**Chapter 1 Data Structures and Algorithms: Instructor's CD questions**

1. The primary purpose of most computer programs is

a) to perform a mathematical calculation.

\*b) to store and retrieve information.

c) to sort a collection of records.

d) all of the above.

e) none of the above.

2. An integer is a

\*a) simple type

b) aggregate type

c) composite type

d) a and b

e) none of the above

3. ！！A payroll records is a

a) simple type

b) aggregate type

c) composite type

\*d) a and b

e) none of the above

4. ！！Which of the following should NOT be viewed as an ADT?

a) list

b) integer

c) array

\*d) none of the above

5. A mathematical function is most like a

\*a) Problem

b) Algorithm

c) Program

6. An algorithm must be or do all of the following EXCEPT:

a) correct

b) composed of concrete steps

\*c) ambiguous（模糊地。引起歧义的）

d) composed of a finite(限定的) number of steps

e) terminate

7. A solution is efficient if

a. it solves a problem within the require resource constraints.

b. it solves a problem within human reaction time.

c. it solves a problem faster than other known solutions.

d. a and b.

\*e. a and c.

f. b and c.

8. An array is

a) A contiguous (连续的)block of memory locations where each memory location

stores a fixed-length data item.

b) An ADT composed of a homogeneous collection of data items, each data item identified by a particular number.

c) a set of integer values.

\*d) a and b.

e) a and c.

f) b and c.

9. ！！Order the following steps to selecting a data structure to solve a problem.

(1) Determine the basic operations to be supported.

(2) Quantify the resource constraints for each operation.

(3) Select the data structure that best meets these requirements.

(4) Analyze the problem to determine the resource constraints that any

solution must meet.

a) (1, 2, 3, 4)

b) (2, 3, 1, 4)

c) (2, 1, 3, 4)

\*d) (4 ,1, 2, 3)

e) (1, 4, 3, 2)

10. ！！Searching for all those records in a database with key value between 10 and 100 is known as:

a) An exact match query.

\*b) A range query.

c) A sequential search.

d) A binary search.

**Chapter 2 Mathematical Preliminaries: Instructor's CD questions**

1. A set has the following properties:

a) May have duplicates(重复), element have a position.

b) May have duplicates, elements do not have a position.

c) May not have duplicates, elements have a position.

\*d) May not have duplicates, elements do not have a position.

2. A sequence has the following properties:

\*a) May have duplicates, element have a position.

b) May have duplicates, elements do not have a position.

c) May not have duplicates, elements have a position.

d) May not have duplicates, elements do not have a position.

3. ！！For set P, the notation |P| indicates

\*a) The number of elements in P.

b) The inverse of P.

c) The powerset of P.

d) None of the above.

4. Assume that P contains n elements. The number of sets in the power set of P is

a) n

b) n^2

\*c) 2^n

d) 2^n - 1

e) 2^n + 1

5. ！！If a sequence has n values, then the number of permutations(排列变化，按一定顺序组织起来) for that sequence will be

a) n

b) n^2

c) n^2 - 1

d) 2^n

\*e) n!

6. If R is a binary relation over set S, then R is reflexive(自反的) if

\*a) aRa for all a in S.

b) whenever aRb, then bRa, for all a, b in S.

c) whenever aRb and bRa, then a = b, for all a, b in S.

d) whenever aRb and aRc, then aRc, for all a, b, c in S.

7. If R is a binary relation over set S, then R is transitive(传递的) if

a) aRa for all a in S.

b) whenever aRb, then bRa, for all a, b in S.

c) whenever aRb and bRa, then a = b, for all a, b in S.

\*d) whenever aRb and aRc, then aRc, for all a, b, c in S.

8. R is an equivalence relation(等价关系) on set S if it is

\*a) reflexive, symmetric, transitive.

b) reflexive, antisymmetric, transitive.

c) symmetric, transitive.

d) antisymmetric, transitive.

e) irreflexive, symmetric, transitive.

f) irreflexive, antisymmetric, transitive.

9.！！ For the power set of integers, the subset operation defines

\*a) a partial order.

b) a total order.

c) a transitive order.

d) none of the above.

10. log nm is equal to

a) n + m

\*b) log n + log m

c) m log n

d) log n - log m

11. ！！A close-form solution is

a) an analysis for a program.

\*b) an equation that directly computes the value of a summation.

c) a complete solution for a problem.

12. Mathematical induction is most like

a) iteration.

\*b) recursion.

c) branching.

d) divide and conquer.

13. A recurrence relation（递归关系） is often used to model programs with

a) for loops.

b) branch control like "if" statements.

\*c) recursive calls.

d) function calls.

14. Which of the following is not a good proof technique.

a) proof by contradiction.(反证法)

\*b) proof by example.

c) proof by mathematical induction.

！！15. We can use mathematical induction to:

a) Find a closed-form solution for a summation.

\*b) Verify（证实） a proposed closed-form solution for a summation.

c) Both find and verify a closed-form solution for a summation.

**Chapter 3 Algorithm Analysis: Instructor's CD questions**

1. A growth rate applies to:

a) the time taken by an algorithm in the average case.

b) the time taken by an algorithm as the input size grows.

c) the space taken by an algorithm in the average case.

d) the space taken by an algorithm as the input size grows.

e) any resource you wish to measure for an algorithm in the average

case.

\*f) any resource you wish to measure for an algorithm as the input size grows.

2！！. Pick the growth rate that corresponds to the most efficient

algorithm as n gets large:

a) 5n

\*b) 20 log n

c) 2n^2

d) 2^n

3. Pick the growth rate that corresponds to the most efficient

algorithm when n = 4.

a) 5n（20）

b) 20 log n（40）

c) 2n^2（32）

\*d) 2^n（16）

Z

4.！！ Pick the quadratic（平方的） growth rate. 2n^2

a) 5n

b) 20 log n

\*c) 2n^2

d) 2^n

5. Asymptotic analysis（渐进算法分析） refers to:

a) The cost of an algorithm in its best, worst, or average case.

\*b) The growth in cost of an algorithm as the input size grows towards

infinity（无限的时间或空间 ）.

c) The size of a data structure.

d) The cost of an algorithm for small input sizes

6. For an air traffic control system, the most important metric is:

a) The best-case upper bound.

b) The average-case upper bound.

\*c) The worst-case upper bound.

d) The best-case lower bound.

e) The average-case lower bound.

f) The worst-case lower bound.

7.！！ When we wish to describe the upper bound for a problem we use:

\*a) The upper bound of the best algorithm we know.

b) The lower bound of the best algorithm we know.

c) We can't talk about the upper bound of a problem because there can

always be an arbitrarily slow algorithm.

8.！！ When we describe the lower bound for a problem we use:

a) The upper bound for the best algorithm we know.

b) the lower bound for the best algorithm we know.

c) The smallest upper bound that we can prove for the best algorithm

that could possibly exist.

\*d) The greatest lower bound that we can prove for the best algorithm

that could possibly exist.

9. When the upper and lower bounds for an algorithm are the same, we

use:

a) big-Oh notation.

b) big-Omega notation.

\*c) Theta notation.

d) asymptotic analysis.

e) Average case analysis.

f) Worst case analysis.

！！10. When performing asymptotic analysis, we can ignore constants and

low order terms because:

\*a) We are measuring the growth rate as the input size gets large.

b) We are only interested in small input sizes.

c) We are studying the worst case behavior.

d) We only need an approximation.

11. The best case for an algorithm refers to:

a) The smallest possible input size.

\*b) The specific input instance of a given size that gives the lowest cost.

c) The largest possible input size that meets the required growth rate.

d) The specific input instance of a given size that gives the greatest

cost.

12.！！ For any algorithm:

\*a) The upper and lower bounds always meet, but we might not know what

they are.

b) The upper and lower bounds might or might not meet.

c) We can always determine the upper bound, but might not be able to

determine the lower bound.

d) We can always determine the lower bound, but might not be able to

determine the upper bound.

13. If an algorithm is Theta(f(n)) in the average case, then it is:

a) Omega(f(n)) in the best case.

\*b) Omega(f(n)) in the worst case.

c) O(f(n)) in the worst case.

