nlogn 2n2,5n3/4032,2n/45

Q.9. Estimate the time complexity for adding two n by m matrices, and for multiplying an n by m

matrix by an m by k matrix.

Adding two matrices: O(nm). Multiplying two matrices: O(nmk)

Q.10. Describe an algorithm for finding the occurrence of the max element in an array. Analyze the

complexity of the algorithm.

The algorithm can be designed as follows: Maintain two variables, max and count. max

stores the current max number, and count stores its occurrences. Initially, assign the first

number to max and 1 to count. Compare each subsequent number with max. If the number is

greater than max, assign it to max and reset count to 1. If the number is equal to max,

increment count by 1. Since each element in the array is examined only once, the complexity

of the algorithm is O(n).

Q.11. Describe an algorithm for removing duplicates from an array. Analyze the complexity of the

algorithm.

The algorithm can be designed as follows: For each element in the input array, store it to a

new array if it is new. If the number is already in the array, ignore it. The time for checking

whether an element is already in the new array is O(n), so the complexity of the algorithm is

O(n^2).

Q.12. Analyze the following sorting algorithm:

for (int i = 0; i < list.length - 1; i++) {

if (list[i] > list[i + 1]) {

swap list[i] with list[i + 1];

i = -1;

}

}

This is similar to bubble sort. Whenever a swap is made, it goes back to the beginning of the

loop. In the worst case, there will be O(n^2) of swaps. For each swap, O(n) number of

comparisons may be made in the worst case. So, the total is O(n^3) in the worst case.

Q.13. Analyze the complexity for computing a polynomial f(x) of degree n for a given x value

using a brute-force approach and the Horner's approach, respectively. A brute-force approach

is to compute each term in the polynomial and add them together. The Horner's approach

was introduced in Section 6.7.

f(x) = anxn + an-1xn-1 + an-2xn-2 + ... + a1x1 + a0

A brute-force for approach to evaluate a polynomial f(x) of degree n will take n+(n-1)+…

+2+1=O(n^2) time. The Horner's method takes O(n) time.

Q.14. What is dynamic programming? Give an example of dynamic programming.

Ans:

Q.15. Why is the recursive Fibonacci algorithm inefficient, but the nonrecursive Fibonacci

algorithm efficient?

The recursive Fibonacci algorithm is inefficient, because the subproblems in the recursive

Fibonacci algorithm overlaps, which causes redundant work. The non- recursive Fibonacci

algorithm is dynamic algorithm that avoids redundant work.

Q.16. Prove that the following algorithm for finding the GCD of the two integers m and n is

incorrect.

int gcd = 1;

for (int k = Math.min(Math.sqrt(n), Math.sqrt(m)); k >= 1; k--) {

if (m % k == 0 && n % k == 0) {

gcd = k;

break;

}

}

To prove this is wrong, all you need is to give a counter example to show the algorithm does

not work. Try n = 64 and m = 48. The algorithm will produce the gcd 8, but the actual gcd is

16.

Q.17. Prove that if n is not prime, there must exist a prime number p such that p <= sqrt(n) and p is

a factor of n.

If n is not a prime, then there exists two number n1 and n2 such that n1 \* n2 = n. Assume n1

<= n2, n1 <= srqt(n). If n1 is not a prime, you can continue the same process to find the

factors of n1, until a factor is a prime.

Q.18. Describe how the sieve of Eratosthenes is used to find the prime numbers.

Ans:

Q.19. What is the divide-and-conquer approach? Give an example.

Ans:

Q.20. What is the difference between divide-and-conquer and dynamic programming?

Ans:

Q.21. Can you design an algorithm for finding the minimum element in a list using divide-andconquer?

What is the complexity of this algorithm?

Yes. Finding the minimum in the first half and the second half of the list and return the

minimum of these two. So, the time complexity is O(n) = 2 \* O(n/2) + O(1) = O(n).

Q.22. What is backtracking? Give an example.

Ans:

Q.23. If you generalize the Eight Queens problem to the n-Queens problem in an n-by-n

chessboard, what will be the complexity of the algorithm?

O(n!)

Q.24. What is a convex hull?

Ans:

Q.25. Describe the gift-wrapping algorithm for finding a convex hull. Should list H be

implemented using an ArrayList or a LinkedList?

Ans:

Q.26. Describe Graham's algorithm for finding a convex hull. Why does the algorithm use a stack

to store the points in a convex hull?

Ans: