Trung tâm Sát hạch Công nghệ thông tin và Hỗ trợ đào tạo

VITEC

http://www.vitec.org.vn

Contents

1 Data Structures

Chapter Objectives

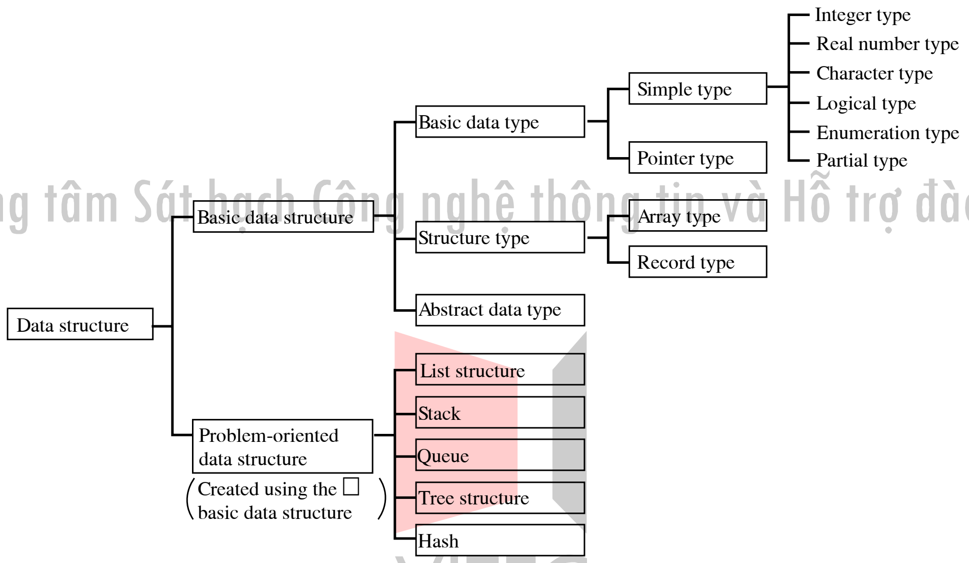
Choosing the most appropriate data structure and data description procedure is the key to creating an efficient, easy-to-understand program.
This chapter describes various data structures you must understand as the first step to learning programming.

① Understanding how various data structures can be classified
② Understanding the most commonly used, basic data types and data arrays
③ Understanding the characteristics and mechanisms of problem-oriented data structures used to solve specific problems, as well as how to use a basic data structure for program implementation

1.1 What is the data structure?

A set of the same kind of data processed by a computer is called a "data type." In the program design stage, the way data should be represented and programmed into a computer must be examined carefully, so that the most appropriate data type can be chosen. A data type that is represented and programmed is called a "data structure."
Figure 1-1-1 shows the classification of data structures.

Figure 1-1-1 Classification of data structures



The basic data structure can be represented in almost all programming languages. The problem-oriented data structure is a data structure that can be effectively used to solve specific problems. There are some problem-oriented data structures that cannot be represented in programming languages. In that case, the basic data structure is used.

1.2 Basic data structure

1.2.1 Basic data type

The basic data type is a set of individual data and is frequently used to create a program. It is classified into simple and pointer types.

(1) Simple type

The simple type is the most basic data type. When using the simple type for programming, the data type is usually declared according to the syntax rule of a language.

① Integer type

The integer type represents integers, and is represented inside a computer as binary numbers of fixed-point numbers that have no significant digits below the decimal point. A maximum or minimum value of the integer type is the unit of data that a computer can process at one time, and it is determined by the length of one word.

② Real number type

The real number type represents real numbers. It is used to represent fixed-point and floating-point numbers.

③ Character type

The character type represents alphabets, numerals and symbols as characters. A character code is expressed as a binary number inside a computer.

④ Logical type

The logical type is used to perform logical operations, such as AND, OR and NOT operations.

⑤ Enumeration type

The enumeration type is defined as a data type that enumerates all possible values of variables. In the case of the enumeration type, integers can be named.

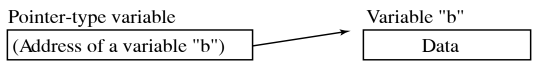
⑥ Partial type

The partial type is used to specify an original-value subset by constraining existing data types. The data type that has upper and lower limits specified as constraints is called the partial range type.

(2) Pointer type

The pointer type has an address allocated in a main memory unit. It is used to refer to variables, file records or functions. It is used for Pascal and C but not for FORTRAN and COBOL.

Figure 1-2-1 Image of the pointer type



1.2.2 Structured type

A data structure that contains a basic data structure or any of defined data types as its elements (data), is called the structured type. The structured type is classified into the array type and record type.

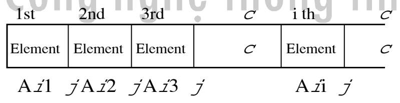
(1) Array type

An array is called a table. The array type is a data structure that contains data of the same type and size. Each individual data is called an array element, a table element or an element. How arrays are described or how data is arranged varies, depending on the programming language used.

① One-dimensional array

A one-dimensional array has a data structure in which data is arrayed in a line. To specify an element in this array, first enter a left parenthesis (or left bracket [after the name of an array, then enter a subscript and a right parenthesis) or right bracket]. A subscript, also called an index, indicates the ordinal number from the top of an array where a specified element is located. An array "A" having the number of elements denoted by "i" is expressed as A (i).

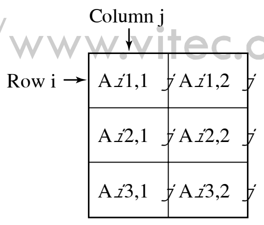
Figure 1-2-2 One-dimensional array



② Two-dimensional array

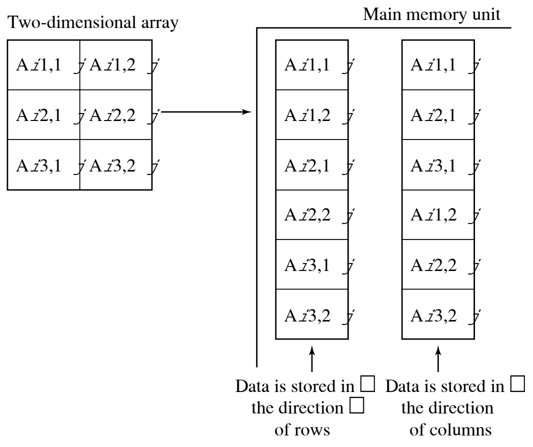
A data structure in which data is lined up in both vertical and horizontal directions is called a two-dimensional array. Data in a vertical direction is called a column and data in a horizontal direction is called a row. To specify a certain element in this array, two subscripts become necessary: one subscript showing in an ordinal number in a vertical direction (in what row) where a specified element is located and the other subscript showing in what ordinal number in a horizontal direction (in what column) it is located. An array "A" located in the "i" row and "j" column, for example, can be expressed as A (i, j).

Figure 1-2-3 Two-dimensional array (with three rows and two columns)



When a two-dimensional array is stored in a main memory unit, it takes the form of a one-dimensional array. The two-dimensional array shown in Figure 1-2-3 takes the form of a one-dimensional array having six elements. As shown in Figure 1-2-4, data is stored sequentially in either the direction of rows or the direction of columns. The direction in which data is stored varies, depending on the compiler of the programming language used. In general, data is stored vertically when Fortran is used and horizontally when COBOL is used.

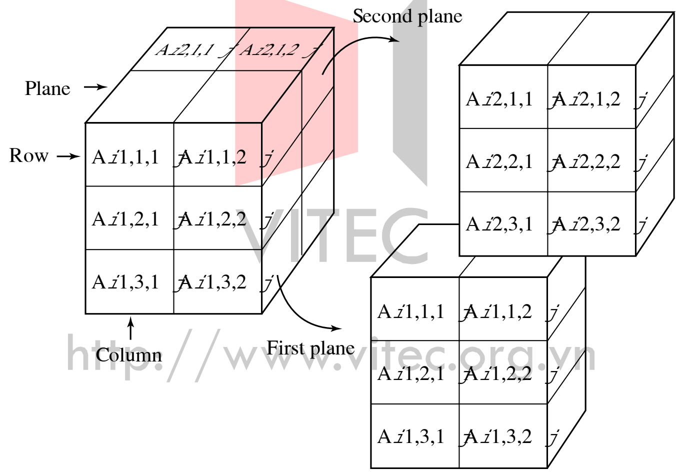
Figure 1-2-4 How data of a two-dimensional array is stored in a main memory unit


Two-dimensional array
A（1,1） A（1,2）
A（2,1） A（2,2）
A（3,1） A（3,2）
Main memory unit
A（1,1） A（1,2） A（2,1） A（2,2） A（3,1） A（3,2）
A（1,1） A（2,1） A（3,1） A（1,2） A（2,2） A（3,2）
Data is stored in the direction of rows
Data is stored in the direction of columns

③ Three-dimensional array

A three-dimensional array has a data structure in which more than one two-dimensional array is contained. It has a three-dimensional structure comprising planes, rows and columns as elements. By developing a three-dimensional array into a two-dimensional one, a three dimensional array can be handled in the same way as a two-dimensional array.

Figure 1-2-5 Developing a three-dimensional array into a two-dimensional array



Multidimensional arrays such as four-, five-, or n-dimensional arrays can be defined. However, there may be certain limitations to the number of definable dimensions, depending on the type of programming language or compiler.

Arrays can be classified into static and dynamic arrays according to the method used to secure an area.

- Static array: An array for which a required area is determined by a program
- Dynamic array: An array for which a required area is determined after a subscript used for arraying is provided with an expression and the expression is evaluated during execution of a program

(2) Record type

Although structured type data is superior in the ease of element reference and operation, it has a drawback in that it can handle only data of the same type. Therefore, data that contains data of different types must take the form of record-type data. This record type is also called a structure.

Figure 1-2-6 Record type

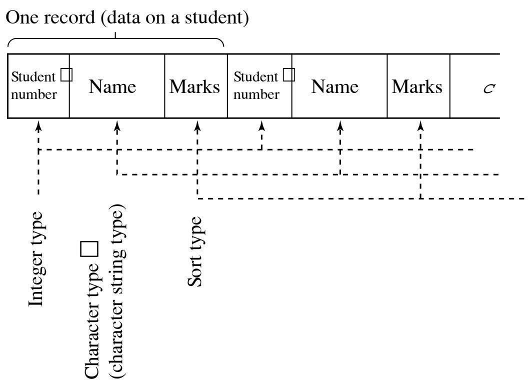
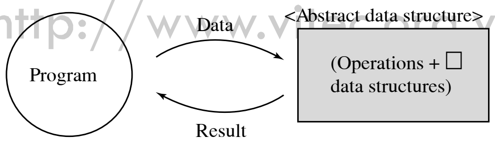

One record (data on a student)
Student number
Name
Marks
Student number
Name
Marks
Integer type
Character type (character string type)
Sort type

Figure 1-2-6 shows the data structure of the record type. One record contains a student number (integer type), a name (character type) and marks (integer type). One record type data contains a set of records of this format. Although one-dimensional record type data can be handled in the same way as a one-dimensional array, each data must be named for identification since each element contains more than one data.

1.2.3 Abstract data type

Data that contains a certain data structure and type of operations is called an abstract data type. To access this type of data, you do not need to be aware of its internal structure. All data are hidden except the data that you access for reference, insertion or deletion. This is called information hiding. Hiding information or hiding data on the level of data types is called data encapsulation.

Figure 1-2-7 Abstract data type



1.3 Problem-oriented data structure

Various problem-oriented data structures can be designed using the array type, pointer type and other basic data structures.

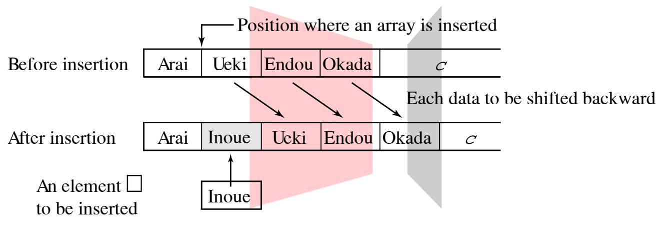
1.3.1 List structure

Unlike the basic data type that handles individual data, a list structure allows data to be linked to each other and handled in one lump. Data arranged according to this list structure is called a list.

(1) List structure and cells

Using a subscript for each element in an array, quick access to any element is possible. Also, a change to data can be made easily. If you insert one piece of data somewhere in arrays, you must shift each of all pieces of data subsequent to the inserted element backward. If you delete one piece of data in arrays, you must likewise shift each of all pieces of data subsequent to the deleted piece forward.

Figure 1-3-1 Inserting an array element


Position where an array is inserted
Before insertion
Arai
Ueki
Endou
Okada
Each data to be shifted backward
After insertion
Arai
Inoue
Ueki
Endou
Okada
An element to be inserted
Inoue

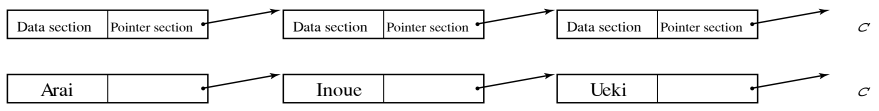
Unlike the array type structure, the list structure allows data element to be inserted or deleted easily. The list structure is similar to the array structure in that data elements of the same type is sequentially lined up. The array type requires that the logical arrangement of elements is the same with the physical arrangement of elements stored in a main memory unit. In the case of the list structure, the logical arrangement does not need to match the physical arrangement.

The list contains cells and each cell consists of the following:

- Data section that contains data element
- Pointer section that contains an address

Therefore, the data section of a cell has the same data structure as that of stored data and the pointer section of a cell has a pointer-type data structure. This means that cells represent record-type (structure) data that contain elements of different data structures. The list contains cell addresses in the pointer section and one cell is linked to another cell via a pointer.

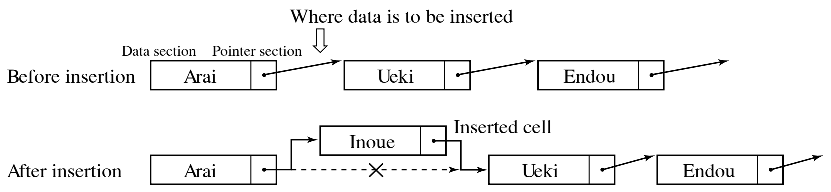
Figure 1-3-2 List structure



(2) Inserting data into a list

To insert data into a list, all you have to do is to replace pointers preceding and subsequent to data to be inserted. Because you do not have to shift elements as in the case of array-type data, you can insert data easily and quickly. (See Figure 1-3-3.)

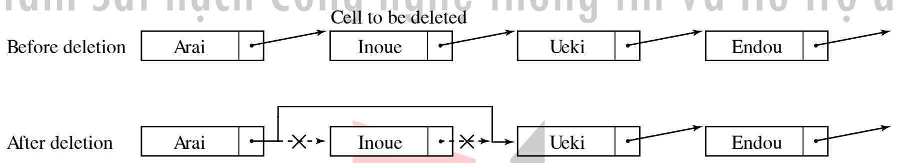
Figure 1-3-3 Inserting data into a list



(3) Deleting data from a list

To delete data from a list, all you have to do is to replace pointers as you do to insert data into a list. Cells that contain data (Inoue) remain in a memory area as garbage after data is deleted, as shown in Figure 1-3-4.

Figure 1-3-4 Deleting data from a list



Although the list structure allows data to be inserted or deleted by simply replacing pointers, it has the drawback that you must track each pointer one by one from the top one if you want to access specific data.

(4) Types of list structures

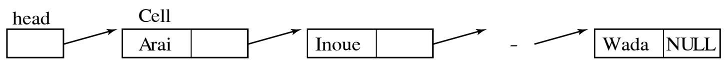
Representative list structures include:
- Uni-directional list
- Bi-directional list
- Ring list.

Because these lists are represented in the form of a straight line, they are generically called linear lists.

① Uni-directional list

The uni-directional list is also called a one-way list. The pointer section of a cell contains the address of a cell in which the next data is stored. By tracking these addresses one by one, you can perform a search on data.

Figure 1-3-5 Uni-directional list

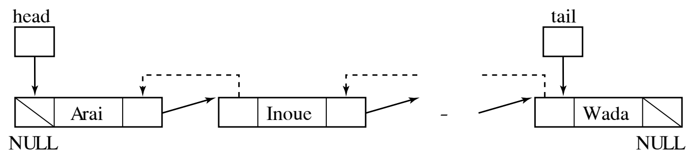


The first pointer is called a root or head. Because the pointer section of the last cell does not have any address in which data can be stored, NULL (numerical value of zero) or ＼is entered in this section.

② Bi-directional list

The bi-directional list has two pointer sections (◇and◆) which contain cell addresses as shown in Figure 1-3-6.

Figure 1-3-6 Bi-directional list

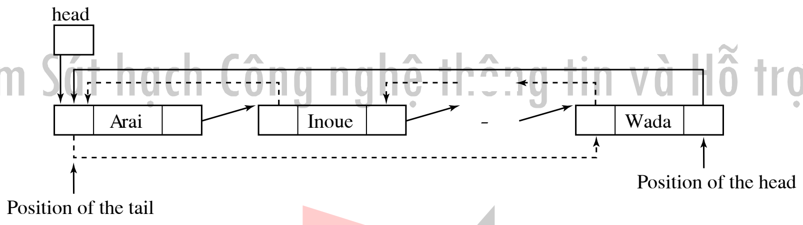


The pointer sections ◇ and ◆ shown in Figure 1-3-6 contain the address of the subsequent cell and the address of the preceding cell respectively. The address of the last cell is contained in the pointer tail. In the case of the bi-directional list, data can be tracked from either the head or the tail of cells.

③ Ring list

A bi-directional list containing NULL in the first cell is called a ring list. The pointer section of this first cell and the pointer section containing NULL of the last cell contain addresses of one another, thus data is in the form of a ring. Data can be searched in the same way as in the bi-directional list.

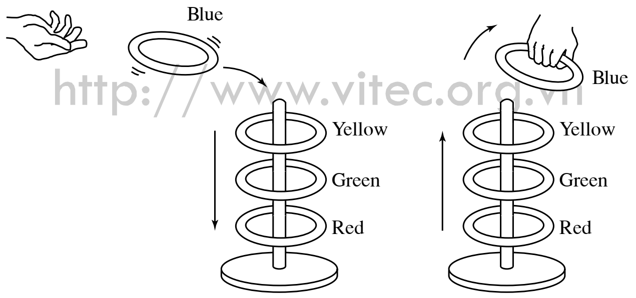
Figure 1-3-7 Ring list



1.3.2 Stack

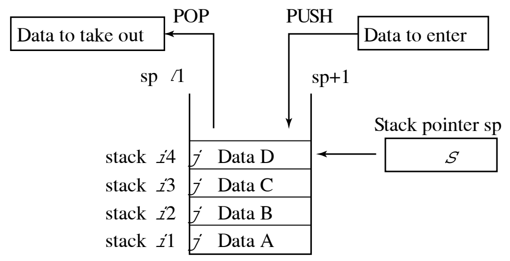
A stack is a data structure designed based on one-dimensional arrays. Last stored data is read first. It is likened to the game of quoits.

Figure 1-3-8 Quoits



Quoits are thrown in the order of red, green, yellow and blue (data input). They are retrieved one by one (data output) in reverse order of throwing, i.e., blue, yellow, green and red. That is, a blue quoit last thrown is first retrieved.
This type of data structure which can be likened to the game of quoits is called a stack. This system is termed the last-in first-out (LIFO) system. Storing data in a stack is called "push down (PUSH)" and taking data from a stack is called "pop up (POP)." A variable that controls pushing down and popping up of data is called a stack pointer.
To push down data into a stack, set the stack pointer "sp" to +1 and store data in array elements indicated by "sp." To pop up data from a stack, take out data stored in array elements indicated by "sp" and set the stack pointer to sp-1.

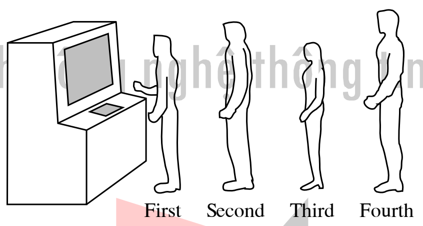
Figure 1-3-9 Stack structure



1.3.3 Queue

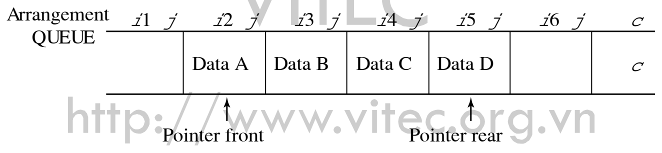
A queue is a data structure designed based on one-dimensional arrays. Data first stored is first read. It is likened to a queue of people who are waiting in front of a cash dispenser machine of a bank.

Figure 1-3-10 Queue



A data structure that allows customers to be served on the first-come first-served basis is called a queue. This system is called the first-in first-out (FIFO) system. Two pointers that indicate the head and tail of a queue are required for queue control. Pointers that indicate the head and tail of a queue are represented as a front variable and a rear variable respectively. (See Figure 1-3-11.)

Figure 1-3-11 Queue structure



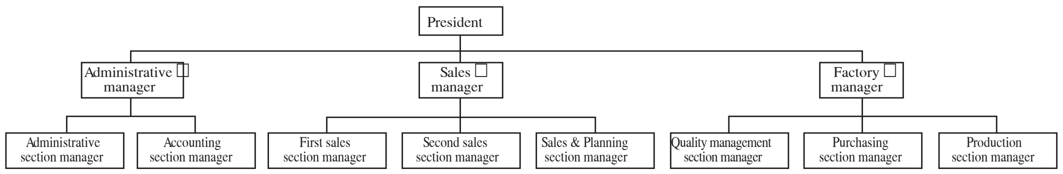
<Queue operation procedure>

1. To store data in a queue (enqueue), set a rear variable of a pointer indicating the tail to +1 and store data.
2. To take out data from a queue (dequeue), take out data and set a front variable of a pointer indicating the head to +1.

1.3.4 Tree structure

The tree structure is one of very useful data structures since it can control complex data better than array- or list-type data structures.
Because it has a hierarchical structure like a company organization or a family tree, it is suited for the type of work involving step-by-step classification.

Figure 1-3-12 Organization chart

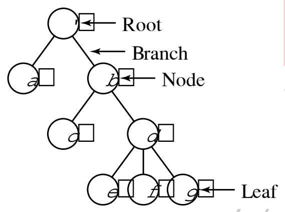

President
Administrative manager
Sales manager
Factory manager
Administrative section manager
Accounting section manager
First sales section manager
Second sales section manager
Sales & Planning section manager
Quality management section manager
Purchasing section manager
Production section manager

Although each cell is linearly correlated in the list structure, data is correlated while it branches off in the tree structure.
The tree structure consists of elements shown below:

- Node:	Correspond to data
- Branch:	Connecting one node to the other
- Root:	Highest-order node that has no parent
- Child:	Node that branches off under a node
- Parent:	Original data before it begins to branch
- Leaf:	Lowest-order node that has no child

Figure 1-3-13 shows the tree structure.

Figure 1-3-13 Tree structure



ÆA parent of the node C is the node A.
ÆChildren of the node C are the nodes D and E.

(1) Binary tree

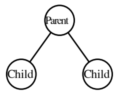
If the number of branches branching off from a node is "n" or less, that is, if the number of children is 0 to "n," such a tree structure is called an N-ary tree. An N-ary tree that has two branches (n=2), that is, an N-ary tree that has no child, one child or two children is called a binary tree, which is the most commonly used data structure. On the other hand, a tree structure that has three or more branches (n>2) is called a multiway tree.
Binary-tree data consists of one data section and two pointer sections. The left pointer section shows the position of a node that extends to the left and branches off while the right pointer section shows the position of a node that extends to the right and branches off.

Figure 1-3-14 Binary-tree data structure

Left pointer section	Data section	Right pointer section

The binary tree has a parent-child hierarchical structure, as shown in Figure 1-3-15.

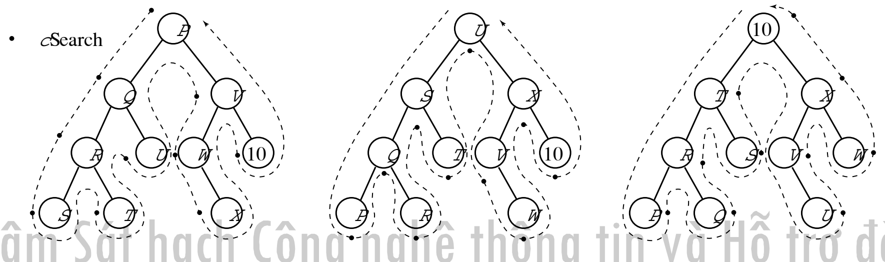
Figure 1-3-15 Basic binary-tree structure



<How to perform a search on binary-tree data>

To find specific data in binary-tree data, each node must be tracked one by one. There are three methods to perform a search on binary-tree data. As it is difficult to explain them in words, check Figure 1-3-16 and see how data is searched using each method.

Figure 1-3-16 How to perform a search on binary-tree data



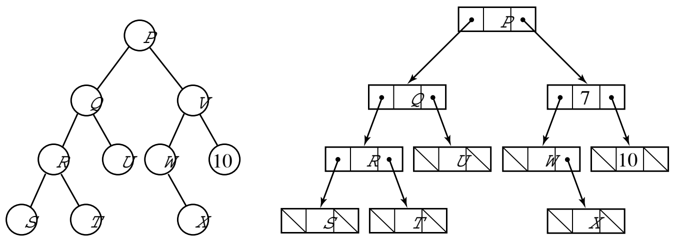
<Pre-order> <Mid-order> <Post-order>

- Pre-order: With a root as a starting point, the left side of each node is tracked in a sequential way.
- Mid-order: With a leaf at the bottom left as a starting point, the underside of each node is tracked in a sequential way.
- Post order: With a leaf at the bottom left as a starting point, the right side of each node is tracked in a sequential way.

(2) Perfect binary tree

If a binary tree is constructed in such a way that the number of branches from a root to each leaf along one branch is equal to or different by one from that of branches from a root to each leaf along another branch (or if the height from a root to each leaf along one branch is equal to or different by one from that from a root to each leaf along another branch), it is called a perfect binary tree. Figure 1-3-17 shows a perfect binary tree that has ten nodes.

Figure 1-3-17 Perfect binary tree



<Perfect binary tree> <Data structure>

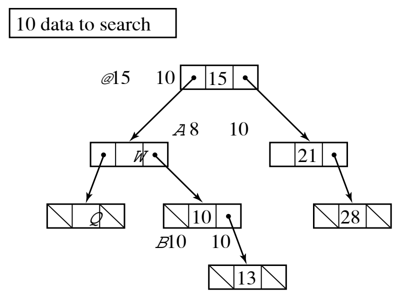
(3) Binary search tree

A binary search tree is used as a variant of a binary tree. In the case of a binary search tree, a descendant to the left is smaller than a parent and a descendant to the right is larger than a parent.
The algorithm of a binary search tree is as follows:

1. A root is a point where a search starts.
2. Binary-tree data is compared with data to search.
3. If binary-tree data = data to search, a search is successful (completed).

4. If binary-tree data > data to search, nodes to the left of a binary tree are searched and compared.
5. If binary-tree data < data to search, nodes to the right of a binary tree are searched and compared.
6. If no child is found, a search is unsuccessful (no data is found).

Figure 1-3-18 Performing a search on data in a binary search tree



(4) Balanced tree

If data is added or deleted in a tree structure, leaves randomly develop and the operation efficiency will decrease. A tree structure capable of reorganizing itself is called a balanced tree. Representative balanced trees are a B-tree and a heap.

① B-tree

A B-tree is a further developed version of a binary tree:

- End leaves are removed and each node has a number of nodes specified as "m."
- Each node has a maximum number of data specified as "m-1."
- All leaves are on the same level.
- Data contained in each node is arranged in a queue

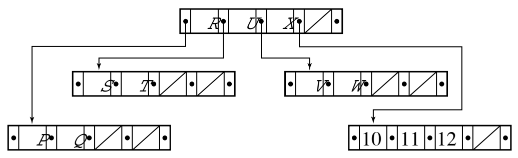
Because the above characteristics can increase the memory and calculation efficiency, the name "B-tree" means a well-balanced tree.

a. Characteristics of a B-tree

- To increase the memory area usage, the number of pointers that each node has is set at m/2 or more and m or less. The number of pointers along a route, however, is set at 2 or more.
- Each time data is deleted or added, splitting and concatenation are automatically executed so that the number of elements of a node can become m/2 or more or m or less. This allows leaves to be always maintained on the same level.
- A search starts from a root.
- Addition or deletion of data starts from a leaf.

Figure 1-3-19 shows a quint-B-tree with elements ｜7, 6, 10, 2, 5, 12, 4, 9, 8, 11, 1, 3｜.

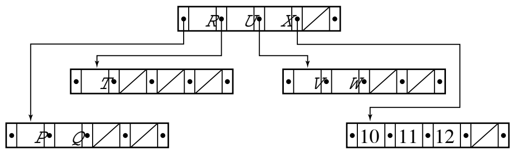
Figure 1-3-19 Quint-B-tree



b. Deleting data

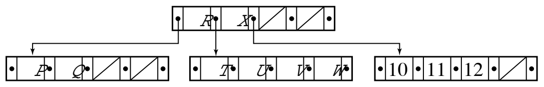
Figure 1-3-20 shows a case where the data "4" is deleted from a B-tree shown in Figure 1-3-19.
If "4" alone is deleted, the number of deleted elements of a node becomes one and the requirements for a B-tree cannot be met.

Figure 1-3-20 Wrong B-tree



The node from which "4" is deleted is merged with an adjacent node either to the right or to the left. Because the pointers of a parent must also be reorganized, "6" is moved to a lower-order node and each node can be reorganized, as shown in Figure 1-3-21.

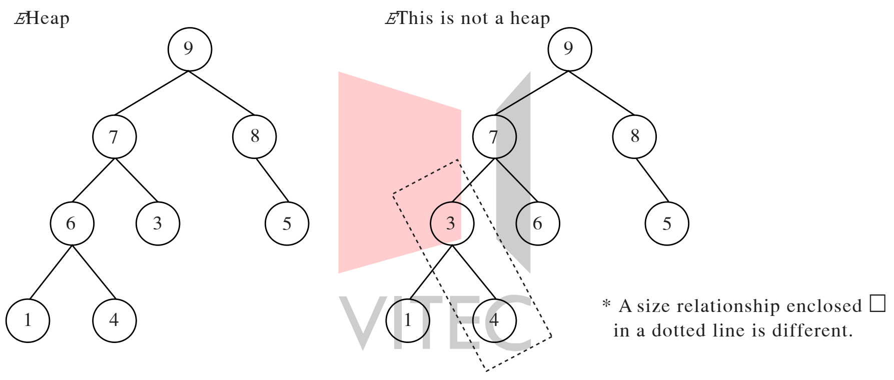
Figure 1-3-21 Correct B-tree



② Heap

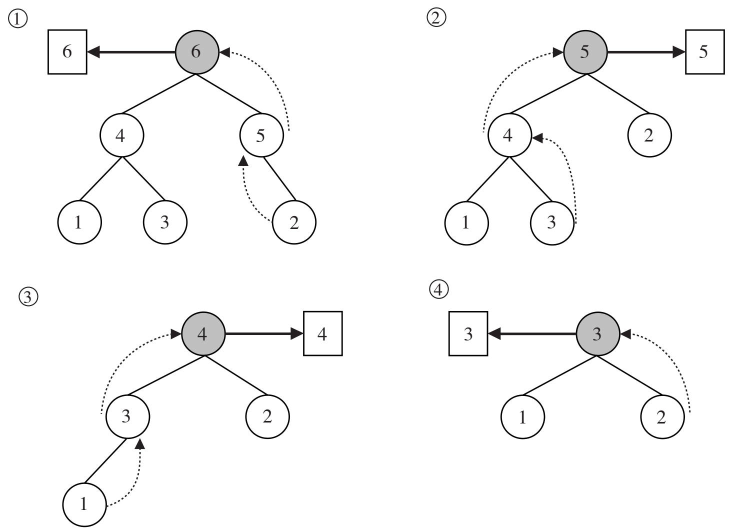
A perfect binary tree that has a certain size relationship between a parent node and a child node is called a heap. A heap is different from a binary search tree in that a heap does not have a certain size relationship between brothers.

Figure 1-3-22 Example of a heap


Heap
This is not a heap
* A size relationship enclosed □ in a dotted line is different.

Because a heap has the structure of a perfect binary tree, it is a kind of organizable, balanced tree. In the case of a heap, maximum values (or minimum values) of all data are recorded in a root Using this characteristic, data can be sorted by taking out data from a root in a sequential way.

Figure 1-3-23 Sorting data using a heap



1.3.5 Hash

A hash is a way of using an array-type data structure. Using a hash, you can directly access specific data using a key without accessing recorded data one by one.

(1) Converting to a hash address

To convert a key to a hash address (subscript), a hash function is used. Representative hash functions are:

- Division method
- Registration method
- Base conversion method.

① Division method

A key is divided by a certain value (a prime number closest to the number of array elements is usually used) and the remainder is used as the address (subscript) of a key.

Example Key: 1234 and the number of array elements: 100
1234÷97 (a prime number closest to 100) = 12 70 : Address (subscript)

② Registration method

A key is decomposed according to a certain rule and the total is used as the address (subscript) of a key.

Example Key: 1234
1234 is decomposed into 12 and 34 and both numbers are totaled.
12 + 34 = 46 : Address (subscript)

③ Base conversion method

A key is normally represented in decimal numbers. This key is converted into a radix other than decimal numbers (tertiary numbers, for example) and the result is used as the address (subscript) of a key.

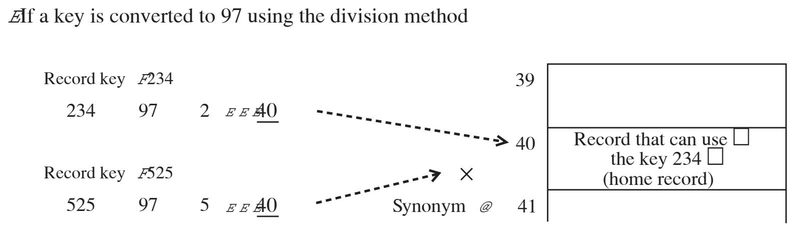
Example Record key: 1234
$1 \times 3^3 + 2 \times 3^2 + 3 \times 3^1 + 4 \times 3^0 = 27 + 18 + 9 + 4$
$= 58$: Address (subscript)

∗ The value of each digit of tertiary numbers is 3 or smaller. In this case, however, this fact does not need to be considered.

(2) Synonym

When a key is converted to an address using a hash function, different keys can be converted to the same address. This is called a synonym (collision) (see Figure 1-3-24). A record that can use the converted address is called a home record and a record that cannot use it is called a synonym record.

Figure 1-3-24 Occurrence of a synonym


If a key is converted to 97 using the division method
Record key 234
234 97 2 40
Record key 525
525 97 5 40
39
40
41
Record that can use the key 234 (home record)
Synonym
×

To deal with a synonym, a sequential method or a chain method is used:

① Sequential method

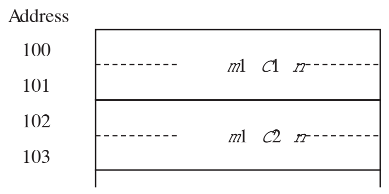
Using a sequential method, a synonym record is stored in free space that is located close to the desired address. There is the possibility that synonyms may occur in a chain reaction.

② Chain method

Using a chain method, a separate memory area is established and synonym records are stored in this area. In this case, it is necessary to provide a pointer that shows the address where synonym records are stored.

Exercises

Q1 **When storing a two-dimensional array "a" with ten rows and ten columns in continuous memory space in the direction of rows, what is the address where a [5, 6] is stored? In this question, the address is represented in decimal numbers.**



a. 145 b. 185 c. 190 d. 208 e. 212

Q2 **The figure below is a uni-directional list. Tokyo is the head of this list and the pointer contains addresses of data shown below. Nagoya is the tail of the list and the pointer contains 0. Choose one correct way to insert Shizuoka placed at address 150 between Atami and Hamamatsu.**

Pointer for the head
10

Address	Data	Pointer
10	Tokyo	50
30	Nagoya	0
50	Shin Yokohama	90
70	Hamamatsu	30
90	Atami	70
150	Shizuoka	

a. The pointer for Shizuoka to be set to 50 and that for Hamamatsu to be set to 150
b. The pointer for Shizuoka to be set to 70 and that for Atami to be set to 150
c. The pointer for Shizuoka to be set to 90 and that for Hamamatsu to be set to 150
d. The pointer for Shizuoka to be set to 150 and that for Atami to be set to 90

Q3 **What is the data structure suitable for the FIFO (first-in first-out) operation?**

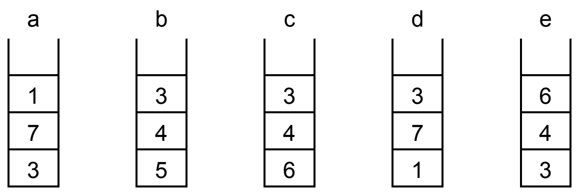
a. Binary tree b. Queue c. Stack d. Heap

Q4 **Two stack operations are defined as follows:**

PUSH n: Data (integer "n") is pushed into a stack.
POP: Data is popped out of a stack.

If a stack operation is performed on an empty stack according to the procedure shown below, what result can be obtained?

PUSH 1 → PUSH 5 → POP → PUSH 7 → PUSH 6 → PUSH 4 →POP → POP → PUSH 3



Q5 **Figure 2 is the array representation of a binary tree shown in Figure 1. What value should be put into the space "a"?**

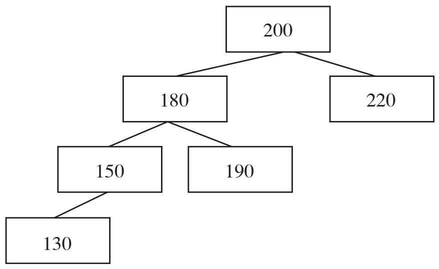


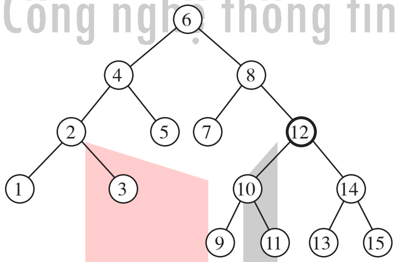
Figure 1 Binary tree

Subscript	Value	Pointer 1	Pointer 2
1	200	3	2
2	220	0	0
3	180	5	$\boxed{a}$
4	190	0	0
5	150	6	0
6	130	0	0

Figure 2
Array representation of a binary tree

a. 2 b. 3 c. 4 d. 5

Q6 **When the element 12 is deleted from the binary search tree shown below, which element should be moved to the point where the element was deleted to reorganize a binary search tree?**

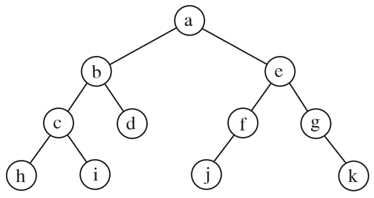


a. 9 b. 10 c. 13 d. 14

Q7 **If two or less than two branches branch off from each node of a tree, such a tree is called a binary tree. A binary tree consists of one node, a left tree and a right tree. There are three methods of performing a search on this tree:**

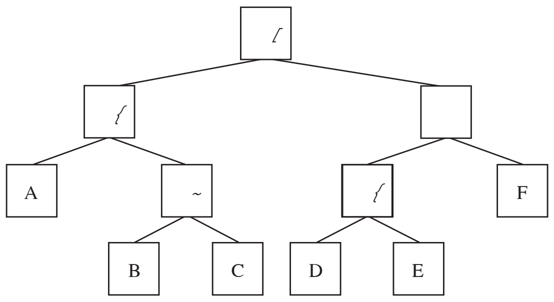
(1) Pre-order: A search is performed in the order of a node, a left tree and a right tree.
(2) Mid-order: A search is performed in the order of a left tree, a node and a right tree.
(3) Post-order: A search is performed in the order of a left tree, a right tree and a node.

If a search is performed using the post-order method, which can be output as the value of a node?



a. abchidefjgk b. abechidfjgk c. hcibdajfegk d. hicdbjfkgea

Q8 **A binary tree shown below can be represented using an arithmetic expression. Which expression is correct?**

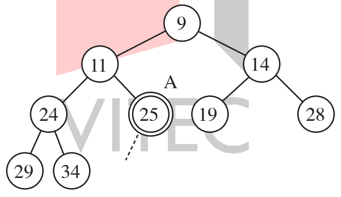


a. A + B × C + (D + E) ÷ F
b. A + B × C - (D + E) ÷ F
c. A + B × C - D + E ÷ F
d. A × B + C + (D - E) ÷ F
e. A × B + C - D + E ÷ F

Q9 **Regarding storing data using a B-tree, choose one correct comment from the following:**

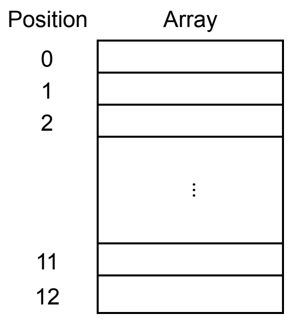
a. Split and merge nodes to allow the hierarchical depth to become the same.
b. Identify the position where data is to be stored by using a certain function and key value.
c. Only sequential access to head data and subsequent data is possible.
d. There are a directory and a member. A member is a sequentially organized file.

Q10 **There is a heap with the value of a parent's node smaller than that of a child's node. To insert data into this heap, you can add an element to the backmost part and repeat exchanging a parent with a child when the element is smaller than a parent. When element 7 is added to the position * of the next heap, which element comes to the position A?**



a. 7   b. 9   c. 11   d. 24   e. 25

Q11 **A five-digit number (a_1 a_2 a_3 a_4 a_5) must be stored in an array using the hash method. If the hash function is defined as mod ($a_1+a_2+a_3+a_4+a_5$, 13) and if the five-digit number is to be stored in an array element located in the position that matches a calculated hash value, in which position will 54321 be placed in an array shown below? Here, the mod (x, 13) value is the remainder when x is divided by 13.**



a. 1   b. 2   c. 7   d. 11

Q12 **Five data with key values distributed uniformly and randomly in the range of 1 to 1,000,000 must be registered in a hash table of 10 in size. What is the approximate probability that collisions occur? Here, the remainder obtained when a key value is divided by the size of a hash table is used as a hash value.**

a. 0.2 b. 0.5 c. 0.7 d. 0.9

Answers to Exercises

Answers for No.3 Chapter1 (Data Structures)

Answer list

Answers

Q 1:	c	Q 2:	b	Q 3:	b	Q 4:	d	Q 5:	c
Q 6:	c	Q 7:	d	Q 8:	b	Q 9:	a	Q 10:	c
Q 11:	b	Q 12:	c						

Answers and Descriptions

Q1

Answer

c. 190

Description

The position of a[5,6] in the array is as follows.

a[1,1]	a[1,2]		…				…		a[1,10]
a[2,1]	a[2,2]		…				…		a[2,10]
a[3,1]	a[3,2]		…				…		a[3,10]
a[4,1]	a[4,2]		…				…		a[4,10]
a[5,1]	a[5,2]	a[5,3]	a[5,4]	a[5,5]	a[5,6]				

That is, it comes in the 46th.

The 2nd one (a[1,2])'s address is 100 + 2*1=102

and the third one(a[1,3])'s address is 100 + 2*2=104.

Therefore, the 46th one's address is

100 + 2*45=190

Q2

Answer

b. The pointer for Shizuoka to be set to 70 and that for Atami to be set to 150

Description

This unidirectional list can be visualized as follows.

10 Tokyo-->50 ShinYokohama-->90 Atami-->70 Hamamatsu-->30 Nagoya

After inserting Shizuoka between Atami and Hamamatsu, it will be

10 Tokyo-->50 ShinYokohama-->90 Atami-->150 Shizuoka-->70 Hamamatsu-->30 Nagoya

Therefore, Shizuoka' s pointer should be 70. Atami's pointer should be 150.

Pointer for the head
10

Address	Data	Pointer
10	Tokyo	50
30	Nagoya	0
50	Shin Yokohama	90
70	Hamamatsu	30
90	Atami	150
150	Shizuoka	70

Q3

Answer

b. Queue

Description

Queue is called the first-in first-out (FIFO) system. → Answer

Stack is termed the last-in first-out (LIFO) system.

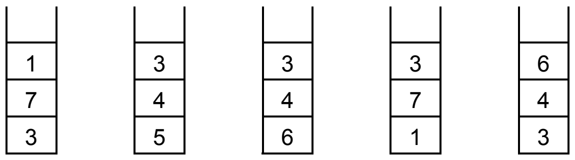
Binary tree is a kind of tree structure in which order of insertion and order of access are independent (neither FIFO or LIFO)

Heap is a kind of binary tree.

Q4

Answer

a b c [✓ d] e

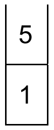


Description

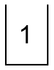
Stack manages data in FILO(First In Last Out) basis.

PUSH 1-->PUSH 5-->POP-->PUSH 7-->PUSH 6-->PUSH 4-->POP-->POP-->PUSH 3

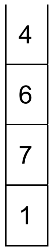
PUSH 1-->PUSH 5



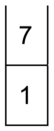
POP



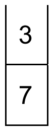
PUSH 7-->PUSH 6-->PUSH 4



POP-->POP



PUSH 3



$\boxed{1}$

Q5

Answer

c. 4

Description

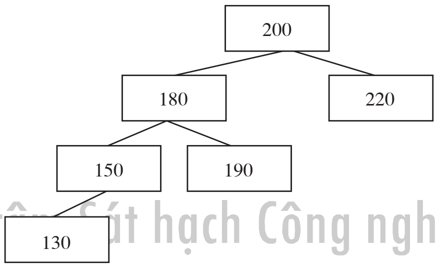


Figure 1 Binary tree

Subscript	Value	Pointer 1	Pointer 2
1	200	3	2
2	220	0	0
3	180	5	$\boxed{a}$
4	190	0	0
5	150	6	0
6	130	0	0

Figure 2
Array representation of a binary tree

In the above Figure 1, for each node represented by the subscript number, the Pointer1 indicates the index value of its left child node, the Pointer2 indicates the index value of its right child node.

"a" means the index value of the 180's right node. What comes there is 190 and its index value is 4.

Q6

Answer

c. 13

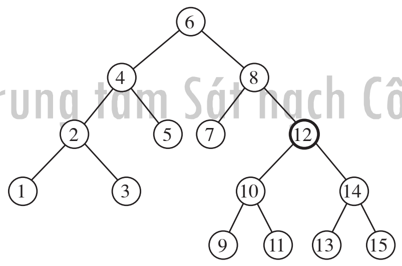
Description

In a binary tree, each node must satisfy the following.

max key value of its left nodes < its key value < min. key value of its right nodes

therefore,

the ex-12 node should have some value greater than 10 and less than 14.



Q7

Answer

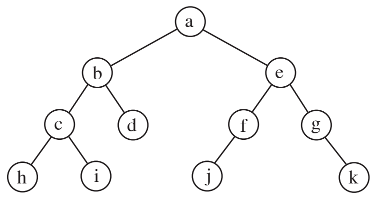
d. hicdbjfkgea

Description

Search is performed using the post-order method, that is,

With a leaf at the bottom left as a starting point, the right side of each node is tracked in a sequential way.

Therefore h,i and c are shown first.

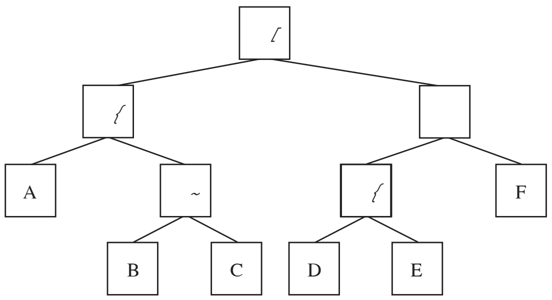


Q8

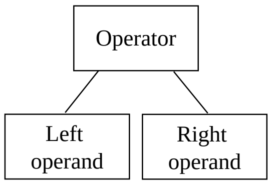
Answer

b. A + B × C - (D + E) ÷ F

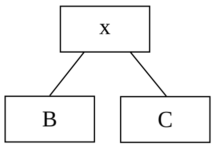
Description



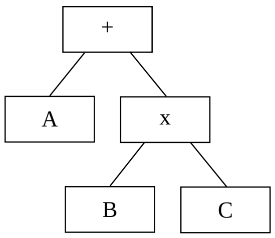
Notation



Therefore,



represents B x C,



represents A + B x C

By interpreting the rest in a similar manner, the answer is

A + B x C - (D + E) / F

Q9

Answer

a. Split and merge nodes to allow the hierarchical depth to become the same.

Description

a is correct.

b refers to hash

(b. Identify the position where data is to be stored by using a certain function and key value.)

c describes sequential access files

(c. Only sequential access to head data and subsequent data is possible.)

d represents partitioned organization files

(d. There are a directory and a member (or members). A member is a sequentially organized file.)

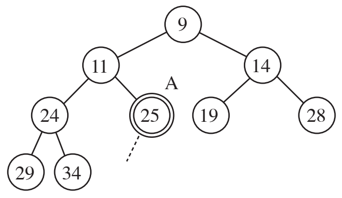
Q10

Answer

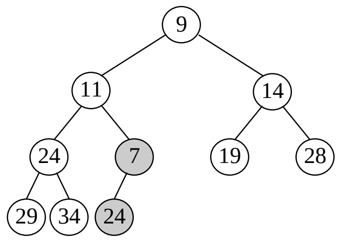
c. 11

Description

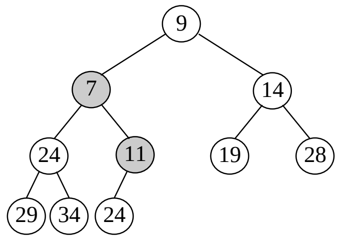
In this question, the element in the position A after 7 is inserted at the "*" position is to be found.



1) Swap 7 and 25 because 7 < 25

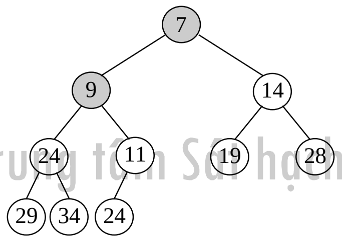


2) Swap 7 and 11 because 7 < 11



3) Swap 7 and 9 because 7 < 9.

This satisfies the requirement and completes swapping.



After the above swapping, the element in the position A is 11. Therefore, the answer is c.

Q11

Answer

b. 2

Description

Hash function

mod(a1＋a2＋a3＋a4＋a5, 13)

mod(54321,13)=mod(5+4+3+2+1,13)=2

Therefore index for 54321 is 2.

Q12

Answer

c. 0.7

Description

A hash table with 10 elements

Therefore the probability that synonym does <u>not</u> occur for 5 values is

9/10 x 8/10 x 7/10 x 6/10 = 3024/10000

The probability that collision occur is

1-0.3024=0.6976

2 Algorithms

Chapter Objectives

The basis of programming is algorithm design. In designing the logical structure of a program, it is important to use the most appropriate algorithm. Also, in choosing a program from a library, what algorithm to use is one of the most important criteria for choosing the most appropriate program.
This chapter describes the basics of algorithm design and representative algorithms.

① Understanding the basics of algorithms, i.e., algorithm definitions, design, relationships with data structures, representation methods, etc.
② Understanding the characteristic and meanings of representative search, sort, character string processing, and file processing algorithms.

Introduction

Using an efficient, easy-to-understand algorithm enables one to increase the execution speed and decrease the number of hidden bugs. An algorithm is one of the critical factors that determine the system performance. Also, the quality of an algorithm is used as criteria for choosing parts from a library.
This chapter describes the basics of algorithms used for module logic design and the algorithms used to solve typical problems. Choosing a well-known or generally used algorithm without fully analyzing given problems should be avoided. An algorithm that is the most appropriate for characteristics of problems must be chosen.
Evaluating algorithms themselves is also an important task. Data obtained by comparing plural algorithms on the basis of objective figures will greatly help you choose the most appropriate algorithm.
In this chapter, you learn the basics of algorithms so that you can design an easy-to-understand module.

2.1 Basics of algorithms

This section explains the following:
- Definition of an algorithm
- Relationships between an algorithm and a data structure

2.1.1 What is an algorithm?

(1) Definition of an algorithm

An algorithm is defined in the Japanese Industrial Standard (JIS) as follows:
A set of a limited number of clearly defined rules that are applied a limited number of times to solve problems.
This means that an algorithm is a set of rules (procedures) established to solve problems. To solve one problem, there are several routes that we can take. As a simple example, when we calculate a value Y times as large as X, we can take two approaches:
- Multiply X by Y
- Adding X Y times

In determining an algorithm, it is important to not only think out a procedure for solving a problem but also design and adopt an algorithm that can solve a problem effectively and efficiently.
Another important point to note is that an algorithm must stop (it must not run in an infinite loop). This matter will be taken up in section 2.3.2 Valuation by correctness.

(2) Algorithm and programming

An algorithm and programming are two sides of the same coin. Programming is describing data and algorithms in a programming language so that a computer can execute its given tasks.
In general, a program consists of algorithms and data and an algorithm consists of logic and control.
Programming can be classified into four different types according to how algorithms are viewed:
- Procedural programming
- Functional programming
- Logic programming
- Object-oriented programming

The most appropriate type of programming must be chosen with consideration of the characteristics of a problem.

① Procedural programming

Procedural programming is the most commonly used type of programming. This category includes the following programming languages, FORTRAN, COBOL, PL/I, Pascal, C, etc.
Characteristic
- A program is divided into modules to make a large, complicated program easy to understand. Coding is performed for each module.
- Structured theorems are introduced to programming (to be explained later).
- Structured figures are used to compile a program.

Because procedural programming is procedure (control)-oriented, there are the following restrictions:
- In addition to a procedure (control), variables (variable names, types, sizes, etc.) must be declared.
- Instructions are executed one by one in a sequential manner (parallel processing cannot be done).
- Comparisons and calculations must be made to solve a problem.

② Functional programming

Functional programming is function-oriented programming. It is used in the field of artificial intelligence (AI), basic calculation theories and other research tasks. LISP, among others, is a functional programming language.
Characteristic
- Unlike sequential-execution-type programming, expressions are built by nesting and they are replaced with results of calculations to execute a program.
- Recursive calls can be described easily.
- The level of affinity with parallel processing is high.
- A calculation process or procedure does not need to be considered.

③ Logic programming

A predicate based on facts and inferences is the base of logic programming. Prolog is an example of a logic programming language.
Characteristic
- Because facts are described based on the predicate logic, the process of programming is made easy.
- Inferences and logical calculations can be made easily.

④ Object-oriented programming

In object-oriented programming, a system is considered a group of objects. Languages include Smalltalk, C++, Java and others.

2.1.2 Algorithm and the data structure

An algorithm and the data structure are closely related to one another. The data structure determines the framework of an algorithm to a certain extent.

(1) Data structure

The data structure is defined as follows:
A procedure that a program follows to store data and perform given tasks.

① Basic data structure

Storing data means storing data in a main memory unit. In storing data in a main memory unit, the data type (data type, size, etc.) must be declared. The most basic data structure unit used to declare the data type is called a basic data structure. In performing this step of data-type declaration, data is manipulated using the name of data whose data type has been declared. (See Figure 2-1-1.)

② Problem-oriented data structure

The problem-oriented data structure built by combining basic data structures contains one or more of the following:
- List

- Stack
- Queue
- Tree structure.

These elements are determined by programming (designed algorithms). If data is processed in the order it is input, a queue is used. If data is processed in the reverse order of input, a stack is used.
Therefore, the contents of the problem-oriented data structure are determined by algorithms to a certain extent.

Figure 2-1-1 Data definition and procedure division in a COBOL program

Data definition

```
DATA                              DIVISION.
FILE                                SECTION.
FD @TOUGETU-FILE.
01 @ T-REC.
 @ @ 02 @T-HIZUKE @ @  @PIC @X(06).
 @ @ @ @ 88 @T-ENDF @ @ @VALUE @HIGH-VALUE.
 @ @ 02 @FILLER @ @ @  @ PIC @X(34).
FD @RUISEKI-FILE.
01 @ R-REC.
 @ @ 02 @R-HIZUKE @  @ @PIC @X(06).
 @ @ @ @ 88 @R-ENDF @ @ @VALUE @HIGH-VALUE.
 @ @ 02 @FILLER @ @ @  @ PIC @X(34).
FD @N-RUISEKI-FILE.
01 @N-REC @ @ @  @ @  @  PIC @X(40).
WORKING-STORAGE @SECTION.
01 @T-COUNT @ @  @ @ @ PIC @9(05).
01 @R-COUNT @ @  @  @ @PIC @9(05).
01 @N-COUNT @ @  @  @ @PIC @9(05).
```

Procedure division

```
PROCEDURE            DIVISION.
HEIGOU-SYORI.
 @ @ OPEN @INPUT @     TOUGETU-FILE @RUISEKI-FILE
 @ @ @ @ @ @OUTPUT @N-RUISEKI-FILE.
 @ @ INITIALIZE @ @ @ @T-COUNT @R-COUNT @N-COUNT.
 @ @ PERFORM @T-YOMIKOMI.
 @ @ PERFORM @R-YOMIKOMI.
 @ @ PERFORM @UNTIL @T-ENDF @AND @R-ENDF
 @ @ @  IF @T-HIZUKE @   @R-HIZUKE
 @ @ @   @THEN @ @WRITE @N-REC @FROM @T-REC
 @ @ @   @ @ @ @ @  PERFORM @T-YOMIKOMI
 @ @ @   @ELSE @ @WRITE @N-REC @FROM @R-REC
 @ @ @   @ @ @ @ @  PERFORM @R-YOMIKOMI
 @ @ @  END-IF
 @ @ @  COMPUTE @ @N-COUNT @   @N-COUNT @ { @1
  @ @END-PERFORM.
 @ @ DISPLAY @T-COUNT @R-COUNT @N-COUNT.
 @ @ CLOSE @TOUGETU-FILE @RUISEKI-FILE @N-RUISEKI-FILE.
 @ @ STOP @RUN.
T-YOMIKOMI.
 @ @ READ @TOUGETU-FILE
 @ @ @  AT @END @ @ @ @MOVE @HIGH-VALUE @TO @T-HIZUKE
 @ @ @  NOT @AT @END @COMPUTE @T-COUNT @   @T-COUNT @ { @1
 @ @ END-READ.
R-YOMIKOMI.
 @ @ READ @RUISEKI-FILE
 @ @ @  AT @END @ @ @ @MOVE @HIGH-VALUE @TO @R-HIZUKE
 @ @ @  NOT @AT @END @COMPUTE @R-COUNT @   @R-COUNT @ { @1
 @ @ END-READ.
```

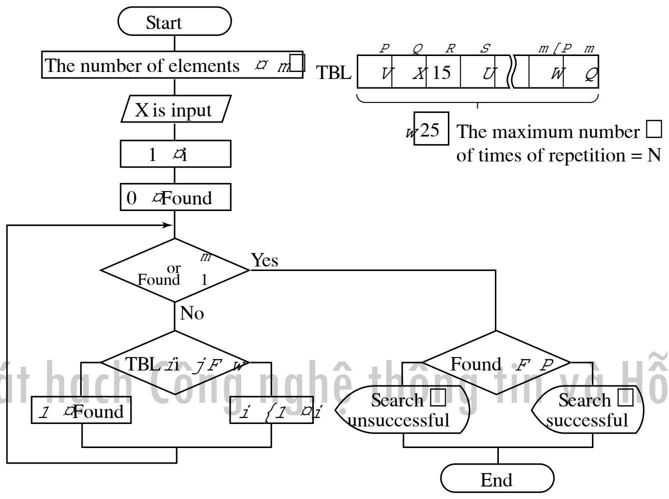
(2) Relationships between an algorithm and the data structure

The relationships between an algorithm and the data structure can be described as follows:

① Array processing

Take a linear search algorithm (details explained in Section 2.2.1) shown in Figure 2-1-2 for example.

Figure 2-1-2 The linear search algorithm and the data structure

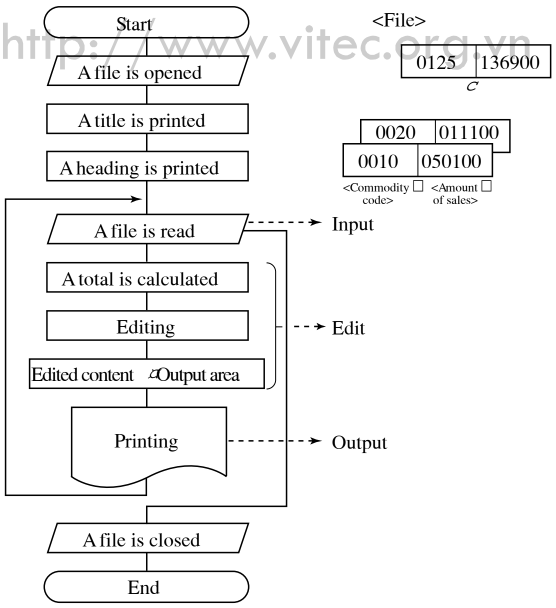


The linear search is most frequently used to search data. In performing the linear search on data, data is searched while subscripts in an array are being incremented. The maximum number of times the search is repeated is the size of an array. That is, if an array data structure is used, the procedure and the number of times of repetition are determined.

② File processing

Take an algorithm for reading and printing a file shown in Figure 2-1-3 for example. (Details are explained in section 2.2.5.)

Figure 2-1-3 Algorithm for processing a file and the data structure



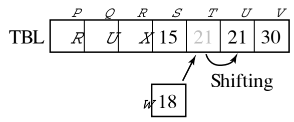
The process of reading, editing and printing a file is repeated until there is no data in a file.

Because a file must be accessed in each process cycle, this task is executed in the form of a loop.

③ List processing

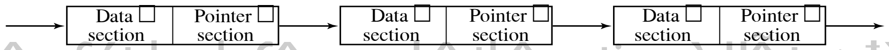
In processing an array structure, data can be searched and updated smoothly but it takes time to insert or delete data. Because data is arranged in queues, inserting or deleting data is inevitably accompanied by the shifting of related data backward or forward.

Figure 2-1-4 Inserting data into an array



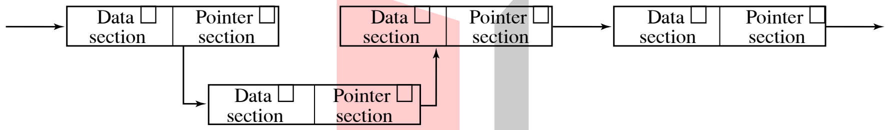
Using the list structure, inserting or deleting data is easy. Although data is arranged in an orderly manner in the list structure, it is arranged so in terms of logic since it does not need to be physically arranged in sequential order.

Figure 2-1-5 List structure



In the case of the list structure, data does not need to be shifted as shown in Figure 2-1-6; data can be inserted or deleted by simply manipulating a pointer.

Figure 2-1-6 Inserting data into the list structure



2.2 Various algorithms

An algorithm should be thought out to solve each specific problem. If an algorithm that can solve similar problems is already designed and made available, using such an algorithm will enable you to create a better algorithm in an efficient manner.
This section describes the representative algorithms that have until now been developed in relation to each type of problem.

2.2.1 Search algorithm

A table search is the method of performing a search on tables and files stored in a memory unit to find the elements that meet specified requirements.
This section describes two table search methods: linear (or sequential) search and binary search methods.

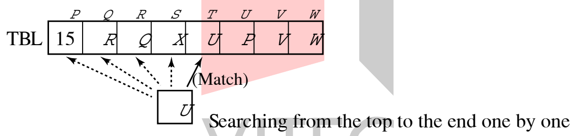
(1) Linear (or sequential) search method

A linear (or sequential) search method is a very simple search method of searching from the top of a table in a sequential manner.

① Exhaustive search method

An exhaustive search method is the simplest method of combing a table from the top to the end.

Figure 2-2-1 Image of the exhaustive search method

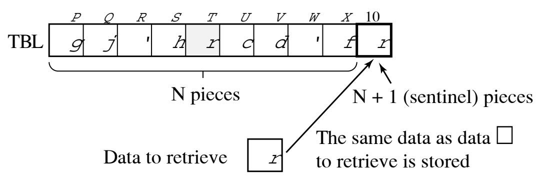


Data retrieved is collated with each data in a table. The algorithm is designed such that the search is successful if there is data that matches data to retrieve and it is unsuccessful if there is no data that matched data to retrieve.
The search procedure ends when the first successful match occurs. Therefore, this search method is inappropriate for counting the number of data that matches data to retrieve.

② Sentinel search method

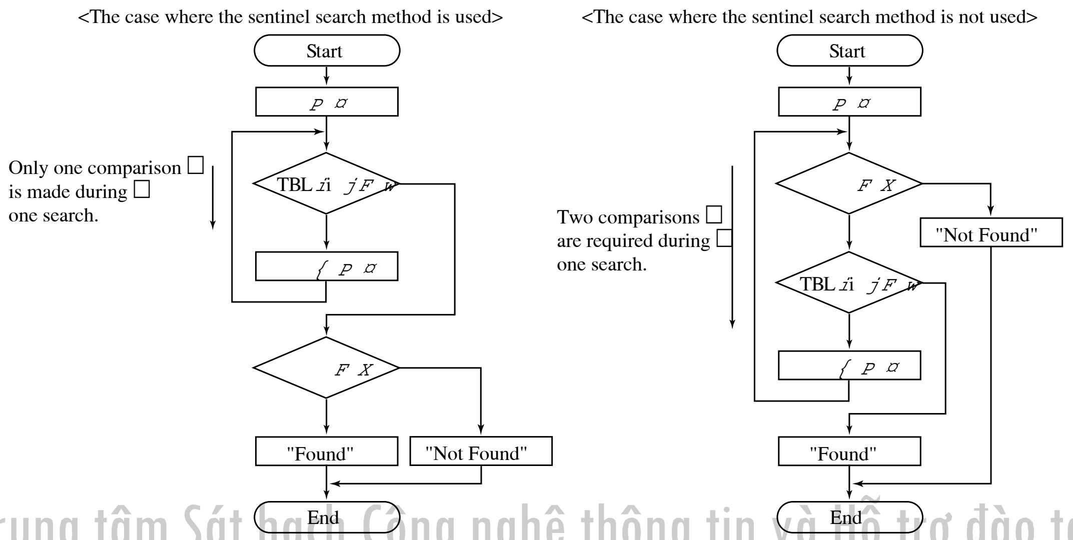
The sentinel search method uses the algorithm in which the same data (sentinel) as the data to retrieve is placed at the end of a table to simplify the search algorithm and to decrease the number of times data is compared.

Figure 2-2-2 Sentinel search method



If the number of data contained in a table is N, the same data (sentinel) as the data to retrieve is stored in the (N + 1) position so that the data to retrieve can be instantly collated with the sentinel data.
Figure 2-2-3 shows a comparison of the case where the sentinel search method is used and the case where it is not used.

Figure 2-2-3 A comparison of the case where the sentinel search method is used and the case where it is not used



<The case where the sentinel search method is used>
Start
TBL
"Found"
"Not Found"
End
Only one comparison is made during one search.
<The case where the sentinel search method is not used>
Two comparisons are required during one search.
"Not Found"
"Found"

<The case where the sentinel search method is used>

In the first round of data collation, data to retrieve is authenticated. In the second round, it is determined whether there is data that matches data to retrieve. That is, if the number of data is N, a comparison is made only (N + 1) times.

<The case where the sentinel search method is not used>

In one round of data collation, the authenticity of data to retrieve as well as whether data to retrieve ends must be determined. That is, a comparison is made (N × 2) times.

<Maximum number of times a comparison is made if the number of elements is N:>

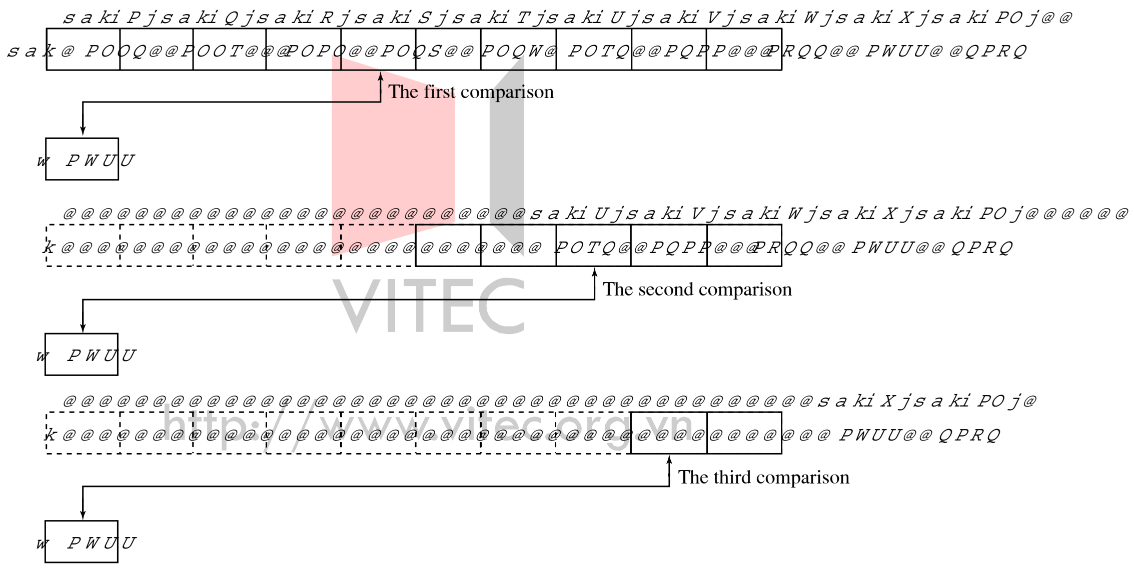
- (N + 1) times if the sentinel search method is used
- (N × 2) times if the exhaustive search method is used

(2) Binary search method

A binary search method is the method of narrowing down target data while successively dividing a search area into two parts. The number of comparisons can be greatly decreased compared with the linear search method and the search efficiency can equally improve. This search method, however, requires that elements are arranged in ascending or descending order.
Figure 2-2-4 shows the algorithm used for the binary search method.

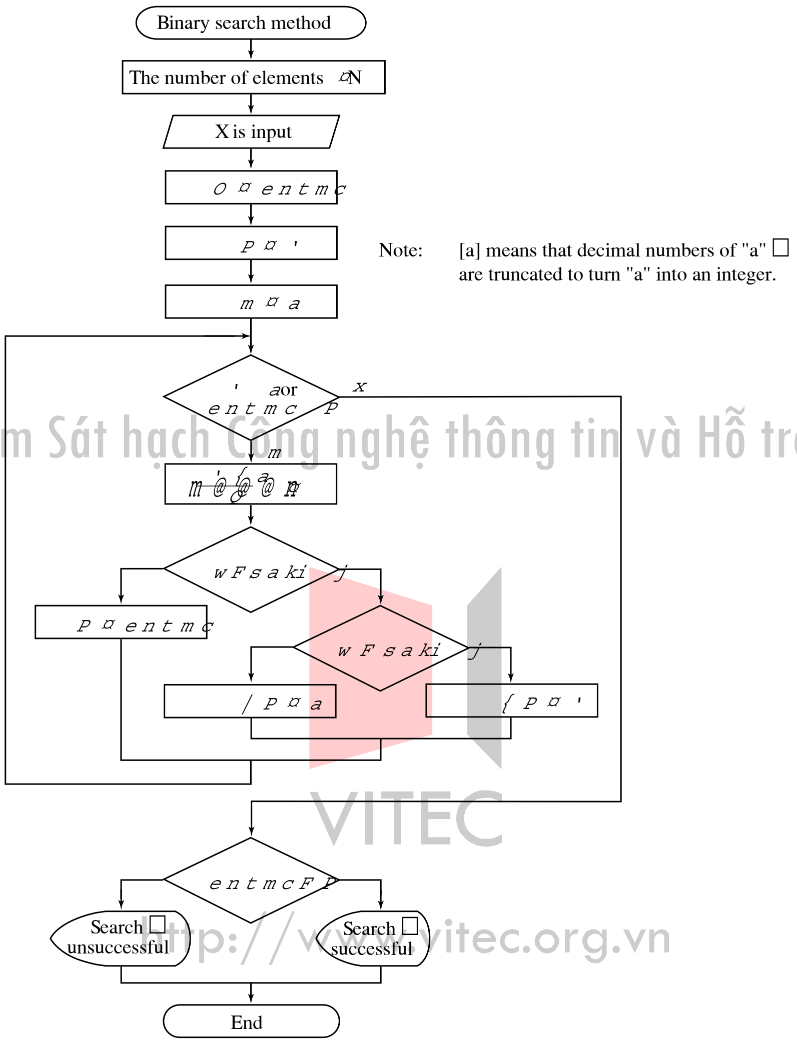
Figure 2-2-4 Algorithm for the binary search method

Step 1:	A total of the value of a subscript representing the top of a table and that of a □ subscript representing the end of a table is divided by 2.
Step 2:	The elements having the value obtained in step 1 as a subscript are compared □ with a target element.
Step 3:	If there is an element that matches a target element, the search is successful.
Step 4:	If the value of a target element is smaller than that of an element in a table, 1 □ is subtracted from the current subscript and the value is used as a subscript □ for representing the end of a table. If the value of a target element is larger than that of an element in a table, 1 is □ added to the current subscript and the value is used as a subscript for □ representing the top of a table.
Step 5:	Step 1 through step 4 is repeated. If an element matching a target element □ cannot be found at the point where the value of a subscript representing the □ top of a table becomes larger than that of a subscript representing the end of a □ table, the search is unsuccessful. This completes the search.



Because elements are arranged in ascending or descending order, data smaller than reference data does not need to be searched if data being searched is larger than reference data. Therefore, the number of data to search can be halved after the first search--a great advantage in the search efficiency.
Figure 2-2-5 shows a flowchart of the binary search method.

Figure 2-2-5 Flowchart of the binary search method


Binary search method
The number of elements →N
X is input
Note: [a] means that decimal numbers of "a" are truncated to turn "a" into an integer.
Search unsuccessful
Search successful
End

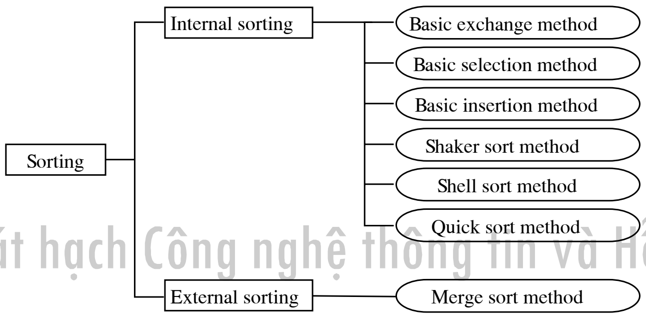
<Maximum and average number of times a comparison is made: If the number of elements is N>

- Average number of times = [$\log_2$N]
- Maximum number of times-[$\log_2$N] +1
([ ] is a Gaussian symbol and decimal numbers of the value shown in this symbol are truncated.)

2.2.2 Sort algorithm

Sorting data is reorganizing data in a specified order. Sorting data in the order of small to large values is called ascending order sorting, and sorting data vice versa is called descending order sorting.
If data is sorted or organized in a certain specified order (amounts of sales, commodity codes, etc.), the efficiency of the work of data processing can be increased. Therefore, the sort algorithm is one of the most commonly used algorithms. Using this sort algorithm, one can sort numbers and characters. This type of sorting is possible because characters are represented as character codes (internal codes) in a computer.
Figure 2-2-6 shows various sort methods.

Figure 2-2-6 Sort methods



Internal sorting means sorting data in a main memory unit. External sorting means sorting data stored on a magnetic disk and other auxiliary storage unit.
In choosing one sort method, the speed of sorting, as well as how data is sorted, must be considered to minimize the time needed for data comparison and exchange.
In choosing one sort method, you must also clarify the time of calculation (computational complexity). This point will be described in detail in Section 2.3.

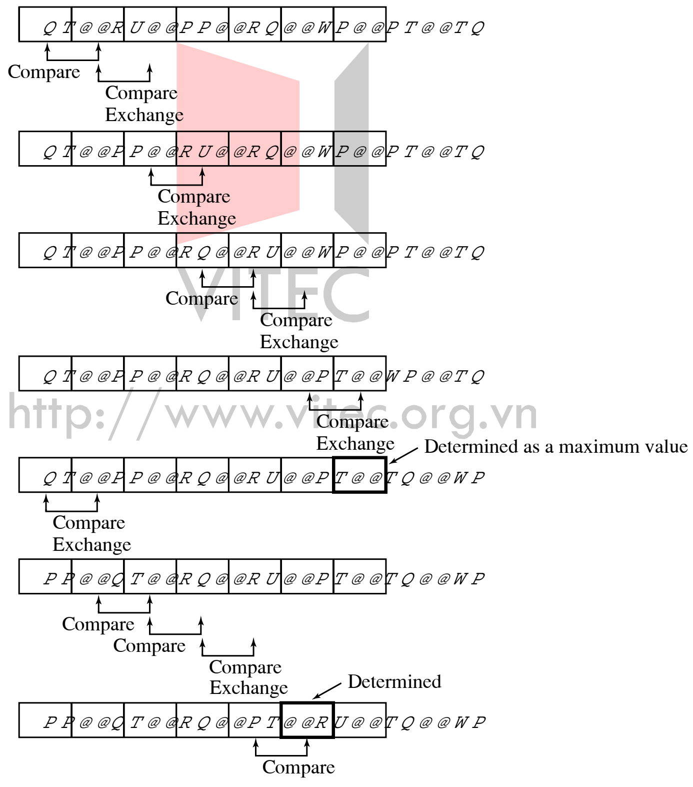
(1) Basic exchange method (bubble sort)

The basic exchange method (bubble sort) is used to compare a pair of data sequentially from the head of an array. If the sequence is wrong, data is exchanged. When all data items in one array are compared, the routine returns to the head of an array and it is repeated until there are no remaining data items to exchange. This method is the simplest, best known sort method. The name "bubble sort" was given since the movement of maximum or minimum data looks like bubbles popping out on the surface of the water.
Figure 2-2-7 shows the algorithm of the basic exchange method.

Figure 2-2-7 Steps of the basic exchange method

Algorithm for sorting data in the ascending order
- Step 1: The first and second elements in a table are compared.
- Step 2: If the first element is larger than the second, the first is exchanged with the second.
- Step 3: If the second element is larger than the first, no exchange occurs.
- Step 4: The second and third elements are compared and steps 2 and 3 are repeated.
- Step 5: This routine operation is repeated to the last element in a table. As it reaches the last element, a maximum value is stored in the last element in that table.
- Step 6: Steps 1 through 4 and step 5 are executed until the operation reaches the last element but one.
- Step 7: Steps 1 through 6 are repeated until only the first and second elements in a table remain.

This completes data sorting.



All these steps are repeated as one cycle.

<Characteristics>

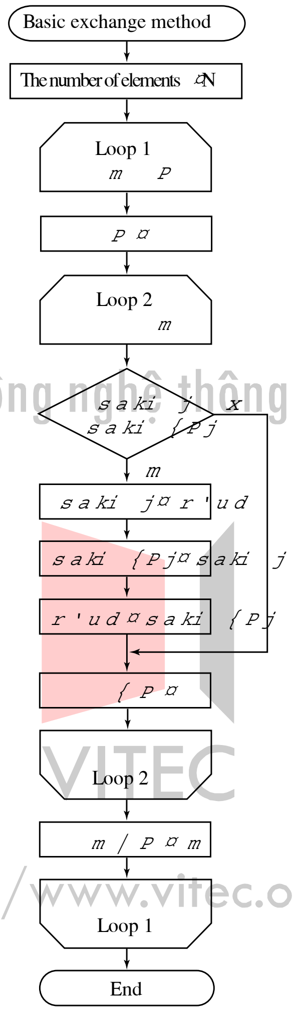
- One of the simplest sort methods
- The efficiency is low since data item are unconditionally compared even if they are sorted correctly.
- If the volume of data is large, the process is time-consuming.

<Computational complexity>

- Maximum computational complexity: $O\ (n^2)$
- Average computational complexity: $O\ (n^2)$

Figure 2-2-8 shows the flowchart of the basic exchange method.

Figure 2-2-8 Flowchart of the basic exchange method

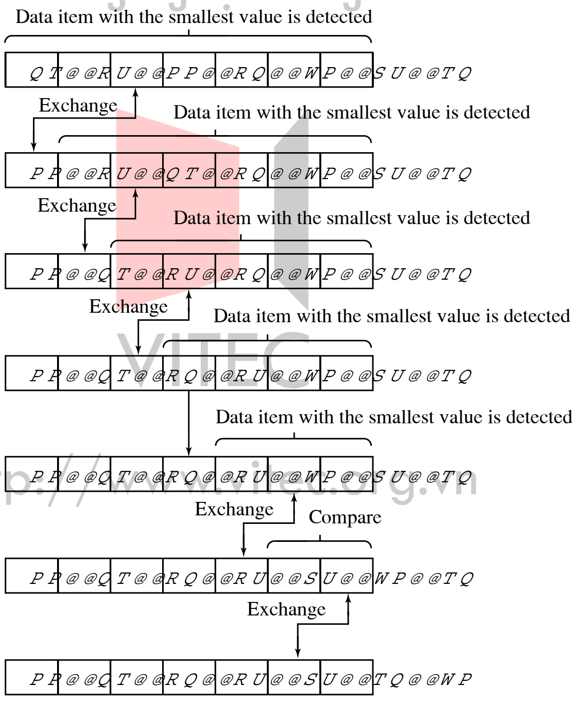


(2) Basic selection method

In the sort algorithm of the basic selection method, a data item with the smallest (or the largest) value is first selected from all the data items and it is exchanged with the data item at the head of an array, then the same routine is repeatedly executed on all the remaining data items. When the correct data item enters the last position but one, data sorting is completed.
Because this method allows a remaining element with the smallest or largest value to be selected, it is called the basic selection method.
Figure 2-2-9 shows the algorithm of the basic selection method.

Figure 2-2-9 Steps of the basic selection method

Steps of the algorithm for sorting data in the ascending order
- Step 1: Data item with the smallest value is detected in data items stored in a table.
- Step 2: Data item with the smallest value detected is exchanged with the first data □ item in a table (space into which the data item with the smallest value can be temporarily saved is required).
- Step 3: Data item with the smallest value is detected in the second to the last data item in a table.
- Step 4: Data item with the smallest value detected is exchanged with the second data item in a table.
- Step 5: The same operation is repeatedly performed until the last data item but one is reached. When the last data item but one is reached, data sorting is completed.


Data item with the smallest value is detected
Q T @ @ R U @ @ P P @ @ R Q @ @ W P @ @ S U @ @ T Q
Exchange
Data item with the smallest value is detected
P P @ @ R U @ @ Q T @ @ R Q @ @ W P @ @ S U @ @ T Q
Exchange
Data item with the smallest value is detected
P P @ @ Q T @ @ R U @ @ R Q @ @ W P @ @ S U @ @ T Q
Exchange
Data item with the smallest value is detected
P P @ @ Q T @ @ R Q @ @ R U @ @ W P @ @ S U @ @ T Q
Data item with the smallest value is detected
P P @ @ Q T @ @ R Q @ @ R U @ @ W P @ @ S U @ @ T Q
Exchange
Compare
P P @ @ Q T @ @ R Q @ @ R U @ @ S U @ @ W P @ @ T Q
Exchange
P P @ @ Q T @ @ R Q @ @ R U @ @ S U @ @ T Q @ @ W P

<Characteristics>

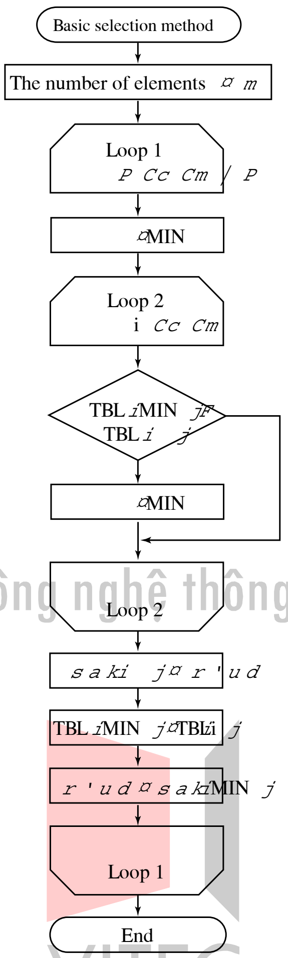
- One of the simplest sort methods, like the basic exchange method
- The efficiency is low since data items are unconditionally compared even if they are sorted correctly.
- If the volume of data is large, the process is time-consuming.

<Computational complexity>

- Maximum computational complexity: $O(n^2)$
- Average computational complexity: $O(n^2)$

Figure 2-2-10 shows the flowchart of the basic selection method.

Figure 2-2-10 Flowchart of the basic selection method



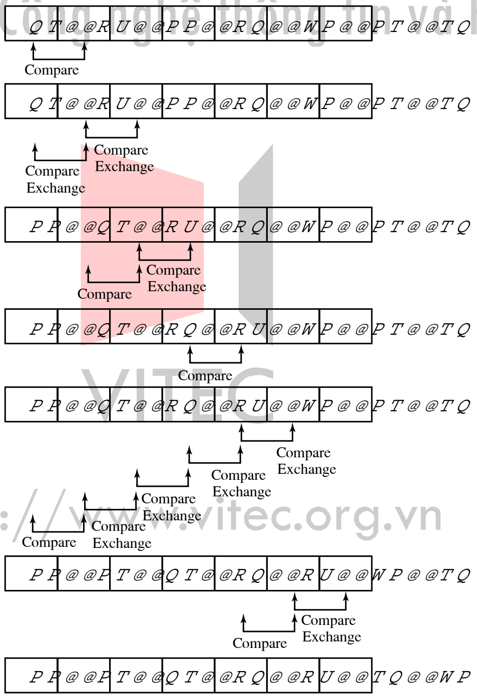
(3) Basic insertion method

In the algorithm of the basic insertion method, while data items are being sorted, an unsorted data item is inserted into a proper position in the sequence of sorted data items.
Figure 2-2-11 shows the algorithm of the basic insertion method.

Figure 2-2-11 Steps of the basic insertion method

Algorithm for sorting data in the ascending order
- Step 1: The first and second elements in a table are compared.
- Step 2: If the first element is smaller than the second, nothing is done.
- Step 3: If the second element is smaller than the first, the first element is □ exchanged with the second. At this point, the first and second □ elements are in the correct order.
- Step 4: The second and third elements are compared.
- Step 5: If the second element is smaller than the third, nothing is done.
- Step 6: If the third element is smaller than the second, the second element is □ exchanged with the third. Then this element is compared with the □ preceding element according to steps 2 and 3. These steps are repeated□ till it is placed in the right position.
- Step 7: Steps 4, 5 and 6 are repeated until the last element in the table is inserted □ into the correct position.

This completes data sorting.

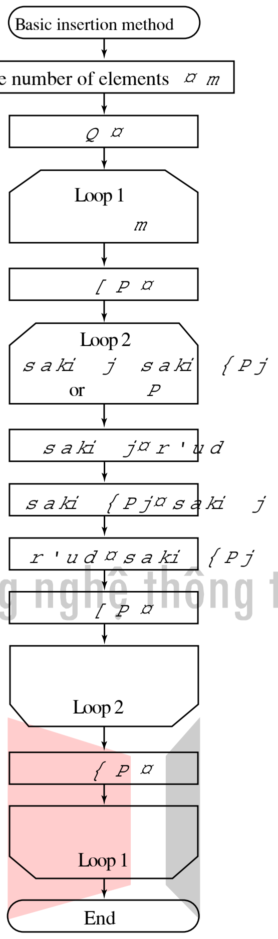

Q T @ @R U @ @P P @ @R Q @ @W P @ @P T @ @T Q
Compare
Q T @ @R U @ @P P @ @R Q @ @W P @ @P T @ @T Q
Compare
Exchange
Compare
Exchange
P P @ @Q T @ @R U @ @R Q @ @W P @ @P T @ @T Q
Compare
Exchange
Compare
P P @ @Q T @ @R Q @ @R U @ @W P @ @P T @ @T Q
Compare
P P @ @Q T @ @R Q @ @R U @ @W P @ @P T @ @T Q
Compare
Exchange
Compare
Exchange
Compare
Exchange
Compare
Exchange
Compare
P P @ @P T @ @Q T @ @R Q @ @R U @ @W P @ @T Q
Compare
Exchange
Compare
P P @ @P T @ @Q T @ @R Q @ @R U @ @T Q @ @W P

<Characteristics>

- One of the simplest sort methods, like the basic exchange method
- Because preceding data items have been sorted, comparison and insertion speeds are fast.
- If the volume of data is large, the process is time-consuming.

Figure 2-2-12 shows the flowchart of the basic insertion method.

Figure 2-2-12 Flowchart of the basic insertion method



(4) Shaker sort method

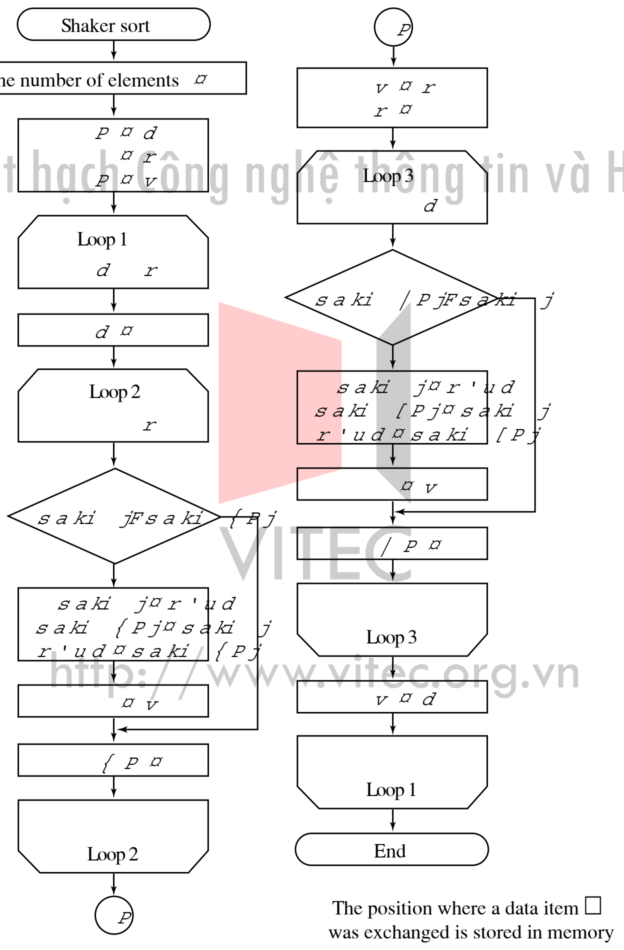
The algorithm of the Shaker sort method is basically the same as that of the basic exchange method (bubble sort). In the algorithm of the basic exchange method, data items are compared from left to right and are sorted in such a way that a maximum (minimum) value is set at the rightmost position. In the algorithm of the Shaker sort method, however, data items are first compared from left to right, then, after a maximum (minimum) value is set at the rightmost position, data items are compared from right to left and a minimum (maximum) value is set at the leftmost position; this operation is repeatedly performed to sort data.

<Characteristic>

- If the volume of data is large, the process is time-consuming.

Figure 2-2-13 shows the flowchart of the Shaker sort method.

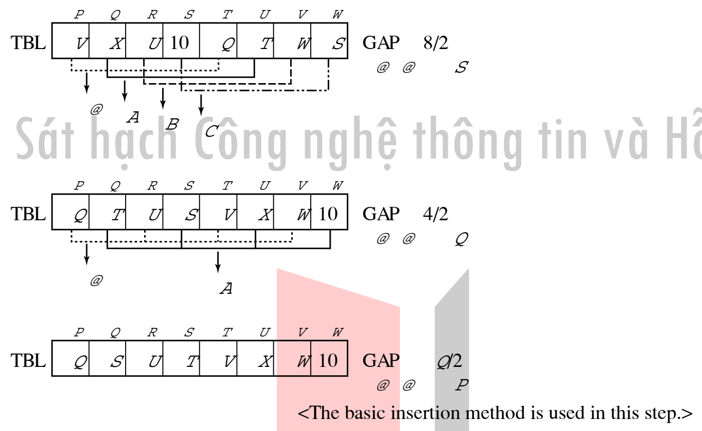
Figure 2-2-13 Flowchart of the Shaker sort method



(5) Shell sort method

The Shell sort method is an extended version of the basic insert method. Two items of data located away from each other at a certain interval are picked out of a data sequence and sorting is executed while the picked data items are compared with each other. The interval is called a gap. This gap is set large at the start of sorting; as sorting proceeds, it is gradually made smaller and finally set to 1. There are various ways of determining this gap. If the number of data is n, a simple way of determining the gap is [n/2], [n/4],··· 1. When the gap is finally set to 1, sorting is executed in exactly the same way as the basic insertion method.
Using the basic insertion method, adjacent elements are compared and exchanged and, therefore, the execution speed is slow. Using the Shell sort method, pieces of data that are away from each other and located in different positions are quickly exchanged so that data items sorted in wrong positions can be resorted to correct positions in the earliest stages of the sorting operation. As sorting proceeds, the gap between data items to be compared is narrowed.
Figure 2-2-14 shows the algorithm of the Shell sort method.

Figure 2-2-14 Steps of the Shell sort method

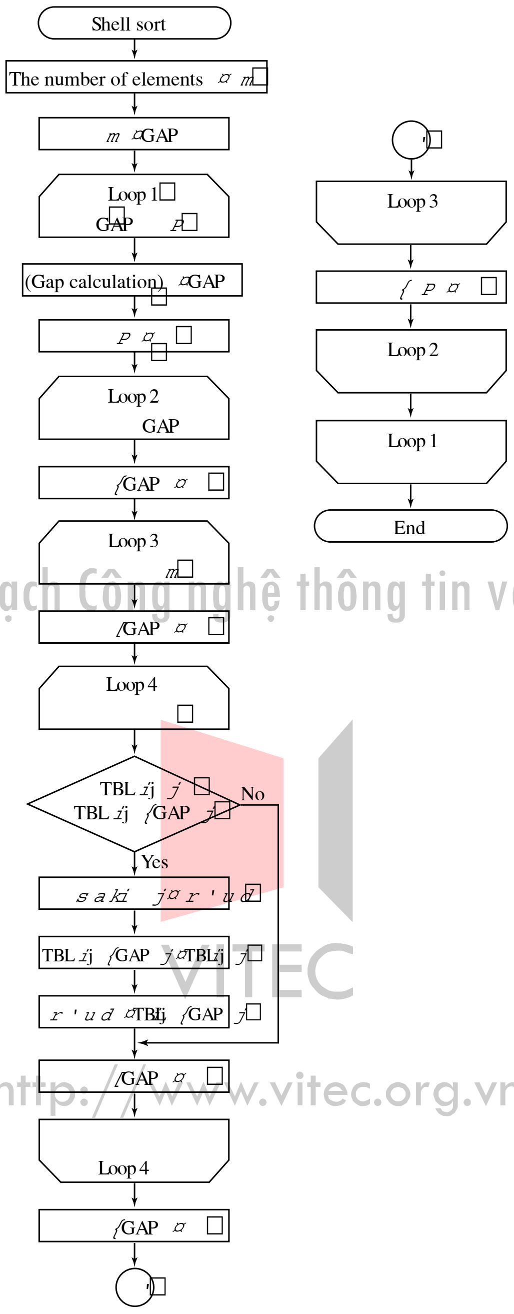


<Characteristics>

- If part of the data is already sorted, sorting can be completed very quickly.
- A high-speed sorting method that uses the basic insertion method

Figure 2-2-15 shows the flowchart of the Shell sort method.

Figure 2-2-15 Flowchart of the Shell sort method



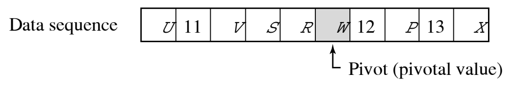
(6) Quick sort method

The quick sort method was designed by Hoare. It is presently the fastest sort method using the recursive call method.

<To sort data in the ascending order>

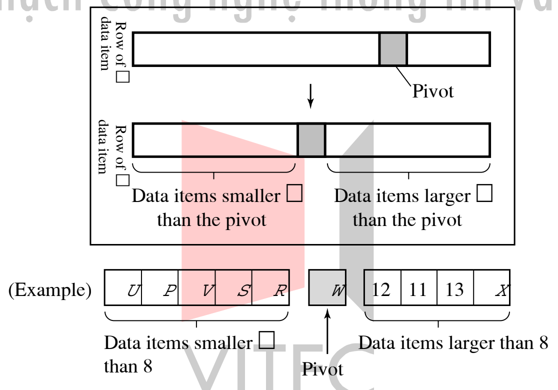
1. A reference value (pivot or pivotal value) is chosen from the data to be sorted. Although various values are used as reference values, a median value or an intermediate value of three elements (right, center and left elements) is usually used. We use a median value in the example of sorting shown below.

Figure 2-2-16 Pivot (pivotal value)



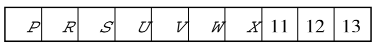
2. Data items smaller than the pivot are moved to the left of the pivot while data items larger than the pivot are moved to the right of it. All the data items are thus divided into two sections (division).

Figure 2-2-17 Moving data



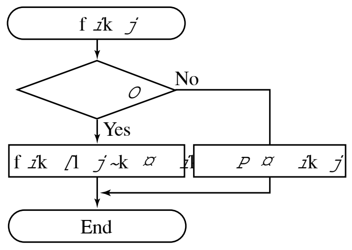
3. A pivot is chosen from each section of data items so that the section is further divided into two sections.
4. This dividing operation is repeatedly performed until only one element remains.

Figure 2-2-18 Data after sorting is completed



A method of dividing a large problem into small problems and solving each small problem individually, like the quick sort method, is called the divide-and-conquer method.
In performing steps 3 and 4 above, the recursive call method of calling the routine itself is generally used. This recursive call method is often used to calculate factorials. (See Figure 2-2-19.)

Figure 2-2-19 Algorithm for calculating factorials



With regard to computational complexity, if an ideal case in which a pivot that can divide data into two sections can always be chosen, the number of comparisons will be very close to $O(n\log n)$. If a maximum (minimum) value in a data sequence is always chosen as a pivot, the number of comparison will become the worst, $O(n^2)$. Although computational complexity usually indicates maximum computational complexity, a case like this may happen in which average computational complexity becomes an important indicator.
After data items are divided into two sections, data items in each section to be subjected to a recursive routine can be processed individually. Therefore, parallel processing can be performed. The average processing time is $O(\log n)$ if parallel processing is executed.
Figure 2-2-20 shows the algorithm for performing quick sorting with a pivot set at the rightmost position of an array.
Specifically, two pointers are used and the routine proceeds from both ends to the center of a data sequence. A pointer moving from the left end to the right is "i" and one moving from the right end to the left is "j."

Figure 2-2-20 Algorithm of the quick sort method

Step 1: As a preparatory step, pointers i and j □ must be initialized.
- i is at the left end, that is 1.
- j is at the last position but a (7) of the □ pivot, that is 6.

Step 2-1: i is moved to the right until an □ element larger than the pivot is □ detected.
Step 2-2: j is moved to the left until an □ element smaller than the pivot is □ detected.
Step 2-3: a (i) is exchanged with a (j).
Step 2-4: Steps 2-1, 2-2 and 2-3 are □ repeated until i j.
Step 2-5: a (i) is exchanged with the pivot at □ the right end.

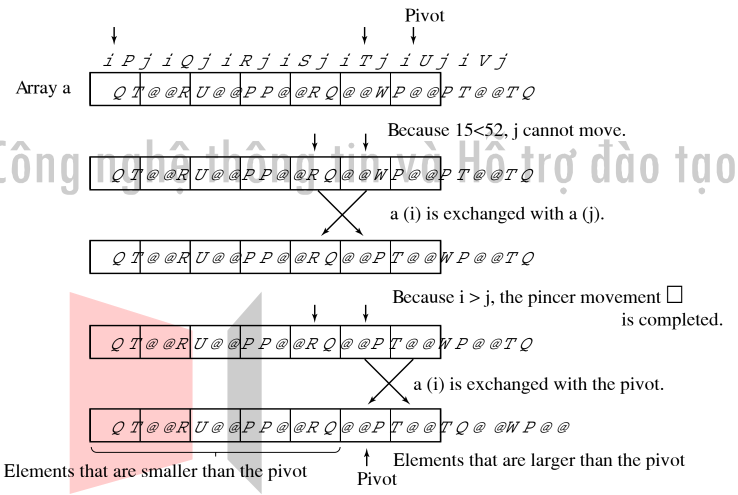
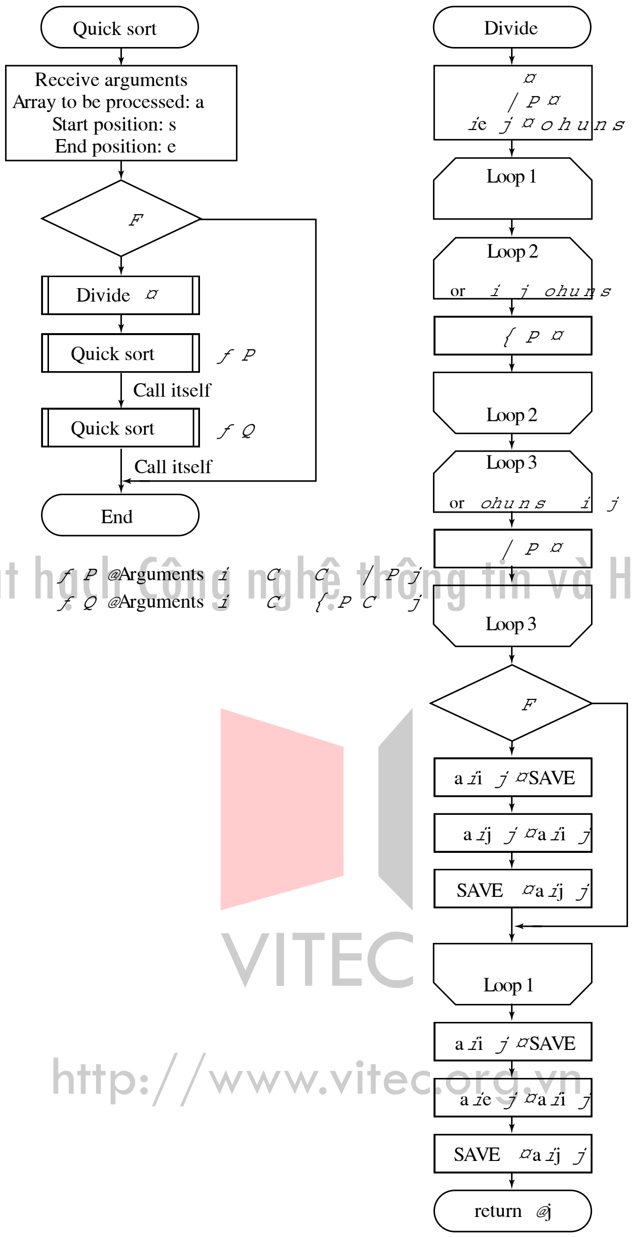

Pivot
Array a
Because 15<52, j cannot move.
a (i) is exchanged with a (j).
Because i > j, the pincer movement □ is completed.
a (i) is exchanged with the pivot.
Elements that are smaller than the pivot
Pivot
Elements that are larger than the pivot

Figure 2-2-21 shows the flowchart of the quick sort method.

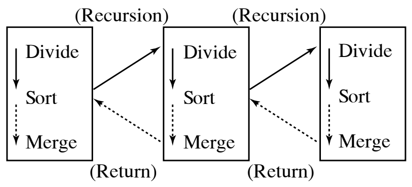
Figure 2-2-21 Algorithm of the quick sort method



(7) Merge sort method

In the algorithm of the merge sort method, all the data items are divided into sections and sorting is executed in each divided section. The sorted sections are merged into a sorted sequence. (see Figure 2-2-22). Merge means comparing data items in two sorted sequences from the head and creating one sequence of sorted data by repeatedly taking out smaller data items in a sequential manner. If the number of data items is 2*n*, repeating the merge by *n* times will complete sorting. If it is not 2*n*, some adjustment is necessary.

Figure 2-2-22 Conceptual diagram of merge sort



<Characteristics>

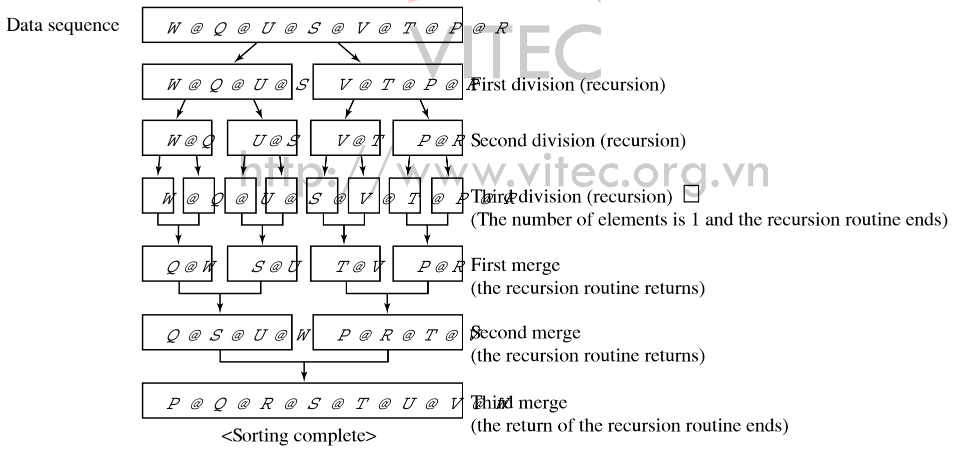
- The recursive call and divide-and-conquer methods are used, as in the case of the quick sort method.
- Because data sequences are sequentially accessed and sorted, the merge sort algorithm is used for external sorting, for example, storing data on magnetic tapes.

<Procedure>

1. Data is divided into two sections and each divided section is further subdivided until there is only one remaining element in a data sequence.
2. After a data sequence is divided, divided sections are sequentially merged.

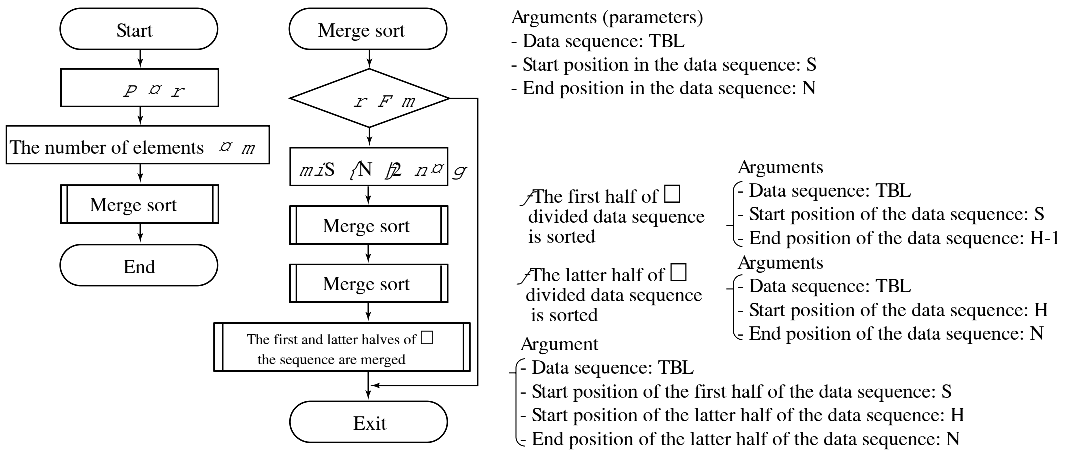
Figures 2-2-23 and 2-2-24 show the status of data sequences during merge sort operations and the flowchart of the merge sort method.

Figure 2-2-23 Status of data sequences during merge sort operations



Data sequence
W Q U S V T P R
W Q U S
V T P R
First division (recursion)
W Q
U S
V T
P R
Second division (recursion)
W Q U S V T P R
Third division (recursion)
(The number of elements is 1 and the recursion routine ends)
Q W
S U
T V
P R
First merge
(the recursion routine returns)
Q S U W
P R T V
Second merge
(the recursion routine returns)
P Q R S T U V W
Third merge
(the return of the recursion routine ends)
<Sorting complete>

Figure 2-2-24 Flowchart of the merge sort method


Start
P ¤ r
The number of elements ¤ m
Merge sort
End
Merge sort
r F m
miS fN f2 n¤ g
Merge sort
Merge sort
The first and latter halves of the sequence are merged
Exit
Arguments (parameters)
- Data sequence: TBL
- Start position in the data sequence: S
- End position in the data sequence: N
The first half of divided data sequence is sorted
Arguments
- Data sequence: TBL
- Start position of the data sequence: S
- End position of the data sequence: H-1
The latter half of divided data sequence is sorted
Arguments
- Data sequence: TBL
- Start position of the data sequence: H
- End position of the data sequence: N
Argument
- Data sequence: TBL
- Start position of the first half of the data sequence: S
- Start position of the latter half of the data sequence: H
- End position of the latter half of the data sequence: N

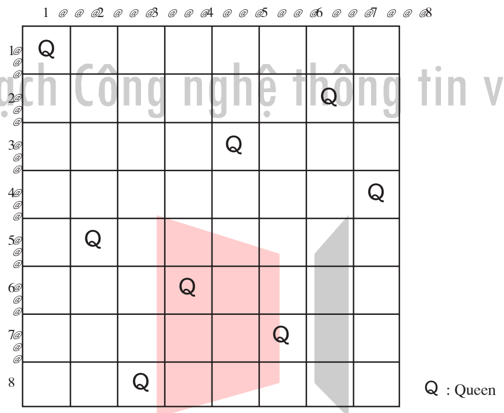
2.2.3 Recursive algorithm

A recursive call is calling a certain routine during data processing. The algorithm designed using a recursive call is the recursive call algorithm. The quick sort and merge sort algorithms are also recursive algorithms.
This section takes up the "eight-queen question" as an example for explaining how the recursive algorithm works.

(1) Eight-queen question

The eight-queen question is a question of how eight queens can be arranged on a chessboard (8 x 8 grid) to prevent pieces from being taken by the opponent. A queen is a chess piece, and it can take a piece that is located on vertical, horizontal or oblique straight lines. In the answer, therefore, queens must be positioned in such a way that they are not located on the same straight lines. Figure 2-2-25 shows one of answers.

Figure 2-2-25 Example of an answer to the eight-queen question

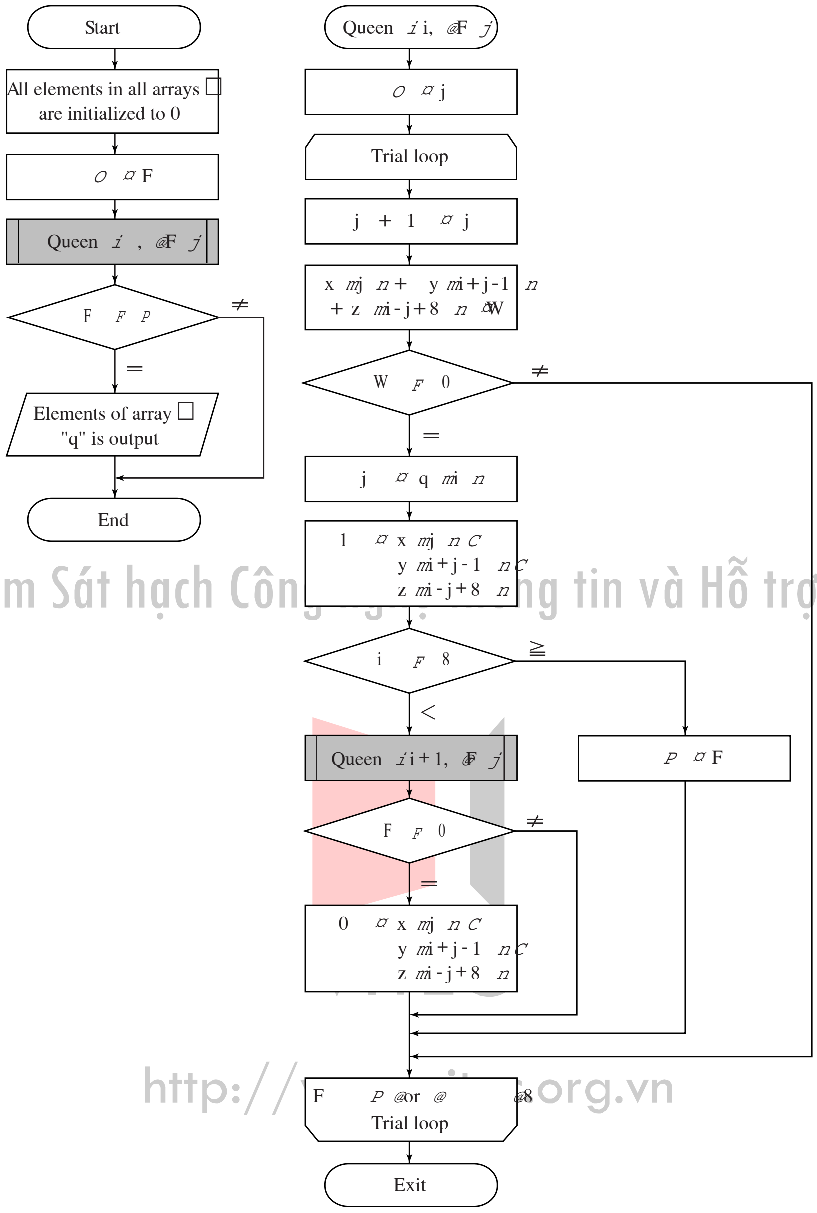


In solving this question, the following four arrangements are used:

- q [i]: The position where a queen is placed in the ith column (i=1 to 8)
  In the above example, q = {1, 5, 8, 6, 3, 7, 2, 4}
- x [j]: To indicate whether or not there is a queen on the jth row (j=1 to 8)
- y [k]: To indicate whether or not there is a queen on the kth left-to-right upward diagonal line. On the same left-to-right upward diagonal line, i + j is always the same. This can be used to obtain a subscript "k" with i+j-1 (k=1 to 15).
- Z [l]: To indicate whether or not there is a queen on the lth left-to-right downward diagonal line. On the left-to-right downward same diagonal line, i-j is always the same. This can be used to obtain a subscript "l" with i-j+8 (l=1 to 15).
- * For arrays x, y and z, subscripts 0 and 1 mean 'there is no queen' and 'there is a queen' respectively.

Figure 2-2-26 shows the flowchart of the algorithm for solving this question.

Figure 2-2-26 Flowchart of the eight-queen question



2.2.4 Character string processing

Characters are searched, inserted, deleted, exchanged or compressed. This section describes character string search and compression.

(1) Character string search

To search character strings, the algorithm for searching a certain character string pattern and locating it in character strings is used.

① Simple collation

Text is compared character by character sequentially from the head. When a character string on which a search should be performed is detected, the position where it is located is set as a return value. In principle, this comparison is repeatedly executed until the first character of a character string to search matches a character in character strings in text. If there is a match, each remaining character of the matched character string is compared with each of a character string to search.

Figure 2-2-27 Image of the search

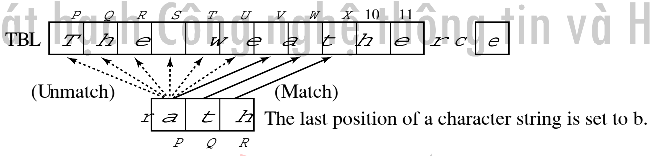
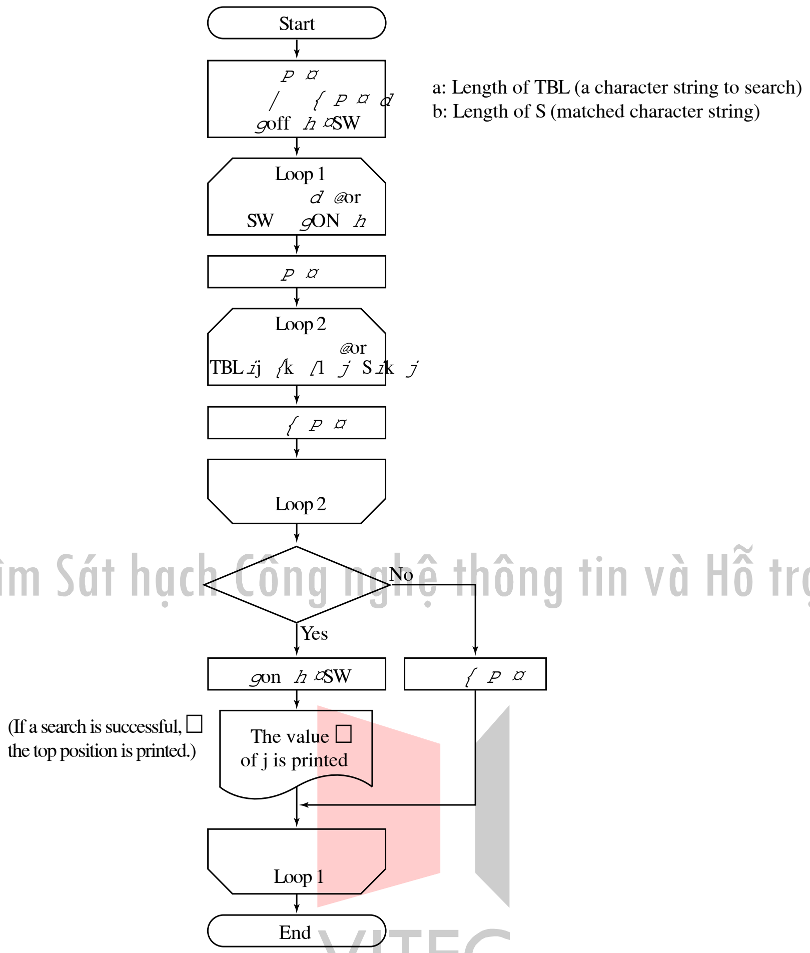


Figure 2-2-28 Flowchart of the simple collation


Start
goff SW
a: Length of TBL (a character string to search)
b: Length of S (matched character string)
Loop 1
@or
SW gON
Loop 2
@or
TBL S
Loop 2
No
Yes
gon SW
(If a search is successful, the top position is printed.)
The value of j is printed
Loop 1
End

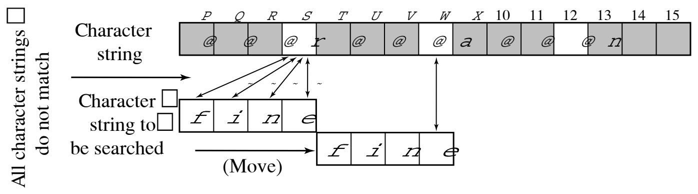
② Boyer-Moore method (BM method)

In the Boyer-Moore algorithm, data is collated while characters in the text are skipped. In this section, only the basic scheme of this algorithm is explained.

a. If there is no character string to search

Figure 2-2-29 shows a comparison of the tail of a character string to search and text patterns.

Figure 2-2-29 BM method (case 1)

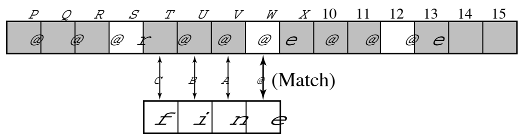

All character strings do not match
Character string
Character string to be searched
(Move)
10
11
12
13
14
15
f i n e
f i n e

In this case, a character at the tail, and all other characters in the first text portion do not match any character of a character string to be searched. Therefore, a search point is moved by the length of a character string to be searched to allow the next search to start from that point.

b. If there is a match with a character at the tail of a character string to be searched

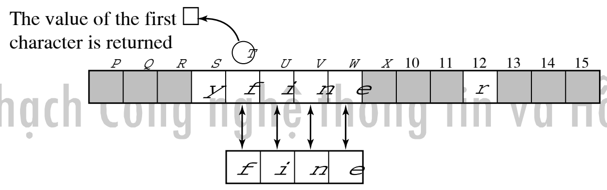
Figure 2-2-30 shows the case in which a character at the tail of a character string to be searched is compared with the text pattern, and a match is found.

Figure 2-2-30 BM method (case 2)



Because a character at the tail of a character string matches a character in the text, characters that precede the matched character must be compared. If all characters match, the value of a subscript of the first character of that matched text pattern is returned.

Figure 2-2-31 Search successful

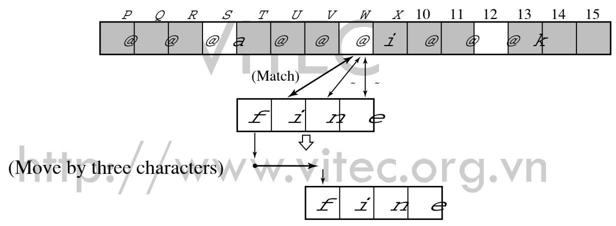


<All characters match>

c. If there is a match with one character somewhere in a character string but unmatch with a character at the tail of a character string

In the case shown in Figure 2-2-32, a character string to be searched can simply be moved. What matters is the distance of movement.

Figure 2-2-32 BM method (case 3)



The distance of movement is determined by how characters in a character string to be searched are arranged as shown in Figure 2-2-33.

Figure 2-2-33 Distance of movement

Character	f	i	n	e	others
Distance of movement	R	Q	P	O	S

By storing this distance of movement in an array, a character string to be searched can be moved to a correct position.

(2) Character string compression

Making a character string short by replacing consecutive characters or blanks with the number of characters is called character string compression.
This section explains the algorithm for compressing blank characters (spaces).
In the case of the example shown in Figure 2-2-34, character strings are searched from the head, and if there are three consecutive blank characters, they are replaced with special characters (#) and the number of blank characters.

Figure 2-2-34 Image of the character string compression

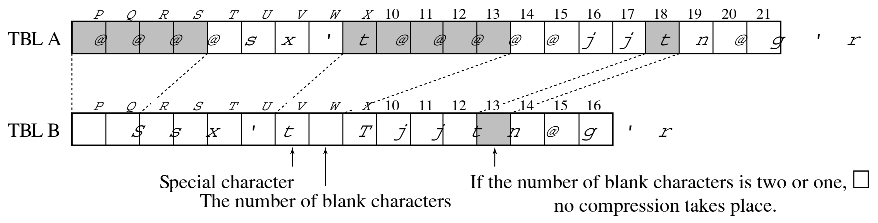
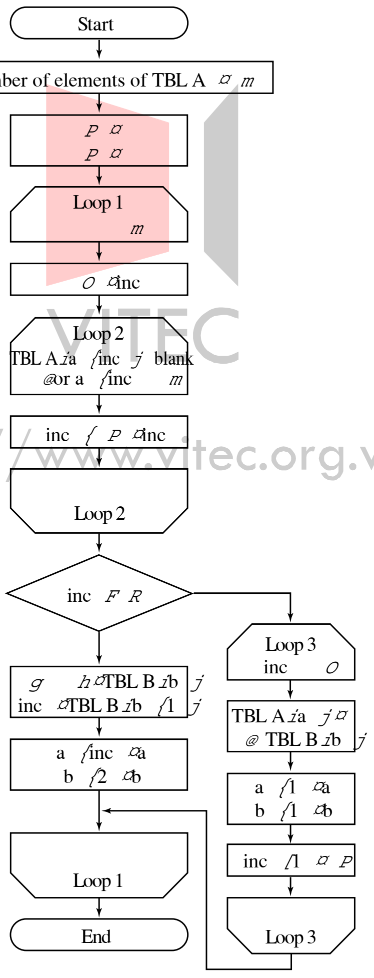

TBL A
TBL B
Special character
The number of blank characters
If the number of blank characters is two or one,
no compression takes place.

Figure 2-2-35 shows the flowchart of the character string compression.

Figure 2-2-35 Flowchart of the character string compression



2.2.5 File processing

Various files are used to perform paperwork tasks. The characteristic of file processing is processing each record one by one.
A typical file processing algorithm is as follows:

Example
Preparatory processing: Opening files, clearing counters, etc.
Main processing: Calculations, editing, outputting, etc.
End processing: Closing files, etc.

This section describes one algorithm for processing files of the same type and another algorithm for processing files of different types.

(1) Group control in processing files of the same type

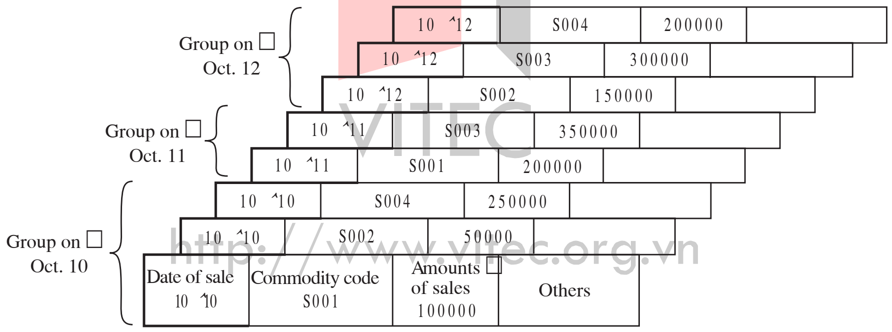
Group control (group total) is to process records with the same key as one lump. In calculating a total of sales for each customer code (key=customer code) or an average mark for each class (key=class code), for example, group control is an indispensable processing routine.
Group control requires that records are sorted according to each key.

Example Algorithm for reading sales files and printing a detailed listing of sales and total amounts of sales on each day

① Sales file layout

Figure 2-2-36 shows the layout of a sales file.

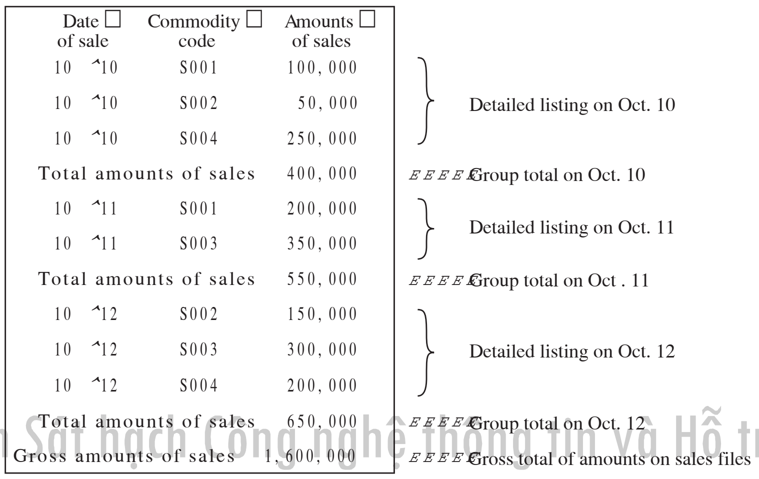
Figure 2-2-36 Layout of a sales file



② Output format (a detailed listing of sales)

Figure 2-2-37 shows the format used to print a detailed listing of sales and total amounts of sales on each day.

Figure 2-2-37 Format used to output a detailed list of sales

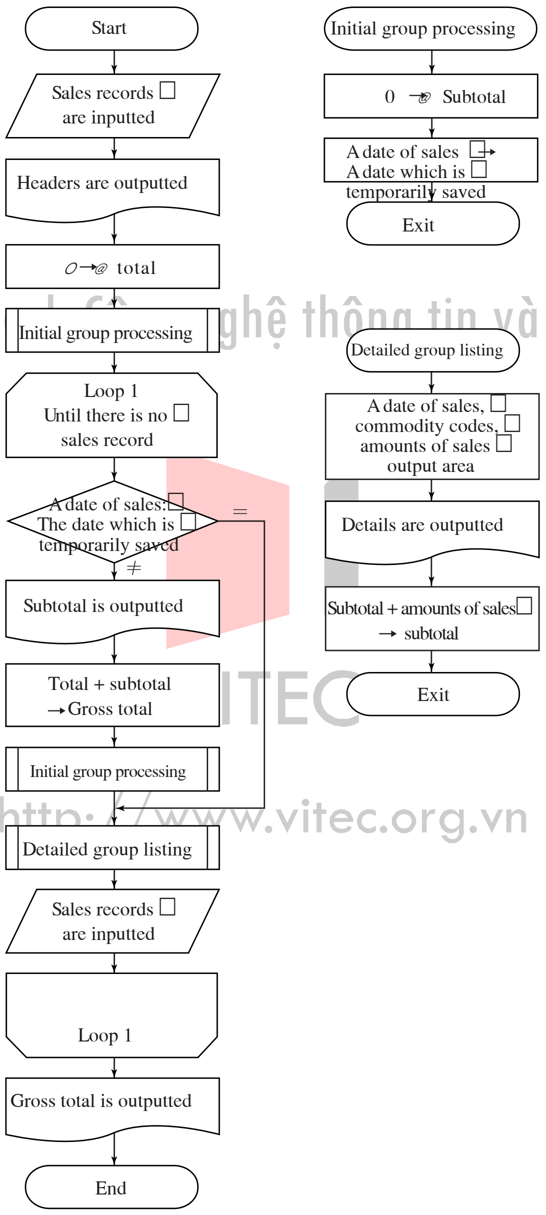


```
  Date □     Commodity □   Amounts □
  of sale      code        of sales
  10 ˆ10       S001        100,000   }
  10 ˆ10       S002         50,000   }  Detailed listing on Oct. 10
  10 ˆ10       S004        250,000   }
 Total amounts of sales    400,000   ΕΕΕΕGroup total on Oct. 10
  10 ˆ11       S001        200,000   }  Detailed listing on Oct. 11
  10 ˆ11       S003        350,000   }
 Total amounts of sales    550,000   ΕΕΕΕGroup total on Oct . 11
  10 ˆ12       S002        150,000   }
  10 ˆ12       S003        300,000   }  Detailed listing on Oct. 12
  10 ˆ12       S004        200,000   }
 Total amounts of sales    650,000   ΕΕΕΕGroup total on Oct. 12
Gross amounts of sales   1,600,000   ΕΕΕΕGross total of amounts on sales files
```

③ Flowchart and the detailed scheme

To obtain group totals, a division between groups must be discerned. For this particular subject, a point where a date of sale changes becomes a division. Therefore, it is necessary to determine whether or not a date of sales in a newly loaded record matches that in the most recently processed records. To do this, a key (a date of sales) is temporarily saved before the first record in a group is processed, and it is compared with a key (a date of sales) of the past records.
Figure 2-2-38 shows the flowchart of the group control.

Figure 2-2-38 Flowchart of the group control


Start
Sales records are inputted
Headers are outputted
0 → total
Initial group processing
Loop 1
Until there is no sales record
A date of sales: The date which is temporarily saved
=
≠
Subtotal is outputted
Total + subtotal → Gross total
Initial group processing
Detailed group listing
Sales records are inputted
Loop 1
Gross total is outputted
End
Initial group processing
0 → Subtotal
A date of sales → A date which is temporarily saved
Exit
Detailed group listing
A date of sales, commodity codes, amounts of sales output area
Details are outputted
Subtotal + amounts of sales → subtotal
Exit

(2) Updating more than one file

If more than one file is compared using the same criteria by matching, all files must be sorted in the sequence of keys, as in the case of the group control.
File processing tasks are as follows:

- Merging files
- Matching files
- Updating files
- Maintaining files

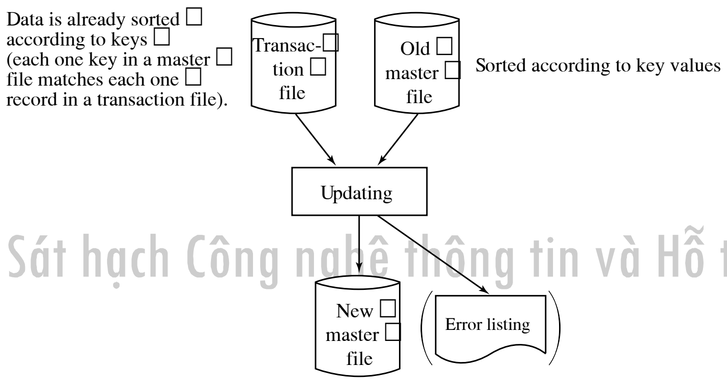
This section describes the task of file updating. File updating is to update master files based on data contained in transaction files. If a master file is a sequential file, it must be completely re-created. If a master file can be accessed randomly, a new master file does not need to be created since records can be retrieved using keys, and updated. Here, we deal with the updating procedure if a master file is a sequential file.

① 1:1 updating

1:1 updating means updating to be executed if each one record in a master file matches each one record in a transaction file. (See Figure 2-2-39.)
Multiple files are read and processing routines are determined by comparing sizes of each key as follows:
Sizes of keys

- TR key = MS key Data is updated.
- TR key > MS key Data is copied (data is written into a new master file).
- TR key < MS key Data is added to a new master file (it may be the case where this status is considered an error and the error routine takes place).

Figure 2-2-39 Image of the 1:1 updating


Data is already sorted according to keys (each one key in a master file matches each one record in a transaction file).
Transaction file
Old master file
Sorted according to key values
Updating
New master file
Error listing

<Transaction file>

Commodity code	Commodity name	Amounts of sales
S001	Television	134,000
S004	Video recorder	98,000
S006	Personal computer	156,000
S007	Printer	43,000
S009	Scanner	86,000

<Old master file>

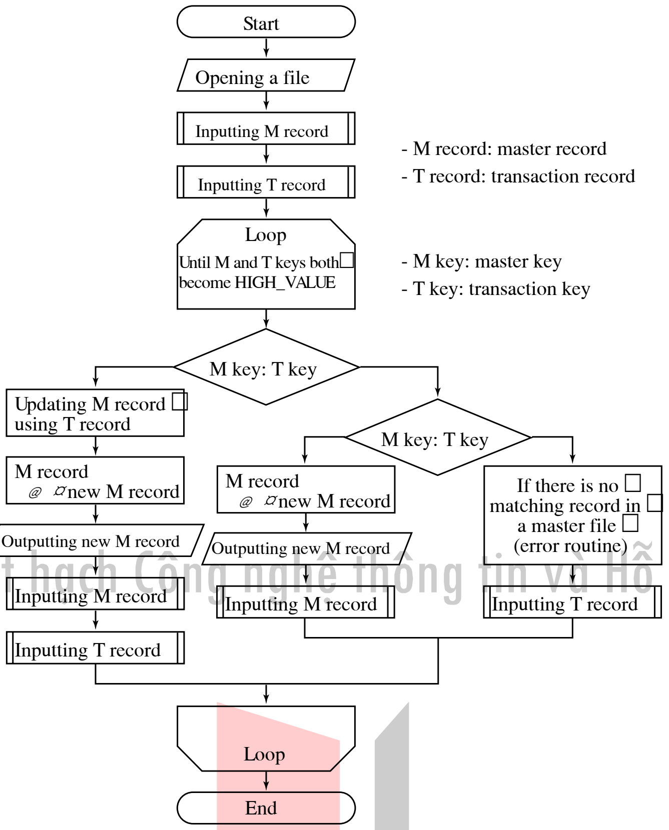
Commodity code	Commodity name	Amounts of sales
S001	Television	656,000
S002	Stereo	423,000
S004	Video recorder	256,000
S006	Personal computer	432,000
S007	Printer	83,000
S008	Digital camera	92,000

Commodity code	Commodity name	Amounts of sales
S001	Television	790,000
S002	Stereo	423,000
S004	Video recorder	354,000
S006	Personal computer	588,000
S007	Printer	126,000
S008	Digital camera	92,000
S009	Scanner	86,000

<New master file>

Figure 2-2-40 shows the flowchart of the 1:1 updating.

Figure 2-2-40 Flowchart of the 1:1 updating



- M record: master record
- T record: transaction record

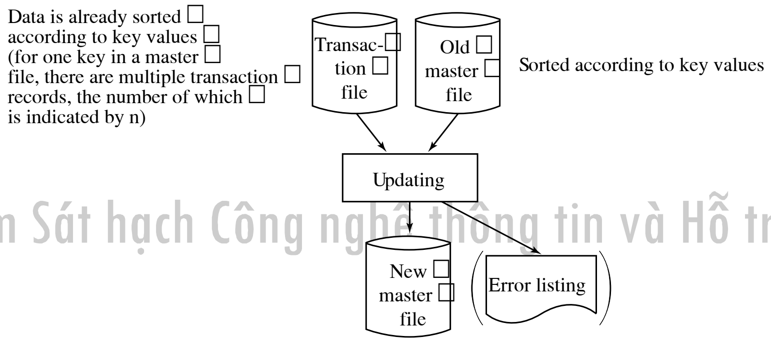
- M key: master key
- T key: transaction key

② 1:n updating

1:n updating is the updating routine used if one record in a master file matches more than one record in a transaction file. (See Figure 2-2-41.)
In principle, multiple files are read, as in the case of the 1:1 updating, and processing routines are determined by comparing sizes of keys as follows:
Sizes of keys

- TR key = MS key	Data is totaled and the results are stored in a work area.
- TR key > MS key	Data is updated.
- TR key < MS key	Data is added to a new master file (it may be the case where this is considered an error and the error routine takes place).

Figure 2-2-41 Image of the 1:n updating


Data is already sorted according to key values (for one key in a master file, there are multiple transaction records, the number of which is indicated by n)
Transaction file
Old master file
Sorted according to key values
Updating
New master file
Error listing

<Transaction file>

Commodity code	Commodity name	Amounts of sales
S001	Television	134,000
S001	Television	121,000
S004	Video recorder	98,000
S006	Personal computer	156,000
S007	Printer	43,000
S007	Printer	41,000
S009	Scanner	86,000

Multiple
Multiple

<Old master file>

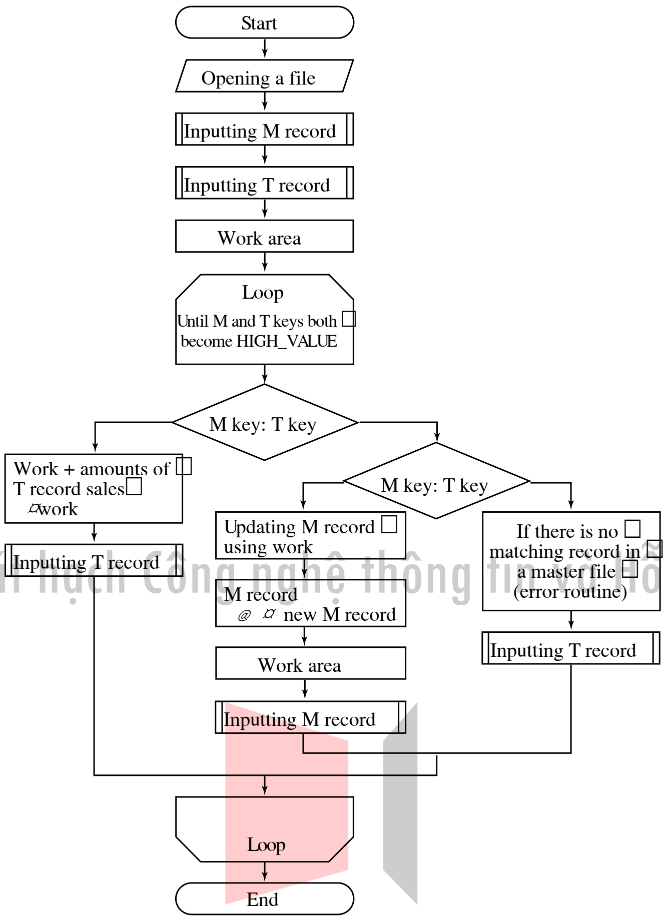
Commodity code	Commodity name	Amounts of sales
S001	Television	656,000
S002	Stereo	423,000
S004	Video recorder	256,000
S006	Personal computer	432,000
S007	Printer	83,000
S008	Digital camera	92,000

Commodity code	Commodity name	Amounts of sales
S001	Television	911,000
S002	Stereo	423,000
S004	Video recorder	354,000
S006	Personal computer	588,000
S007	Printer	167,000
S008	Digital camera	92,000
S009	Scanner	86,000

<New master file>

Figure 2-2-42 shows the flowchart of the 1:n updating.

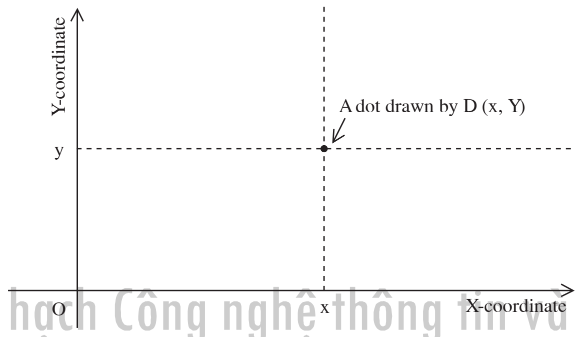
Figure 2-2-42 Flowchart of the 1:n updating



2.2.6 Drawing figures

In the present world of computers, CAD, CG and other figure drawing technologies are used. In an algorithm for drawing figures, a figure is treated as a set of dots. How a simple straight line and a circle can be drawn is explained in this section.
A function D (x, y) is used to draw a dot at an intersection point where a line along the x-coordinate meets a line along the y-coordinate, as shown in Figure 2-2-43.

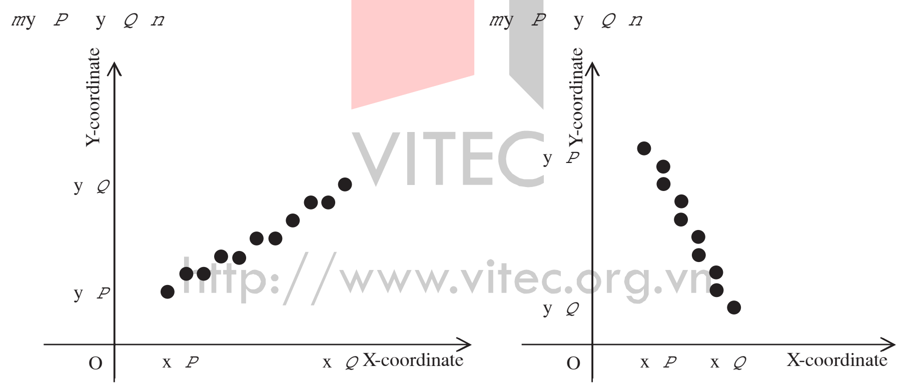
Figure 2-2-43 How the function D (x, y) works



(1) Drawing a straight line

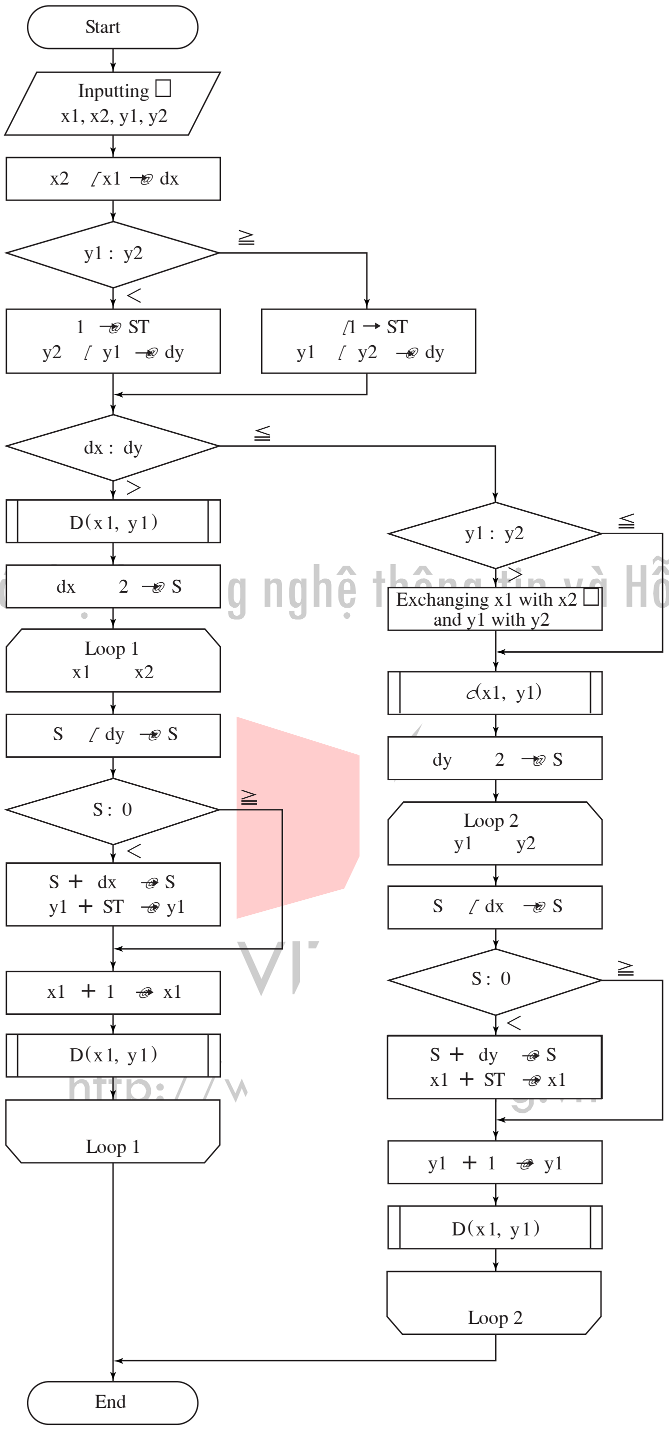
Figure 2-2-44 shows the routine of straight line drawing.

Figure 2-2-44 Examples of straight line drawing



The flowchart shown in Figure 2-2-45 is an algorithm for drawing a straight line that connects two given points in the coordinate, i.e., (x1, y1), (x2, y2) ｜0<x1≦x2, 0<y1, 0<y2｜. Although this algorithm can draw straight lines running obliquely in upper or lower right directions by drawing dots on an integer coordinate, it is designed to allow lines to look like straight lines. Also, multiply-divide operations are kept to a minimum and add-subtract operations are mostly used to increase the processing speed.

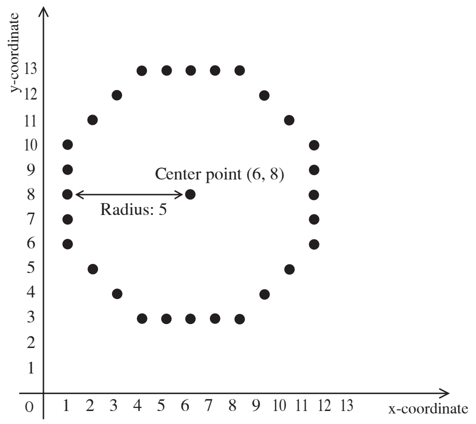
Figure 2-2-45 Flowchart of straight line drawing



(2) Drawing a circle

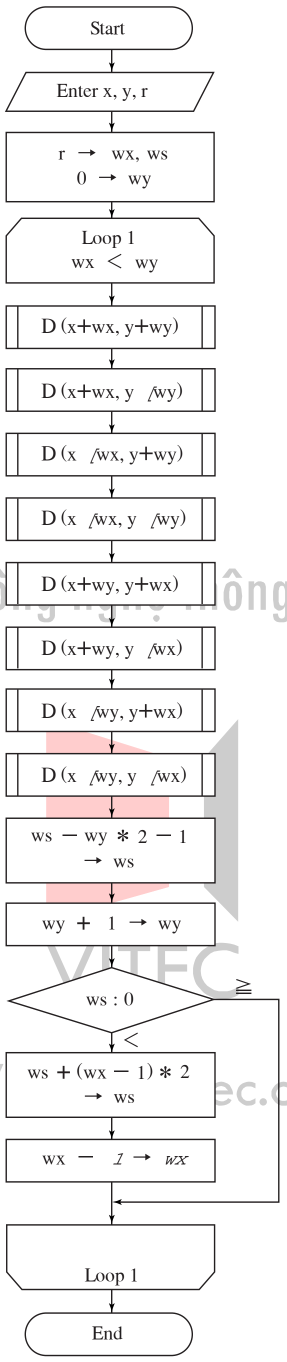
Figure 2-2-46 shows how a circle is drawn.

Figure 2-2-46 How a circle is drawn



The flowchart in Figure 2-2-47 shows the algorithm for drawing a circle based on the coordinate (x, y) of a given center point and the radius r. To draw this circle, a trigonometric function that needs a long execution time is not used.

Figure 2-2-47 Flowchart for drawing a circle



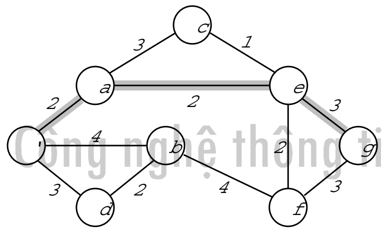
2.2.7 Graph

The graph that we discuss in this section is one made up of arcs formed by connecting more than one point. Basically an undirected graph is assumed. A directed graph can also be used, depending on the type of problem to be solved.
The shortest path problem is described here as one of the representative graph problems.

(1) Shortest path problem

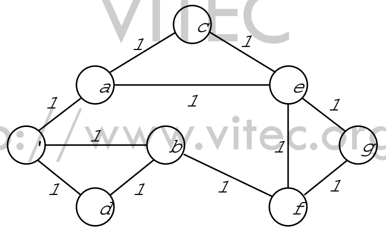
As shown in Figure 2-2-48, there are routes and nodes, and the shortest route that runs from point A to point H must be found. Because the numbers along each route show a distance, you can see at once that the thick line is the shortest route.

Figure 2-2-48 Map (the thick line is the shortest route)



The graph shown in Figure 2-2-48 that has numbers along each route to show a route distance is called a weighted graph. Although we can discern the shortest route by sight, an algorithm must be used to enable a computer to find the shortest route through calculations.
We will here describe three route search methods: the depth-first and the breadth-first search method which assume that the distance of each route is 1, and the Dijkstra's search method which is a major method for finding the shortest route.

Figure 2-2-49 Graph in which the distance of all routes is set to 1



① Depth-first search method

Using the depth-first search method, one route is chosen as a start point and a search starts to find a route to a destination using the stack.

<Procedure> (Figure 2-2-50)

1. A node at the start point is put into the stack.
2. A node is picked from the stack. Nodes adjacent to a picked node are checked and those that have never before been pushed into the stack are chosen. The chosen nodes are pushed into the stack.
3. When the chosen nodes are pushed into the stack, the node picked in step 2 above is stored in the array.
4. Steps 1, 2 and 3 are repeatedly executed until a target node is put into the stack or the stack becomes empty.

Figure 2-2-50 The state of the stack

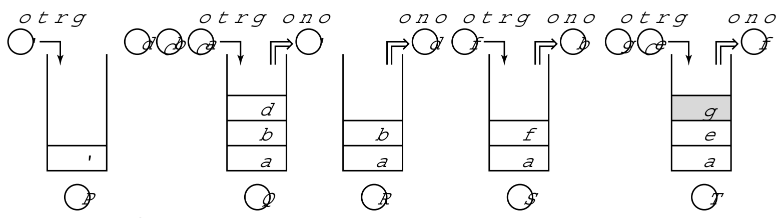
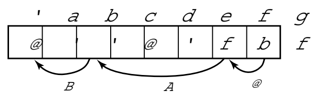


Figure 2-2-51 The state of the array



Before the chosen nodes are pushed into the stack, the node picked in step 2 above must be stored in the array. For example, if A is picked and B, C and E are pushed into the stack, A must be stored in the elements of B, C and E. After a target node has been put into the stack, nodes are traced one by one sequentially to find the shortest route. (The same routine is executed in the case shown in Figure 2-2-53.)

<Results>

A search executed with the depth-first search method results in the route A-C-G-H shown in Figure 2-2-51. Although the above Figures show the basic scheme of the search execution routine, what happens during an actual search is more complex. That is, assuming that the shortest route cannot be found, different routes are repeatedly searched, that is, as the search routine reaches the step before the step of putting H, shown in Figure 2-2-50, step ⑤ into the stack, it leaps back to step 1 and repeats all the steps. This routine is repeated until the stack becomes empty so that all routes are counted to find the shortest route.

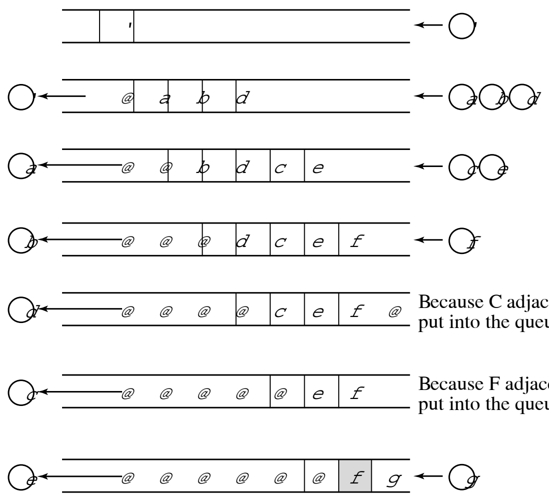
② Breadth-first search method

Using the breadth-first search method, all possible routes that can lead to a destination are searched using the queue.

<Procedure> (Figure 2-2-52)

1. A node at the start point is put into the queue.
2. One node is picked from the queue. Nodes adjacent to the picked node are checked and those that have never before been put into the queue are chosen. The chosen nodes are put into the queue.
3. The node picked in step 2 is stored in the array.
4. Above steps 1, 2 and 3 are repeatedly executed until a target node is reached or the queue becomes empty.

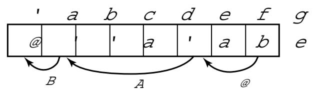
Figure 2-2-52 The state of the queue



Because C adjacent to E has already been put into the queue, nothing takes place.

Because F adjacent to D has already been put into the queue, nothing takes place.

Figure 2-2-53 The state of the array



Before a node is put into the queue, the node picked in step 2 is stored in the array. For example, if A is picked and B, C and E are put into the queue, A is stored in the elements of B , C and E .

A search executed using the breadth-first search method results in the shortest route A-B-F-H shown in Figure 2-2-53. All nodes are searched one by one, as shown above, while calculations are being made to find the best route.

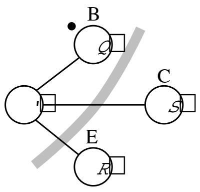
③ Dijkstra's search method

The Dijkstra's search method is a method to which the breadth-first search method is applied.

<Procedures> (Figure 2-2-54)

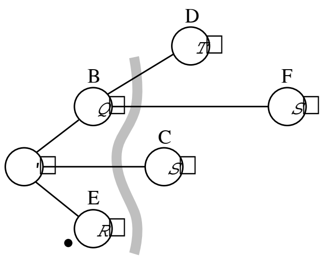
1. The distance to each node adjacent to the start node is measured and the node located at the shortest distance is chosen (this chosen node is defined as X).
2. The distance to each node adjacent to X from the start node as well as the distance to nodes except X from the start node are measured and the node located at the shortest distance is chosen.
3. Step 2 is repeatedly performed on all nodes until a target node is reached.
4. By adding the distance attached to each node, the shortest route can be established.

Figure 2-2-54 Dijkstra search method

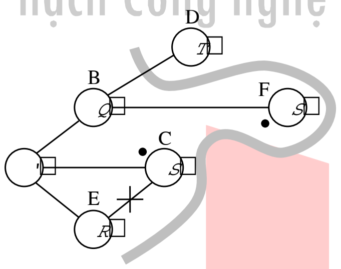


The shaded line is the line that separates nodes or routes that have already been processed.

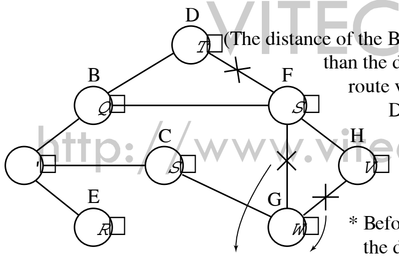
The distance from the start node to each adjacent node is compared and the node located at the shortest distance is marked.



The distance to a node located on the A-to-B route is compared with the distance of the node adjacent to A and the node at the shortest distance is marked.



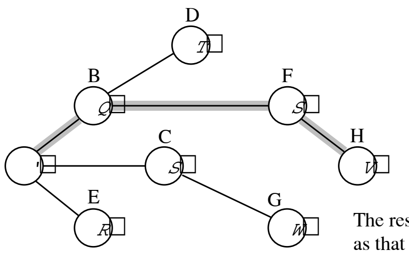
C adjacent to E located at the shortest distance can be reached via A. Because the distance from A to C is shorter than the distance from A to C via E, the E-to-C route is cut off.



(The distance of the B-to-F route is shorter than the distance of the B-to-F route via D. Therefore, the D-to-F route is cut off.)

* Before a route is cut off, the distance is considered.

(Both routes are cut off.)



<The search is completed.>

The result becomes the same as that obtained using the breadth-first search method.

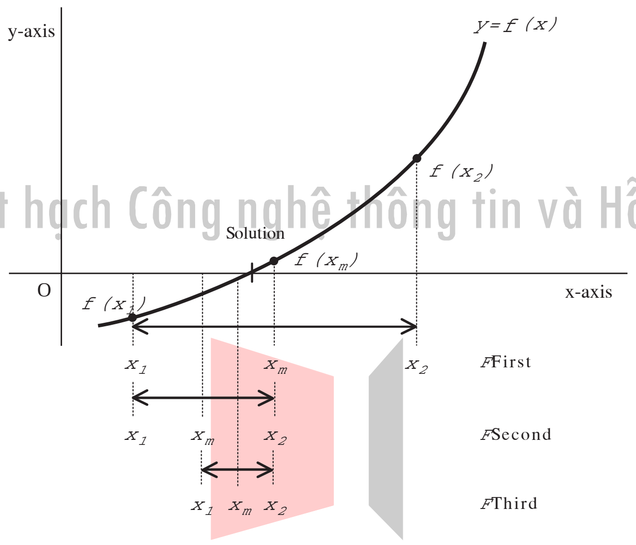
2.2.8 Numeric calculation

(1) Solution of algebraic equations

① Bisection method

The bisection method is the simplest solution of higher-degree equations. A commonly used higher-degree equation is defined as y = f(x) and various values are assigned to x. The line is plotted by assigning values into x. The value of x when the line intersects y = 0 (x-axis) becomes the root of this equation.
Figure 2-2-55 is a graph plotted based on y = f(x).

Figure 2-2-55 Graph plotted based on y = f(x)



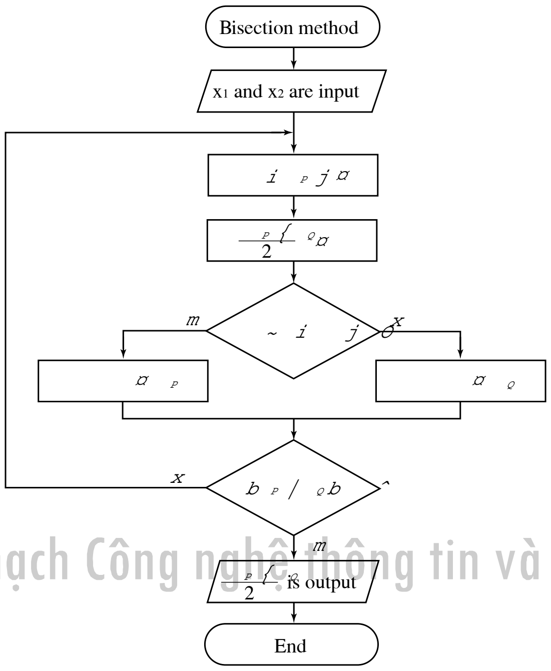
$f(x_1) \cdot f(x_2)<0$ shows that there is at least one solution within the $[x_1, x_2]$ region in which the line intersects the x-axis. Using the bisection method, this $[x_1, x_2]$ region is divided into two sections. To obtain an approximate solution, the routine of dividing a region into two sections is repeatedly executed to choose a section (indicated by ↔) where there is a solution.

Figure 2-2-56 Algorithm of the bisection method

Step 1	The point xm that divides a region into two sections is calculated.
Step 2	Each of the divided sections $[x_1, x_m]$ and $[x_m, x_2]$ is checked to determine in □ which section there is a solution. If $f(x_1)$ Ε $f(x_m)<0$, there is a solution in the left □ section. If not, there is a solution in the right section.
Step 3	If $f(x_1)$ Ε$f(x_m)<0$, x_m Øx_2. If not, x_m Øx_1.
Step 4	A judgment made -- whether the section length $\mid x_1-x_2 \mid$ is smaller than the □ convergence judgment value ˆor not -- is used to determine whether the □ approximate solution has been obtained or not. If $\mid x_1-x_2 \mid$ is larger than ˆ, □ the routine goes back to step 1 and repeats steps 1 through 4.

Figure 2-2-57 shows the flowchart of the algorithm for obtaining the solution of f(x) = 0 using the bisection method.

Figure 2-2-57 Flowchart of the algorithm for obtaining the solution of f(x) = 0 using the bisection method



② Newton's method

Newton's method assumes that an approximate solution of a higher-degree equation is already known. The approximate solution is repeatedly corrected to obtain a true solution. Newton's method is superior to other methods in that the convergence speed is faster and both real and imaginary solutions can be obtained.
Figure 2-2-58 is a graph plotted using Newton's method.

Figure 2-2-58 Newton's method

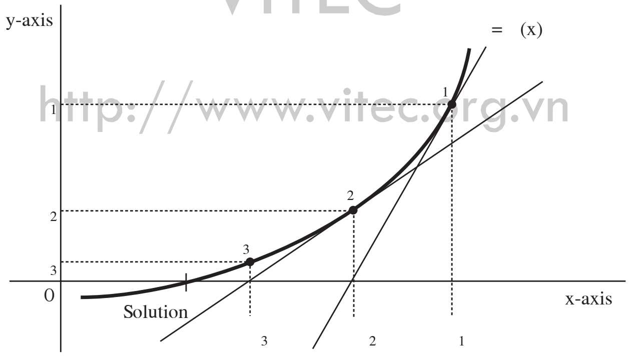


Figure 2-2-59 Algorithm of Newton's method

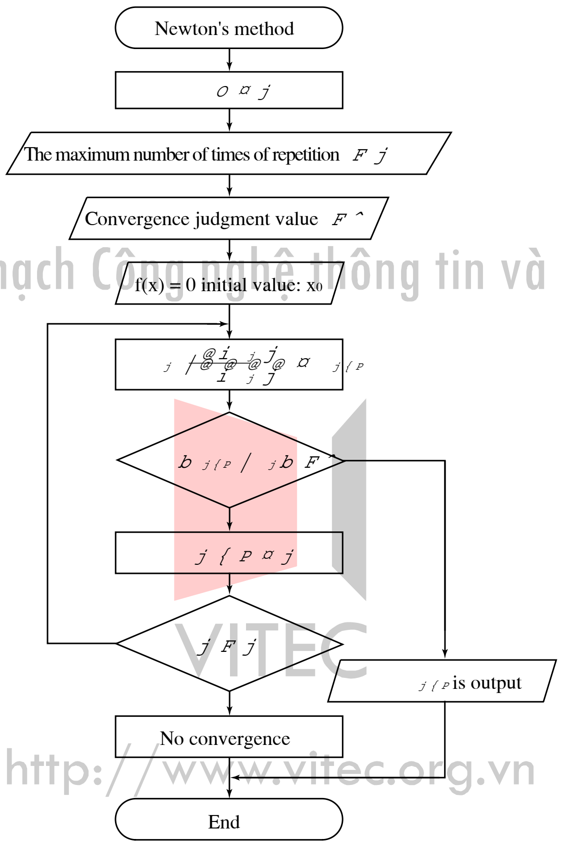
Step 1	A tangential line y = f(x) at point p_1 of the coordinate x_1, y_1 is drawn □ and point x_2 where the tangential line intersects x-axis is obtained.
Step 2	A tangential line y = f(x) at point p_2 of the coordinate x_2, y_2 is likewise □ drawn and point x_1 where the tangential line intersects x-axis is □ obtained. As this step is repeatedly executed the tangential line □ moves closer to a solution.
Step 3	The difference between adjacent approximate values obtained in step □ 2 is compared with a predetermined convergence judgment value □ precision. Steps 1 and 2 are repeatedly executed until this difference □ becomes smaller than the predetermined convergence judgment value.

Because the equation for a tangential line at point p1 is y-f(x_1) = f'(x_1)(x-x_1), point x_2 where a tangential line intersects x-axis can be obtained using the following equation:

$$x_{i+1} = x_1 - \frac{f(x_i)}{f'(x_i)} \quad (i = 0, 1, 2, ...)$$

Figure 2-2-60 shows the flowchart of the algorithm for obtaining the solution of f(x) = 0 using Newton's method.

Figure 2-2-60 Flowchart of the algorithm for obtaining the solution of f(x) = 0 using Newton's method


Newton's method
The maximum number of times of repetition
Convergence judgment value
f(x) = 0 initial value: x0
is output
No convergence
End

(2) Numerical integration

Numerical integration methods are used to calculate the area of a figure enclosed by curved lines. This section describes the trapezoidal rule and Simpson's method of all commonly used numerical integration methods.

① Trapezoidal rule

Figure 2-2-61 shows the basic concept of the trapezoidal rule.

Figure 2-2-61 Basic concept of the trapezoidal rule

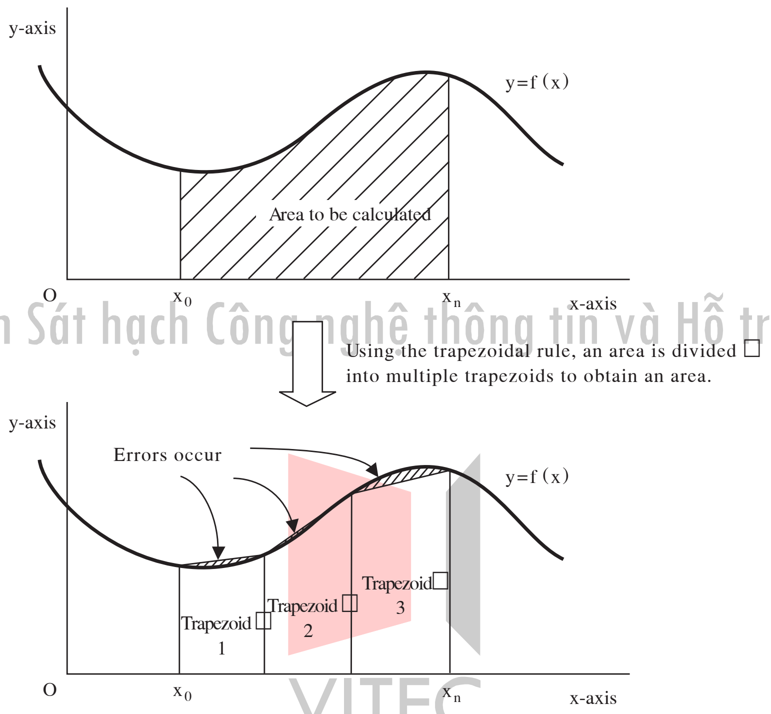


Figure 2-2-62 shows the procedure for calculating an area enclosed by the curved line y = f(x), a region along the x-axis and a portion enclosed by the x-axis.

Figure 2-2-62 Procedure for calculating an area using the trapezoidal rule

Step 1	Vertical lines are drawn in an area at regular intervals to divide it into parts.
Step 2	Assuming that the curved line of each divided area is a straight line, each divided area is regarded as a trapezoid.
Step 3	Each divided area is regarded as a trapezoid and each area is calculated. An area of one trapezoid = (upper base + lower base) x height 2
Step 4	Areas of all trapezoids are totaled to obtain an area.

If an area is calculated using the trapezoidal rule, errors occur with respect to shaded portions, as shown in Figure 2-2-61, since a true area is different from a trapezoidal area. To reduce errors, the number of trapezoids is increased. Figure 2-2-63 shows how an area is divided into trapezoids.

Figure 2-2-63 An area divided according to the trapezoidal rule

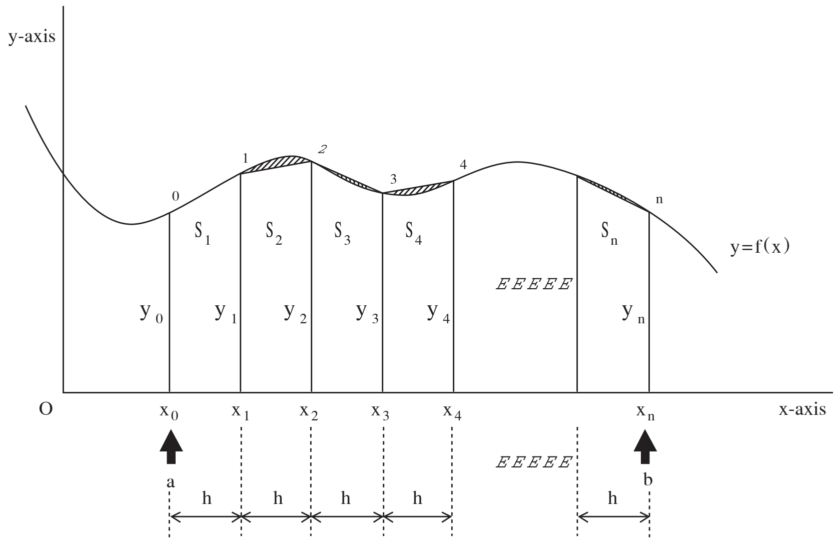


Figure 2-2-64 shows the algorithm for calculating an area using the trapezoidal rule based on Figure 2-2-63.

Figure 2-2-64 Algorithm of the trapezoidal rule

Step 1 The portion where integration takes place, i.e., [a, b] of y = f(x), is divided into parts, the number of which is defined as n.
Step 2 Points where lines set at regular intervals intersect x-axis are defined as x_0, x_1, x_2, ... x_n from left to right.
Step 3 The values of functions at above intersection points are defined as y_0, y_1, y_2, ... y_n.
Step 4 Points where lines set at regular intervals intersect curved lines are defined as p_0, p_1, p_2, ... p_n. With x, y and p points connected, a trapezoid is formed.
Step 5 Providing that an area of each trapezoid is S_1, S_2, S_3, ... S_n,

$$S_1 = \frac{h}{2}(y_0 + y_1)$$

$$S_2 = \frac{h}{2}(y_1 + y_2)$$

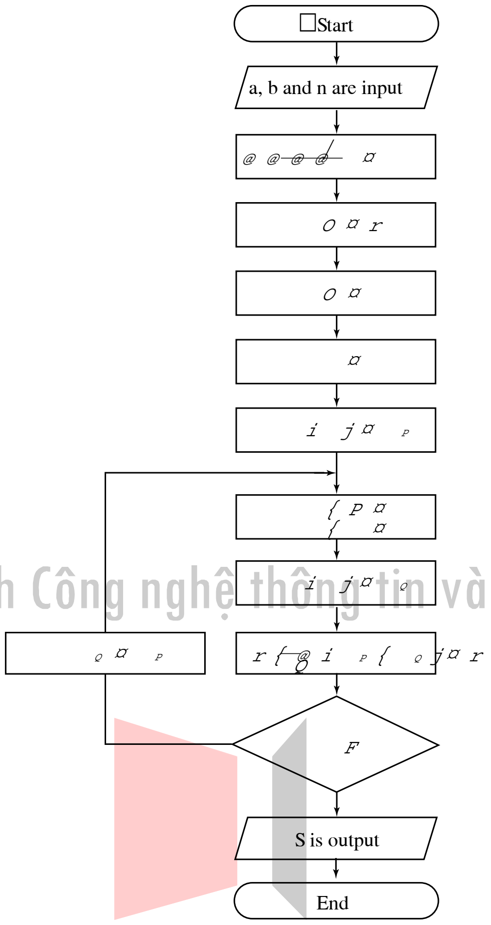
$$\vdots$$

$$S_n = \frac{h}{2}(y_{n-1} + y_n)$$

Each area is calculated using the above equations.
Step 6 An area enclosed by the curved line can be obtained by totaling areas of each divided area: $S = S_1+S_2+S_3+ ...+S_n$

Figure 2-2-65 shows the flowchart of the algorithm for calculating an area using a computer and the trapezoidal rule.

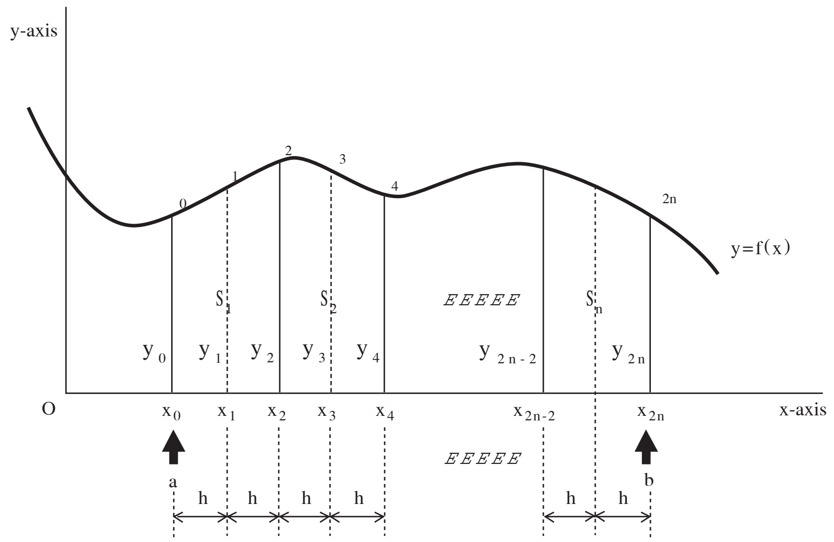
Figure 2-2-65 Flowchart of the approximate area calculation using the trapezoidal rule



② Simpson's method

Using the trapezoidal rule, a curved line is divided at regular intervals and intersection points on the curved line are connected to form trapezoids. Areas of each trapezoid are totaled to obtain a whole area. Although errors can be reduced by increasing the number of trapezoids, it still holds that the greater the number of trapezoids, the longer it takes to execute totaling. Furthermore, because the method is based on the scheme wherein an area enclosed by three straight lines and one curved line is regarded as a trapezoid, there is concern over the accuracy of the result obtained.
As a solution, Simpson's method is used as shown in Figure 2-2-66. Using this method, a curved line is approximated to an easy-to-handle parabola to calculate an area.

Figure 2-2-66 Area divided using Simpson's method



To calculate an area S1 enclosed by y = f(x), $x = x_0$, $x = x_2$ and x-axis shown in Figure 2-2-66 using Simpson's method, f(x) is considered a parabola that runs through p_0, p_1 and p_2, and S_1 is approximated as follows:

$$S_1 = \frac{(y_0 + 4y_1 + y_2)h}{3}$$

This method is quite different from the trapezoidal rule in that an area is equally divided into 2n (equal, even-number division), not into n.

Figure 2-2-67 shows the algorithm for calculating an area shown in Figure 2-2-66 using Simpson's method.

Figure 2-2-67 Algorithm for calculating an area using Simpson's method

Step 1 The section [a, b] of the function y = f(x) where integration takes place is equally □ divided into 2n (equal, even-number division). H

Step 2 Points where dividing lines intersect x-axis are defined from the left as x_0, x_1, x_2, □ ... x_{2n}.

Step 3 Function values at each intersection point are defined as y_0, y_1, y_2, ... y_{2n}.

Step 4 Points where dividing lines intersect a curved line are defined as p_0, p_1, p_2, ... p_{2n}.

Step 5 Three points $p_{2i-2}(x_{2i-s}, y_{2I-2})$, $p_{2i-1}(x_{2i-1}, y_{2i-1})$ and p_{2i} (x_{2i}, y_{2i}) in two sections are □ approximated to a parabola to calculate an area Si.

$$S_1 \quad \frac{í y_0 \ \{4y_1 \ \{y_2 \ j}{R}$$

$$S_2 \quad \frac{í y_2 \ \{4y_3 \ \{y_4 \ j}{R}$$

C

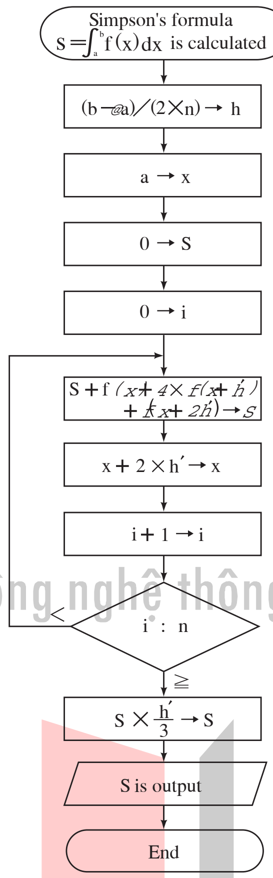
$$S_n \quad \frac{í y_{2n-2} \ \{4y_{2n-1} \ \{y_{2n} \ j}{R}$$

Step 6 Areas Si in each section are totaled to obtain an area S enclosed by a curved line.

@ @ @ @ @ @ @ r n1 { n2 { c { rn

Figure 2-2-68 shows the flowchart of the algorithm for calculating an area using a computer and Simpson's method.

Figure 2-2-68 Flowchart of the area approximation calculation using Simpson's method



Simpson's formula
$S = \int_a^b f(x)dx$ is calculated
$(b-a)/(2\times n) \to h$
$a \to x$
$0 \to S$
$0 \to i$
$S + f(x) + 4\times f(x+h') + f(x+2h') \to S$
$x + 2\times h' \to x$
$i + 1 \to i$
$i : n$
<
≧
$S \times \frac{h'}{3} \to S$
S is output
End

2.2.9 Collation algorithm

In the collation algorithm, values stored in the array are compared to obtain a solution. A search of character strings described under Section 2.2.4, Character string processing, also uses one of the collation algorithms.
This section describes the stable marriage problem to explain one of the representative collation algorithms.

(1) Stable marriage problem

In solving the stable marriage problem, stable pairs of men and women are determined.
Stable means a state in which men and women feel more fondness for their present partners than others. Specifically, supposing that there are three men and three women, men are named A, B and C and women are named a, b and c. The levels of fondness (high to low levels in the order 1, 2 and 3) are shown in the table below:

<Men>

	1	2	3
A	b	a	c
B	c	a	b
C	c	b	a

<Women>

	1	2	3
a	A	C	B
b	C	B	A
c	A	B	C

If they are paired as
P = { (A, a), (B, b), (C, c) },
A feels fonder of b than a, who is the present partner. b feels fonder of B, who is the present partner, than A.

Therefore, there should be no problem. B feels fonder of a and c than b, who is the present partner. Because a feels fonder of A, who is the present partner, than B, there should be no problem. However, c feels fonder of B than C who is the present partner. This state is called an unstable state. By changing partners, they can be paired as

P = { (A, a), (B, c), (C, b) }.

This state of pairing can be analyzed as follows:

①A feels fonder of b than a, who is the present partner. → b feels fonder of C, who is the present partner, than A.
② B feels fonder of c who is the present partner.
③ C feels fonder of c than b, who is the present partner. → c feels fonder of B, who is the present partner, than C.
④ a feels fondest of A who is the present partner.
⑤ b feels fondest of C, who is the present partner.
⑥c feels fonder of A than B, who is the present partner. → A feels fonder of a, who is the present partner, than c.

In this pairing, no pairs have an unstable factor. This result is called stable or stable matching.

Stable matching may not necessarily be determined uniquely. There may be a case where stable matching cannot be achieved using the approach described above. The stable marriage problem to be described in the following, is designed to achieve stable matching by specifying the conditions for determining pairs:

[Stable marriage problem]

N men and N women are looking for stable pairs.

- Men's and women's levels of fondness are preset in array M and F (the number of elements: N+1).
- As levels of fondness, element numbers 1 through 5 (high to low) are assigned to the numbers of each partner.

$\boxed{\text{Example}}$ If there are five men and five women

[Combination M: Levels of fondness seen from the side of the men]

	1	2	3	4	5	6 (PT)
M (1)	2	1	4	3	5	0
M (2)	1	3	4	2	5	0
M (3)	1	4	5	2	3	0
M (4)	5	4	1	3	2	0
M (5)	4	2	3	1	5	0

* A partner is entered in PT. 0 is set here as the initial value.

[Combination F: Levels of fondness seen from the side of the women]

	1	2	3	4	5	6 (PT)
F (1)	3	1	5	2	4	0
F (2)	2	1	4	3	5	0
F (3)	3	2	5	4	1	0
F (4)	5	2	1	3	4	0
F (5)	5	4	2	1	3	0

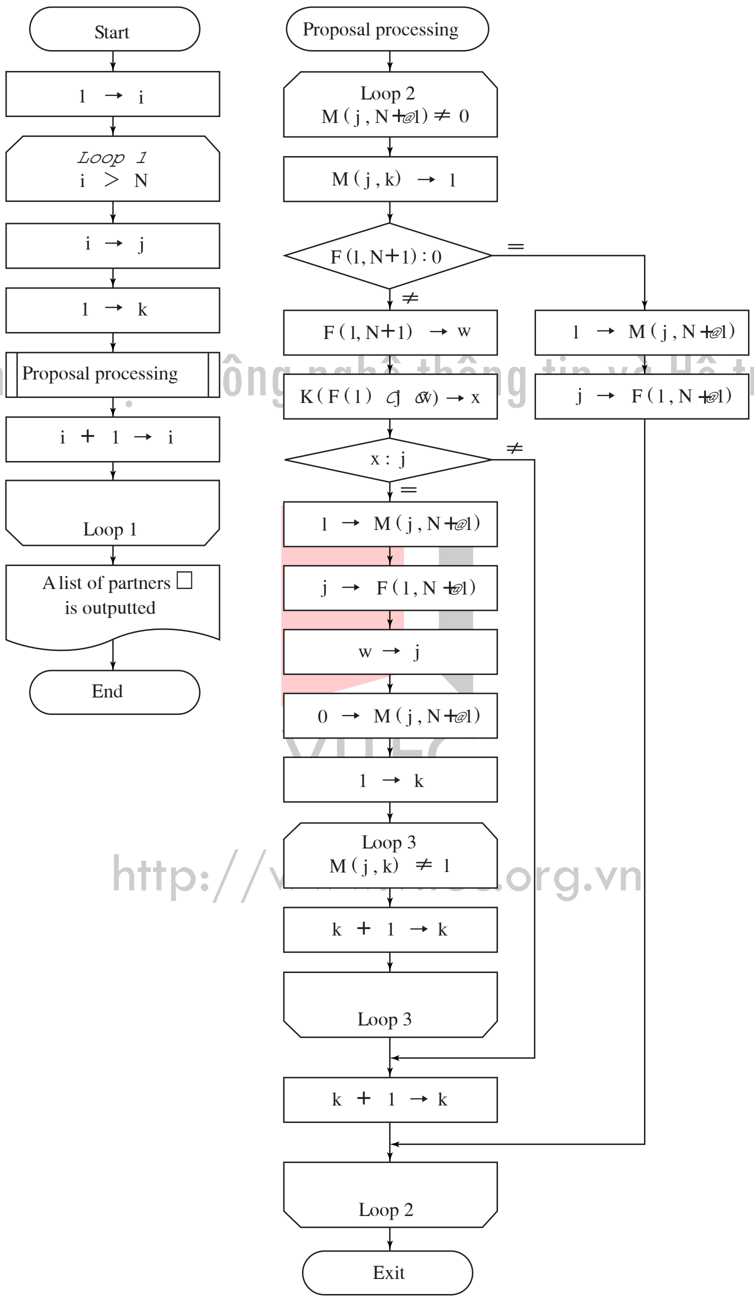
* A partner is entered in PT. 0 is set here as the initial value.

- Proposals are made in the order of M (1), M (2).... M (N).
- Men make proposals to women in high-to-low order of fondness. They repeatedly make proposals until they obtain OKs. It may happen, however, that the OKs thus obtained are canceled.
- Women give OK or NO on the following criteria:
  ① If one receives a proposal for the first time, she gives an OK and secures him as a partner.
  ② If she already has a man, she compares her fondness for the present partner and that for the one who has made the proposal.
- If she feels fonder of the present partner, she gives a NO.
- If she feels fonder of a man who has made a proposal, she cancels the present partner and secures the man who has made the proposal as a partner (OK).
- A man who obtained an OK but was canceled later begins making proposals to women in high-to-low

order of fondness. He repeatedly makes proposals until he obtains an OK.

In the flowchart shown in Figure 2-2-69, the function K (p1, p2, p3) is used to compare levels of fondness.
Function K (p1, p2, p3): This function is for returning either p2 or p3 in the array element p1, whichever is higher in the level of fondness. K (F(2), 1, 3) $\rightarrow$ 1
Figure 2-2-69 shows the flowchart of the algorithm of the stable marriage problem.

Figure 2-2-69 Flowchart of the algorithm of the stable marriage problem



2.2.10 Approximate and probability algorithms

Algorithms are generally used to obtain true values or solutions. There are problems that require a very long time to solve, and there are problems for which no algorithm is available. In this case, algorithms for obtaining the solutions whose errors or whose possibility of containing mistakes are very small are used.

(1) Approximation algorithms

Approximation algorithms are used to obtain an approximate solution in those cases where obtaining a true solution to a problem is impossible or it takes a very long time. Both Newton's method and Simpson's method described in Section 2. 2. 8 fall under the approximation algorithm category.
This section describes the knapsack problem as one representative approximation algorithm.

[Knapsack problem]

You have n goods which are different in weight and value. You want to sell them in a town but you cannot put all of them into your knapsack. Find what combination of goods maximizes their value.

We apply the following values to explain how the algorithm works:

The number of goods (items)	: 5
Weight of each item (kg)	: {2, 3, 5, 7, 8} which apply to item 1, item 2,......item 5 in that order
Value of each item (10,000 yen)	: {2, 4, 8, 9, 10} which apply to item 1, item 2, ... item 5 in that order
Capacity of the knapsack	: 10 kg

One way to solve this problem is to consider each combination of goods that would bring the total weight to 10 kg and compare the total value of goods in each combination. For this particular problem, the total value becomes highest if goods 1, 2 and 3 are packed into the knapsack, as shown in the table below:

Goods chosen	Total weight	Total value	
Goods 1, 2 and 3	2 + 3 + 5 = 10	2 + 4 + 8 = 14	← Maximum
Goods 1 and 5	2 + 8 = 10	2 + 10 = 12	
Goods 2 and 4	3 + 7 = 10	4 + 9 = 13	

Making the total weight equal to the knapsack capacity is not always the best solution. For example, if there were a sixth item, 9 kg in weight and \150,000 in value, the maximum value could be obtained by packing this item alone into the knapsack. To find the best solution, all possible combinations of goods must be considered. As goods increase in number, the number of combinations becomes enormous, and it takes a great deal of time to find the solution. As a solution to this, the approximation algorithm can be used.
The knapsack problem can be formulated as follows:

There are n positive integers $a_1, a_2, a_3, ..., a_n$, and $b_1, b_2, b_3, ..., b_n$, and a positive integer, c. Find a combination of $x_1, x_2, x_3, ..., x_n$ that maximizes the total sum of $b_i x_i$ (i = 1-n) where $x_i = \{0, 1\}$ and the total sum of $a_i x_i$ (i = 1-n) is equal to or smaller than c.

If this formula is considered in analogy to the problem previously discussed, a is the weight of goods, b is the value of goods, and c is the capacity of the knapsack. x is whether the good is packed into the knapsack or not. 0 means that the good is not packed and 1 means that the good is packed. Therefore, the knapsack problem and the solution can be expressed as follows:

a = | 2, 3, 5, 7, 8 |
b = | 2, 4, 8, 9, 10 |
c = 10
x = | 1, 1, 1, 0, 0 |

Using this formula, a search is made the 2^n times to check all possible combinations of 0, 1 in array x if the approximation algorithm is not used.
Using the approximation algorithm, however, unit values of all goods are first identified. A unit value means the value per weight and it is given by value (b_i)÷weight (a_i).

Unit values k = | 2/2, 4/3, 8/5, 9/7, 10/8 |

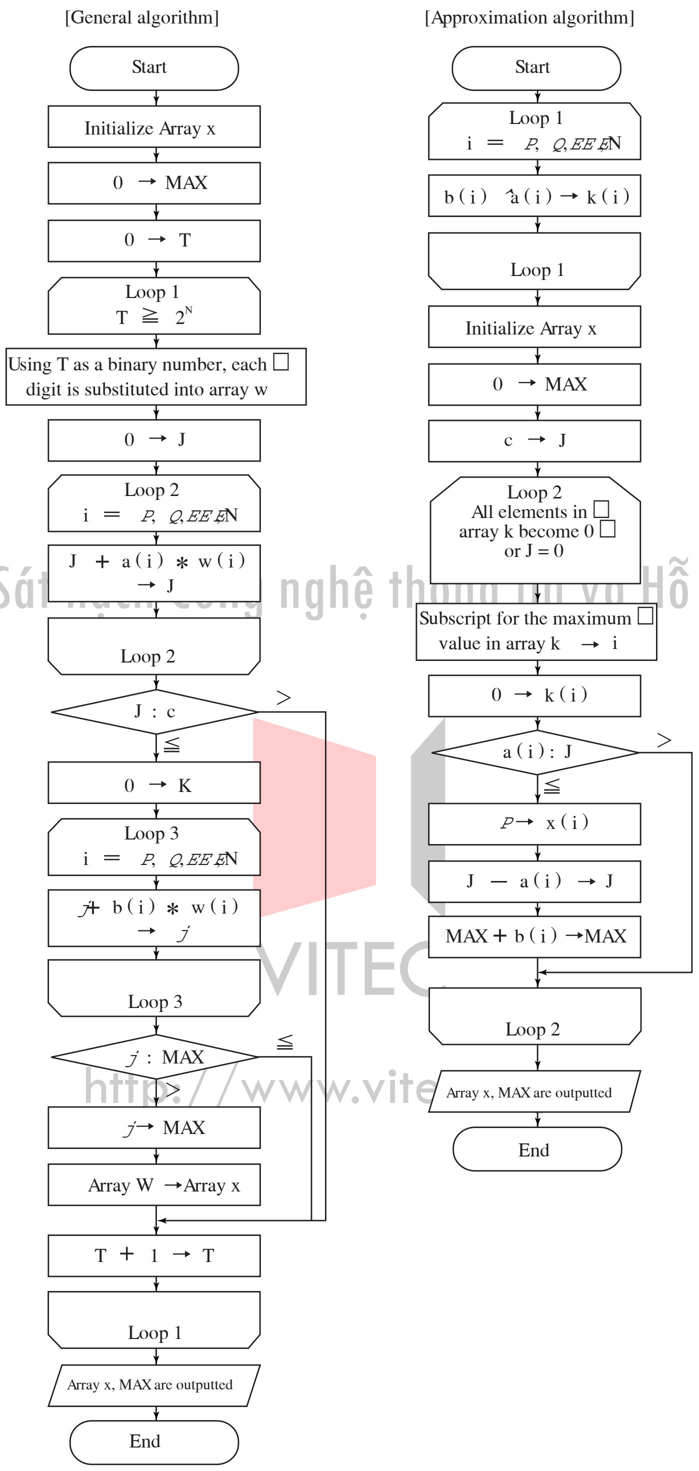
= | 1.00, 1.33, 1.60, 1.28, 1.25 |

After all unit values are identified, goods are packed into the knapsack in the order of high to low unit values. Because goods cannot be divided, goods that exceed the unused capacity of the knapsack cannot be packed.

① Item 3 with the highest unit value is packed: Unused capacity = 10-5 = 5
② Item 2 with the second-highest unit value is packed: Unused capacity = 5-3 = 2
③ Item 4 with a unit value less than that of item 2 cannot be packed: Unused capacity <the weight of item 4
④ Item 5 with a unit value less than that of item 4 cannot be packed: Unused capacity <the weight of item 5
⑤ Item 1 with a unit value less than that of item 5 is packed: Unused capacity = 2-2 = 0

This way to obtain a solution does not necessarily give the best solution. If values are defined as {2, 4, 8, 11, 10}, the approximation algorithm gives the solution: (item 1, item 2, item 3) = \140,000, though we already have the best solution: (item 2, item 4) = \150,000. The approximation algorithm, however, is used in many different fields as a method of quickly obtaining an approximate value very close to the best solution.
Figure 2-2-70 shows the flowchart of the algorithm for solving the knapsack problem.

Figure 2-2-70 Flowchart of the algorithm for solving the knapsack problem



(2) Probability algorithm

The probability algorithm uses a probabilistic approach to find a solution using random numbers. A probabilistic approach is one in which the appropriateness of a solution is examined in terms of probability, or that a solution is found based on the probability of occurrence of a certain event.
This section describes the primarity test problem as an algorithm for examining the appropriateness of a solution in terms of probability, and the case of obtaining a circular constant π as an algorithm for finding a solution based on the probability of occurrence of a certain event.

① Primarity test problem

Primarity test problem is a process of checking given N (= $2^S d+1$, d is an odd number) and determining whether it is a prime number or not. To solve the primarity test problem, given N is divided using integers 2, 3, ..., $\sqrt{N}$, and it is considered a prime number if it is not divisible without a remainder by any of the integers. Although a correct solution can be obtained using this approach, a much longer time period is required to arrive at a solution if the value of N is large. As a solution, Rabin's algorithm, designed with the following theorem, is used:

[Theorem of prime numbers]

If N (= $2^S d+1$, d is an odd number) is a prime number, either of two conditions shown below stand for the arbitrarily chosen positive integer, a (>1):
- $a^d = 1$ (mod N)
- $a^{2kd} = -1$ (mod N) where $0 \le k \le S-1$

If the arbitrarily chosen integer, a, does not meet the above conditions, N is considered a composite number, not a prime number. The probability that the arbitrarily chosen integer, a, for a composite number N can meet the above conditions is equal to or lower than 1/4. Therefore, if the arbitrarily chosen integer, a, meets the above conditions, the probability of the correctness of the judgement that N is a prime number is equal to or higher than 3/4.