！！14. For the purpose of performing algorithm analysis, an important

property of a basic operation is that:

a) It be fast.

b) It be slow enough to measure.

c) Its cost does depend on the value of its operands(操作数).

\*d) Its cost does not depend on the value of its operands.

15. For sequential search,

a) The best, average, and worst cases are asymptotically the same.

\*b) The best case is asymptotically（渐进） better than the average and worst

cases.（最坏平均情况相等为n 跟系数无关）

c) The best and average cases are asymptotically better than the worst

case.

d) The best case is asymptotically better than the average case, and

the average case is asymptotically better than the worst case.

**Chapter 4 Lists, Stacks and Queues: Instructor's CD questions**

！！1. An ordered list is one in which:

a) The element values are in sorted order.

\*b) Each element a position within the list.

！！2. An ordered list is most like a:

a) set.

b) bag.

\*c) sequence.

一二题前后矛盾啊

3. As compared to the linked list implementation for lists, the

array-based list implementation requires:

a) More space

b) Less space

\*c) More or less space depending on how many elements are in the list.

4. Here is a series of C++ statements using the list ADT in the book.

L1.append(10);

L1.append(20);

L1.append(15);

If these statements are applied to an empty list, the result will look

like:

a) < 10 20 15 >

\*b) < | 10 20 15 >

c) < 10 20 15 | >

d) < 15 20 10 >

e) < | 15 20 10 >

f) < 15 20 10 | >(fence 位置始终在最前面)

5. ！！When comparing the array-based and linked implementations, the

array-based implementation has:

\*a) faster direct access to elements by position,

but slower insert/delete from the current position.

b) slower direct access to elements by position,

but faster insert/delete from the current position.

c) both faster direct access to elements by position, and faster

insert/delete from the current position.

d) both slower direct access to elements by position, and slower

insert/delete from the current position.

6. ！！For a list of length n, the linked-list implementation's prev function requires worst-case time:

a) O(1).

b) O(log n).

\*c) O(n).

d) O(n^2).

7. Finding the element in an array-based list with a given key value requires worst case time:

a) O(1).

b) O(log n).

\*c) O(n).

d) O(n^2).

！！8. In the linked-list implementation presented in the book, a header node is used:

\*a) To simpliy special cases.

b) Because the insert and delete routines won't work correctly without

it.

c) Because there would be no other way to make the current pointer

indicate the first element on the list.

9. When a pointer requires 4 bytes and a data element requires 4

bytes, the linked list implementation requires less space than the

array-based list implementation when the array would be:

a) less than 1/4 full.

b) less than 1/3 full.

\*c) less than half full.（当数组超过半满时，顺序表效率更高）

d) less than 2/3 full.

e) less than 3/4 full

f) never.

！！10. When a pointer requires 4 bytes and a data element requires 12

bytes, the linked list implementation requires less space than the

array-based list implementation when the array would be:

\*a) less than 1/4 full.

b) less than 1/3 full.

c) less than half full.

d) less than 2/3 full.

e) less than 3/4 full

f) never.

11. When we say that a list implementation enforces homogeneity,(同种) we

mean that:

a) All list elements have the same size.

\*b) All list elements have the same type.

c) All list elements appear in sort order.

12. When comparing the doubly and singly linked list implementations,

we find that the doubly linked list implementation(双链表节约了时间 浪费了额外的空间)

\*a) Saves time on some operations at the expense of additional space.

b) Saves neither time nor space, but is easier to implement.

c) Saves neither time nor space, and is also harder to implement.

13. We use a comparator() function in the Dictionary class ADT:

a) to simplify implementation.

\*b) to increase the opportunity for code reuse.

c) to improve asymptotic efficiency of some functions.

14.！！ All operations on a stack can be implemented in constant time

except:

a) Push

b) Pop

c) The implementor's choice of push or pop (they cannot both be

implemented in constant time).

\*d) None of the above.

15.！！ Recursion is generally implemented using

a) A sorted list.

\*b) A stack.

c) A queue.

**Chapter 5 Binary Trees: Instructor's CD questions**

1. The height of a binary tree is:

a) The height of the deepest node.

b) The depth of the deepest node.

\*c) One more than the depth of the deepest node.

2. A full binary tree is one in which:

\*a) Every internal node has two non-empty children.

b) all of the levels, except possibly the bottom level, are filled.(complete binary tree)

3. The relationship between a full and a complete binary tree is:

a) Every complete binary tree is full.

b) Every full binary tree is complete.

\*c) None of the above.

4. The Full Binary Tree Theorem （原理）states that:

\*a) The number of leaves in a non-empty full binary tree is one more

than the number of internal nodes.

b) The number of leaves in a non-empty full binary tree is one less

than the number of internal nodes.

c) The number of leaves in a non-empty full binary tree is one half of

the number of internal nodes.

d) The number of internal nodes in a non-empty full binary tree is one

half of the number of leaves.

5. The correct traversal to use on a BST to visit the nodes in sorted order is:

a) Preorder traversal.

\*b) Inorder traversal.

c) Postorder traversal.

6. When every node of a full binary tree stores a 4-byte data field,

two 4-byte child pointers, and a 4-byte parent pointer, the overhead fraction(结构性开销) is approximately:((4+4\*2)/(4+4\*2+4))

a) one quarter.

b) one third.

c) one half.

d) two thirds.

\*e) three quarters.

f) none of the above.

7. When every node of a full binary tree stores an 8-byte data field and

two 4-byte child pointers, the overhead fraction(结构性开销) is approximately:

a) one quarter.

b) one third.

\*c) one half.

d) two thirds.

e) three quarters.

f) none of the above.

！！8. When every node of a full binary tree stores a 4-byte data field

and the internal nodes store two 4-byte child pointers, the

overhead fraction is approximately:

a) one quarter.

b) one third.

\*c) one half.

d) two thirds.

e) three quarters.

f) none of the above.

9. If a node is at position r in the array implementation for a complete binary tree, then its parent is at:

\*a) (r - 1)/2 if r > 0

b) 2r + 1 if (2r + 1) < n

c) 2r + 2 if (2r + 2) < n

d) r - 1 if r is even

e) r + 1 if r is odd.

！！10. If a node is at position r in the array implementation for a complete binary tree, then its right child is at:

a) (r - 1)/2 if r > 0

b) 2r + 1 if (2r + 1) < n

\*c) 2r + 2 if (2r + 2) < n

d) r - 1 if r is even

e) r + 1 if r is odd.

！！11. Assume a BST is implemented so that all nodes in the left subtree of a given node have values less than that node, and all nodes in the right subtree have values greater than or equal to that node. When implementing the delete routine, we must select as its replacement:

a) The greatest value from the left subtree.

\*b) The least value from the right subtree.

c) Either of the above.

！！12. Which of the following is a true statement:

a) In a BST, the left child of any node is less than the right child,

and in a heap, the left child of any node is less than the right

child.

\*b) In a BST, the left child of any node is less than the right child,

but in a heap, the left child of any node could be less than or

greater than the right child.

c) In a BST, the left child of any node could be less or greater than

the right child, but in a heap, the left child of any node must be

less than the right child.

d) In both a BST and a heap, the left child of any node could be

either less than or greater than the right child

！！13. When implementing heaps and BSTs, which is the best answer?

a) The time to build a BST of n nodes is O(n log n), and the time to

build a heap of n nodes is O(n log n).

b) The time to build a BST of n nodes is O(n), and the time to

build a heap of n nodes is O(n log n).

\*c) The time to build a BST of n nodes is O(n log n), and the time to

build a heap of n nodes is O(n).

d) The time to build a BST of n nodes is O(n), and the time to

build a heap of n nodes is O(n).

！！14. The Huffman coding tree works best when the frequencies for letters are

a) Roughly the same for all letters.

\*b) Skewed so that there is a great difference in relative frequencies

for various letters.

！！15. Huffman coding provides the optimal(最理想的；最佳的) coding when:

a) The messages are in English.

b) The messages are binary numbers.

\*c) The frequency of occurrence for a letter is independent of its context within the message.

d) Never.