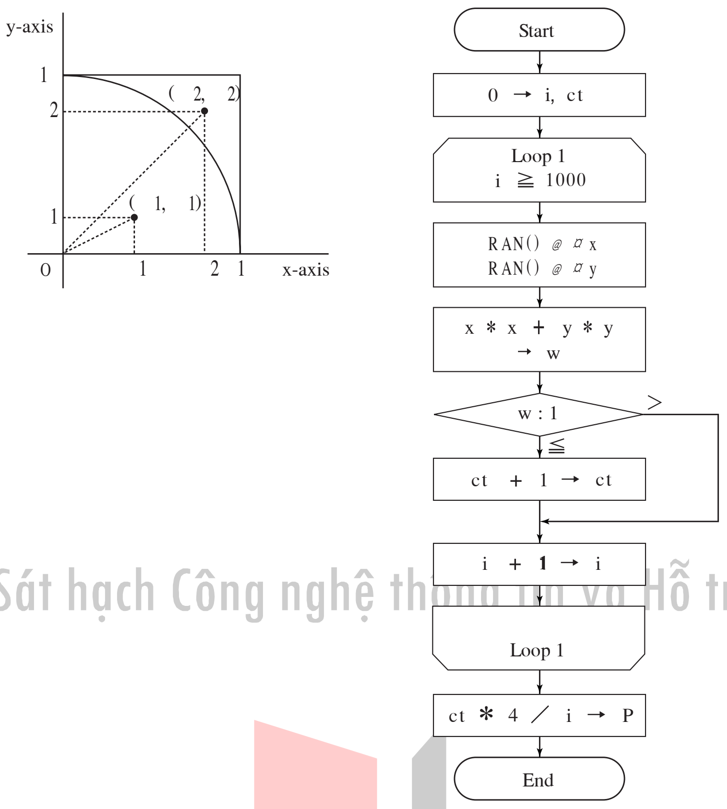
	Probability that the conditions can be met	Probability that the conditions cannot be met
Prime number N	100%	0%
Composite number N	25% or lower	75% or higher

If this algorithm is used, the possibility remains that the solution given by the algorithm is incorrect. This type of algorithm is called the probability algorithm with bounded errors. For the above particular example, since the probability that errors will occur is sufficiently low, the solution given by the algorithm should be considered dependable. In addition to the probability algorithm with bounded errors, the probability algorithm with no errors is sometimes used. Although this algorithm can theoretically assure the correctness of a given solution, it sometimes takes too long to execute the algorithm process, or it ends up giving no definite solution.
A probability algorithm with no errors is the probabilistic quick sort algorithm. This algorithm is designed to improve efficiency by randomly rearranging input data using random numbers.

② How to find a circular constant π

Figure 2-2-71 shows a circle of 1 in radius. The area enclosed by the axes and arc is a fourth of that of the complete circle having a radius 1. Therefore, it is π/4 (= 1 × 1 × π/4). On the other hand, the area of a square enclosed by four lines, x = 0, x = 1, y = 0 and y = 1 is 1. If points on this square are designated in a random manner, the probability that these designated points are inside the circle is π/4.
In the flowchart shown in Figure 2-2-71, 1,000 points are generated using the function RAN ( ) that can generate random numbers between 0 and 1. Whether generated points are inside the circle or not is determined by measuring the straight distance from (0, 0) to each point. To simplify the calculation, the root $\sqrt{\ }$ calculation is omitted. In the case of the example shown in Figure 2-2-71, it is judged that (x1, y1) is inside if $(x1^2+y1^2) \le 1$ and that (x2, y2) is outside if $(x2^2+y2^2) > 1$.

Figure 2-2-71 Flowchart of the algorithm for finding a circular constant π



Because a circular constant obtained in this way contains errors resulting from the characteristics inherent in methods used to generate random numbers, it is generally used in the representation containing a standard error (solution ± standard errors).
An algorithm like this one, that uses random numbers to solve mathematical problems, is called the Monte Carlo method.

2.3 Evaluation of algorithms

Algorithms should be evaluated and selected based on certain criteria. If the most appropriate algorithm can be found to solve a problem, the work of programming can be done efficiently and a high-quality program can be created.
This section describes the following three algorithm evaluation criteria:

- Evaluation in terms of computational complexity (a criterion for evaluating efficiency)
- Evaluation in terms of validity (a criterion for evaluating reliability)
- Evaluation in terms of representation (a criterion for evaluating elimination of redundancy and improvement of processing speed)

2.3.1 Evaluation by computational complexity

Computational complexity, which is also called computational measure, quantitatively shows the workload needed to perform calculations.
Computational complexity is a measure used to mathematically clarify how much time and memory area a computer requires to perform calculations.

Computational complexity is expressed in O (order).
Example: If an algorithm deals with data in such a way that the data increases twofold, threefold, fourfold....etc., the execution time likewise becomes twofold, threefold, fourfold...etc., computational complexity of this algorithm is $O(n)$.

(1) Types of computational complexity

There are two types of computational complexity.

- Time complexity measure: The maximum time that an algorithm needs to process all data
- Space complexity measure: The maximum area that an algorithm needs to process all data

In general, computational complexity assume the worst case. Computational complexity discussed in this section also assume the worst case, if not otherwise indicated. If average computational complexity is $O(n\log n)$ and if maximum computational complexity is $O(n^2)$, as in the case of quick sorting, average computational complexity should be considered more important.
Computational complexity of an algorithm is represented as data sizes (for example, a square of a size n) are not restricted by hardware or software limitations.

(2) Computation example

We now take up linear search (sequential search) and binary search algorithms as examples. Assuming that the data size is n, the number of times of comparison and computational complexity are as follows:

① Linear search (sequential search)

- Minimum number of times of comparison: 1 (computational complexities: $O(1)$)
- Average number of times of comparison: $n/2$ (computational complexities: $O(n)$)
- Maximum number of times of comparison: n (computational complexities: $O(n)$)

② Binary search

- Minimum number of times of comparison: 1 (computational complexities: $O(1)$)
- Average number of times of comparison: $[\log_2 n]$ (computational complexities: $O(\log_2 n)$)
- Maximum number of times of comparison: $[\log_2 n]+1$ (computational complexities: $O(\log_2 n)$)

2.3.2 Evaluation by validity

The validity of an algorithm is a criterion for making a judgment about whether or not an algorithm satisfies program specifications (module design document, etc.).
The validity can be further divided into three categories:

- Partial validity: Whether segments or functions satisfy specifications or not
- Terminability: Whether a program can terminate after a specified number of execution steps, as defined by an algorithm; because an infinite loop must not occur.
- Total validity: Whether the whole algorithm satisfies specifications or not.

Although various methods are used to verify the validity of an algorithm, it is difficult to verify it completely. If the range of data is specified, the validity can be verified easily by detecting trouble that may occur when data outside the specified range is inputted.

2.3.3 Evaluation by representation

Algorithms must be evaluated in terms of not only computational complexity and validity, but also algorithm representation, which is called the elaboration of algorithm representation.

Example

- The same step is repeatedly executed: If repeated steps are defined as a subroutine, they can be described as one single step so that the flow of steps in an algorithm can be presented in a simple, easy-to-understand manner.
- It is necessary to increase the speed of an algorithm: Attention should be paid to steps that are repeated most frequently, and ways to increase the speed should be studied (if quick sorting is used, it may be necessary to quit the use of recursive calls and to work out an alternative).

2.4 How to design algorithms

Based on data obtained by analyzing a problem and the results of evaluation described in the previous section, an algorithm is designed with the most attention given to how a solution can be had with the highest level of efficiency. There are several methods used to design an algorithm. Representative methods are:

- Dynamic programming method
- Greedy algorithm method
- Reduction method

(1) Dynamic programming method

In the dynamic programming method, one problem is considered to be a set of more than one sub-problem. Steps to follow are:

1. A problem is divided into some sub-problems.
2. A solution to each sub-problem is found.
3. Some sub-problems are put together to make a slightly larger partial problem.
4. Above steps 2 and 3 are repeated until a solution method to the original problem is found.

(2) Greedy algorithm method

In the greedy algorithm method, which is used to find a solution to an optimization problem, the values of variables are changed by degrees depending on each present condition. For example, to find how the number of coins can be decreased to a minimum to pay a specified amount, the numbers of large- to small-value coins are determined using the greedy algorithm method.

Example	To pay		\574,
	One 500-yen coin -	\74	remains
	One 50-yen coin -	\24	remains
	Two ten-yen coins -	\4	remains
	Four 1-yen coins - Nothing remains		

(3) Reduction method

In the reduction method, an algorithm originally designed has complexity of $O(n)$ is subjected to a reduction process of time length cn, so that it can be reduced to a smaller size to create a new algorithm of $O(n/x)$. If $O(n)>cn+O(n/x)$ cannot be satisfied, this algorithm is meaningless. Therefore, if n is very small, this algorithm cannot produce the expected results.

Exercises

Q1 **What is the term used to show a algorithm that searches the elements sequentially from the head to the foot of a table?**

a. Linear b. Binary c. Hash d. Heap

Q2 **Values in the array containing n elements, are sequentially compared with data X which is data to be searched. If data X matches a certain value in the table, "exist" is shown. Data X is stored in the position designated as index n+1.**

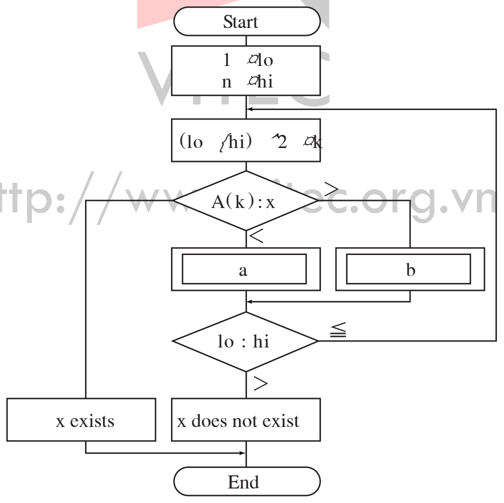
Subscript	1	2	3	...	i	...	n	n+1
Value	a_1	a_2	a_3	...	a_i	...	a_n	X

In the linear search algorithm shown below, what condition should be entered in the space $\boxed{}$?

Step 1 1 is set to index i.
Step 2 If $\boxed{}$, the routine jumps to step 5.
Step 3 1 is added to index i.
Step 4 The routine jumps to step 2.
Step 5 If index i is n or less, "exist" is shown.
Step 6 End

a. $i \geq n$ b. $i \neq n$ b. $i < n$ d. $X = a_i$ e. $X \neq a_i$

Q3 **There is array A which contain n data item sorted in the ascending order. The following flowchart shows the routine for retrieving data x from array A using the binary search method. Choose a correct combination of operations and enter them in the spaces a and b. Decimal numbers of a value obtained by division should be truncated.**



	a	b
a	$k+1 \rightarrow hi$	$k-1 \rightarrow lo$
b	$k-1 \rightarrow hi$	$k+1 \rightarrow lo$
c	$k+1 \rightarrow lo$	$k-1 \rightarrow hi$
d	$k-1 \rightarrow lo$	$k+1 \rightarrow hi$

Q4 **There is a table that has 2,000 different elements sorted in the ascending order of the keys. Using a key input from outside, this table is searched using the binary search method, and the elements that match the key are retrieved. What is the maximum number of comparison times needed before all matching elements are found? It is assumed that the table contains the matching key.**

a. 10 b. 11 c. 12 d. 13

Q5 **Which of the following comments made on search methods is wrong?**

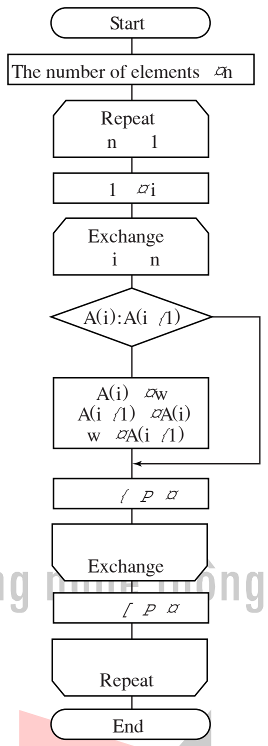
a. To use the binary search method, data must be sorted.
b. To search 100 data using the binary search method, the maximum number of comparison times that is needed to find the target data is 7.
c. If the linear search method is used, the number of comparison times does not necessarily decrease even if the data is already sorted.
d. If the number of data is 10 or smaller, the average number of comparison times that the linear search method requires, is smaller than that of the binary search method.
e. If the number of data is increased from 100 to 1,000, the number of comparison times increases to 10 times as large as when the linear search method is used. Using the binary search method, the number increases twice or less.

Q6 **Concerning data sort and merge, choose the proper combinations of words and enter them in the space $\boxed{}$.**

Sorting data in the order of small- to large-value queues is referred as $\boxed{\text{A}}$ $\boxed{\text{B}}$. If a target data sequence is in an auxiliary storage, this operation is called $\boxed{\text{C}}$.
Integrating two or more files $\boxed{\text{D}}$ in certain order into one file is called $\boxed{\text{E}}$.

	A	B	C	D	E
a	descending	sort	external sorting	sorted	merging
b	ascending	merge	external merge	merged	sorting
c	descending	merge	internal merge	merged	sorting
d	ascending	sort	external sorting	sorted	merging
e	ascending	merge	internal merge	merged	sorting

Q7 What sort of algorithm does the following flowchart show?



a. Quick sort b. Shaker sort c. Shell sort d. Insertion sort e. Bubble sort

Q8 It took 1.0 seconds for a certain computer to sort 1,000 data using the bubble sort method. How long will it take to sort 100,000 data of the same type? The time that it takes for a computer to bubble sort data is proportional to a square of the number of data that is defined as n.

a. 1 b. 10 c. 100 d. 1,000 e. 10,000

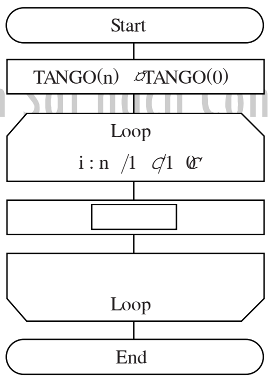
Q9 Concerning data sorting methods, which of descriptions given below is correct?

a. Quick sort is a method of sorting data in sub-sequences consisting of data items fetched at an interval and sorting smaller sub-sequences consisting of data items fetched at a smaller interval.
b. Shell sort is a method of sorting data by comparing a pair of adjacent elements and exchanging them if the second element is larger than the first.
c. Bubble sort is a method of sorting data by setting a median reference value, allocating elements with values larger than the reference value in one section and those with values smaller than the reference value in the other section and repeating this routine in each individual section.
d. Heap sort is a method of sorting data by representing an unsorted area as a subtree, fetching the maximum or minimum value from the unsorted area, moving the maximum or minimum value to a sorted area and repeating this routine to narrow down unsorted areas.

Q10 Which is an appropriate description of quick sort?

a. Compare and exchange are executed for two data away from one another at a certain distance. This distance is gradually and continuously narrowed to sort all data.
b. The first minimum value is found in data. The second minimum value is found in data in which the first minimum value is not included. This routine is repeatedly executed.
c. Data is divided into one group of data smaller than a reference value and the other group of data larger than a reference value. In each group, a new reference value is then selected and data is likewise divided into two groups based on the reference value. This routine is repeatedly executed.
d. Adjacent data are repeatedly compared and exchanged to allow smaller data to move to the end of a data array.

Q11 There is array TANGO whose index number starts with 0. n words are contained in TANGO (1) through TANGO (n). A flowchart below shows the steps for reorganizing the word table by shifting words in TANGO (1) through TANGO (n-1) backward, by one word, to put an n-th word in TANGO (1). Enter a proper step in the space $\boxed{}$.



Note: Names of variables ☐
(initial value, increment and termination value) ☐
are shown as loop conditions.

a. TANGO (i) → TANGO (i+1)
b. TANGO (i) → TANGO (n-i)
d. TANGO (i+1) → TANGO (n-i)
e. TANGO (n-i) → TANGO (i)

Q12 From the descriptions made on Newton's method, which is known as an algorithm for obtaining an approximate solution of the equation f(x) = 0, choose the most appropriate one.

a. Although a function, f(x), cannot be differentiated, an approximate solution can be obtained.
b. As seen from the geometrical viewpoint, the method is to obtain an approximate solution by using a tangential line of y = f(x).
c. Two different initial values must be provided.
d. Whatever initial values are provided, an approximate value can always be obtained.

Answers for No.3 Chapter2 (Algorithms)

Answer list

Answers

Q 1:	a	Q 2:	d	Q 3:	c	Q 4:	b	Q 5:	d
Q 6:	d	Q 7:	e	Q 8:	e	Q 9:	d	Q 10:	c
Q 11:	a	Q 12:	b						

Answers and Descriptions

Q1

Answer

a. Linear

Description

a. Linear

The linear search algorithm searches the elements sequentially from the beginning to the end of a table → answer

b. Binary

A binary search method is the method of narrowing down target data while successively dividing a search area into two parts. Elements are not searched sequentially.

c. Hash

A hash is a way of using an array-type data structure. Using a hash, you can directly access specific data using a key without accessing recorded data one by one.

d. Heap

A heap is a kind of binary search tree. It is a perfect binary tree that has a certain size relationship between a parent node and a child node. A heap is different from a binary search tree in that a heap does not have a certain size relationship between brothers.

Q2

Answer

d. $X = a_i$

Description

Subscript	1	2	3	...	i	...	n	n+1
Value	a_1	a_2	a_3	...	a_i	...	a_n	X

The step2 through step4 form a loop.

The space $\boxed{}$ represents the loop condition.

The condition for exit is either “the value is found” or “at-end is reached”.

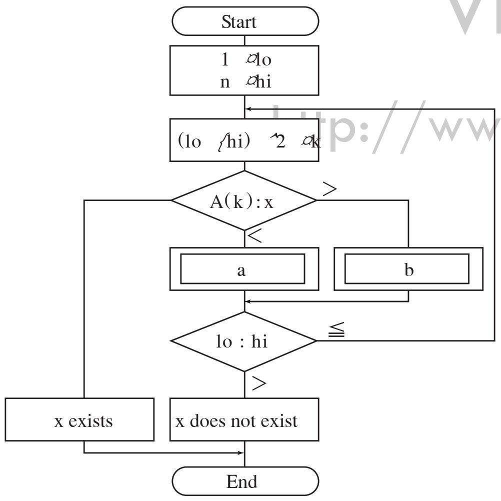
Since “whether at end or not” is examined in the step 5, the space should be “the value is found”, i.e. X matches the current element value. → the answer is d

Q3

Answer

	a	b
c	$k+1 \rightarrow lo$	$k-1 \rightarrow hi$

Description



A binary search method is the method of narrowing down target data while successively dividing a search area into two parts.

1) space a

If A(k) < X, further divide the higher half area into two parts, therefore “k+1 → lo”.

2) space b

If A(k) > X, further divide the lower half area into two parts, therefore k-1 → hi

Therefore, the answer is c.

Q4

Answer

b. 11

Description

In binary search, the maximum and average number of times a comparison is made: If the number of elements is N,

-Average number of times is $[\log_2 N]$

-Maximum number of times is $[\log_2 N] + 1$

([ ] is a Gaussian symbol and decimal numbers of the value shown in this symbol are truncated.)

Average number of times "k" is

$1 <= 2000/2^k < 2$

$2^k <= 2000/2^k < 2^{k+1}$

Here,

$\log_2 1024 = \log_2(2)^{10} = 10 <= \log_2 2000 < \log_2 2048 = \log_2(2)^{11} = 11$

Therefore k = 10 and the Maximum number of times is k+1 = 11

→ the answer is b.

Q5

Answer

d. If the number of data is 10 or smaller, the average number of comparison times that the linear search method requires, is smaller than that of the binary search method.

Description

In binary search, the maximum and number of times a comparison is made: If the number of elements is N,

-Average number of times is $[\log_2 N]$

-Maximum number of times is $[\log_2 N] + 1$

In linear search, the maximum number of times a comparison is made: If the number of elements is N,

-Average number of times is N/2

-Maximum number of times is N

a. To use the binary search method, data must be sorted.

a is correct.

b. To search 100 data using the binary search method, the maximum number of comparison times that is needed to find the target data is 7.

if N=100, Average number of times for binary search is

$\log_2 100 < \log_2 128 = \log_2 (2)^7$

Therefore Average is 6 and Maximum is 7 → b is also correct.

c. If the linear search method is used, the number of comparison times does not necessarily decrease even if the data is already sorted.

C is also correct.

d. If the number of data is 10 or smaller, the average number of comparison times that the linear search method requires, is smaller than that of the binary search method.

if N=10,

Average number of times for linear search is 10/2 = 5

Average number of times for binary search is $\log_2 10 < \log_2 (2)^4 = 4$

Therefore d is incorrect → Answer

e. If the number of data is increased from 100 to 1,000, the number of comparison times increases to 10 times as large as when the linear search method is used. Using the binary search method, the number increases twice or less.

Average number of times for binary search for N=100 and N=1000 is as follows. Therefore e. is also correct.

If N=100, $\log_2 100 < \log_2 128 = \log_2 (2)^7$

If N=1000, $\log_2 1000 < \log_2 1024 = \log_2 (2)^{10}$

Q6

Answer

	A	B	C	D	E
d	ascending	sort	external sorting	sorted	merging

Description

1) A, B

"Arranging data in the order of small- to large-value queues" means "ascending sort".

2) C

If a target data sequence is in an auxiliary storage, this operation is called “external sorting”. c.f. Internal sorting means sorting data in a main memory unit. External sorting means sorting data stored on a magnetic disk and other auxiliary storage unit.

3) D,E

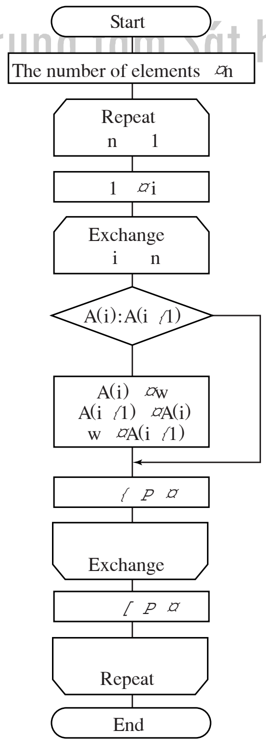
Integrating two or more files “sorted” in certain order into one file is called “merging.”

Q7

Answer

e. Bubble sort

Description



In the above flowchart, A(i) and A(i+1) are compared, and if A(i) > A(i+1), they are swapped. i.e. a pair of elements next to each other in the sequence is compared and if sequence is wrong, swap occurs.

This describes the bubble sort. → the answer is e.

Q8

Answer

e. 10,000

Description

In the bubble sort mechanism, a pair of elements next to each other in the sequence is compared and if sequence is wrong, swap occurs.

Its computational complexity is n^2 (proportional to a square of n, n: number of data)

If number of data becomes 100 times bigger than earlier (previously 1000 and now 100,000), time for computation will be $(100)^2$ = 10,000 times longer than earlier.

→ the answer is e. 10,000.

Q9

Answer

d. Heap sort is a method of sorting data by representing an unsorted area as a subtree, fetching the maximum or minimum value from the unsorted area, moving the maximum or minimum value to a sorted area and repeating this routine to narrow down unsorted areas.

Description

a. Quick sort is a method of sorting data in sub-sequences consisting of data items fetched at an interval and sorting smaller sub-sequences consisting of data items fetched at a smaller interval.

This describes “Shell sort method”, in which two items of data located away from each other at a certain interval are picked out of a data sequence.

b. Shell sort is a method of sorting data by comparing a pair of adjacent elements and exchanging them if the second element is larger than the first.

This describes “Bubble sort method” where comparison of a pair of adjacent elements takes place.

c. Bubble sort is a method of sorting data by setting a median reference value, allocating elements with values larger than the reference value in one section and those with values smaller than the reference value in the other section and repeating this routine in each individual section.

This describes “Quick sort method”.

d. Heap sort is a method of sorting data by representing an unsorted area as a subtree, fetching the maximum or minimum value from the unsorted area, moving the maximum or minimum value to a sorted area and repeating this routine to narrow down unsorted areas.

This is a correct description of the Heap sort method. → answer

Q10

Answer

c. Data is divided into one group of data smaller than a <u>reference value</u> and the other group of data larger than a reference value. In each group, a new reference value is then selected and data is likewise divided into two groups based on the reference value. This routine is repeatedly executed.

Description

In this question, an appropriate description of quick sort is to be found.

The quick sort method was designed by Hoare. It is presently the fastest sort method using the recursive call method.

A reference value (pivot or pivotal value) is chosen from the data to be sorted. A median value or an intermediate value of three elements (right, center and left elements) is usually used. Then Data items smaller than the pivot are moved to the left of the pivot while data items larger than the pivot are moved to the right of it. All the data items are thus divided into two sections (division). A pivot is chosen from each section of data items so that the section is further divided into two sections.

This dividing operation is repeatedly performed until only one element remains.

→ c. is an appropriate description of the quick sort method → answer

a. Compare and exchange are executed for two data away from one another at a certain distance. This distance is gradually and continuously narrowed to sort all data.

a. describes the shell sort method.

b. The first minimum value is found in data. The second minimum value is found in data in which the first minimum value is not included. This routine is repeatedly executed.

b. describes the basic selection method.

d. Adjacent data are repeatedly compared and exchanged to allow smaller data to move to the end of a data array.

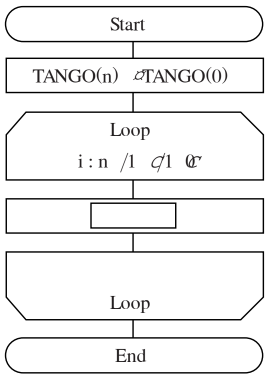
d. describes the bubble sort method.

Q11

Answer

a. TANGO (i) → TANGO (i+1)

Description



Note: Names of variables □
(initial value, increment and termination value) □
are shown as loop conditions.

This algorithm consists of

1) TANGO(n) → TANGO(0)

2) A loop, index "i" value changing from n-1 to 0 decrementing by 1

Take a look at an example of applying the above.

Assume that n=5 and the array named "TANGO" is initially as follows. i.e. the words "FE", "SW", "JITEC", "JIPDEC" and "METI" are in stored in TANGO(1), TANGO(2), TANGO(3), TANGO(4) and TANGO(5) respectively.

	FE	SW	JITEC	JIPDEC	METI
TANGO(0)	TANGO(1)	TANGO(2)	TANGO(3)	TANGO(4)	TANGO(5)

After applying this algorithm, the word in TANGO(n) is supposed to be stored in TANGO(1), then the remaining words are supposed to be shifted to right.

METI	METI	FE	SW	JITEC	JIPDEC
TANGO(0)	TANGO(1)	TANGO(2)	TANGO(3)	TANGO(4)	TANGO(5)

After doing "TANGO(5)→TANGO(0)", the following should be done in the loop (whose index "i" value changing from n-1 to 0 decrementing by 1)

TANGO(4) → TANGO(5)

TANGO(3) → TANGO(4)

:

TANGO(0) → TANGO(1)

This can be described using index "I" as TANGO(i) → TANGO(i+1).

Therefore the answer is a.

b. TANGO (i) → TANGO (n-i)

c. TANGO (i+1) → TANGO (n-i)
d. TANGO (n-i) → TANGO (i)

b,c,d are incorrect.

Q12

Answer

b. As seen from the geometrical viewpoint, the method is to obtain an approximate solution by using a tangential line of y = f(x).

Description

Algorithm of Newton's method is described as follows. As shown below, It obtains a solution by using a tangential line y=f(x). Therefore the answer is b.

Step 1
A tangential line y = f(x) at point p1 of the coordinate x1, y1 is drawn and point x2 where the tangential line intersects x-axis is obtained.
Step 2
A tangential line y = f(x) at point p2 of the coordinate x2, y2 is likewise drawn and point x1 where the tangential line intersects x-axis is obtained. As this step is repeatedly executed the tangential line moves closer to a solution.
Step 3
The difference between adjacent approximate values obtained in step 2 is compared with a predetermined convergence judgment value precision. Steps 1 and 2 are repeatedly executed until this difference becomes smaller than the predetermined convergence judgment value.

The equation for a tangential line at point p1 is y-f(x_1) = f'(x_1)(x-x_1), point x_2 where a tangential line intersects x-axis can be obtained using the following equation:

$$x_{i+1} = x_1 - \frac{f(x_i)}{f'(x_i)} \quad (i = 0, 1, 2, ...)$$

a. Although a function, f(x), cannot be differentiated, an approximate solution can be obtained.

This is incorrect because this method use differential as explained above.

c. Two different initial values must be provided.

This is also incorrect because one solution can be obtained from one initial value.

d. Whatever initial values are provided, an approximate value can always be obtained.

In case f'(x_n