Sethu Institute of Technology
An Autonomous Institution
Accredited with 'A' grade by NAAC
(Affiliated to Anna University, Chennai and Approved by AICTE - New Delhi)
(Recognized by UGC under section 2(f) & 12 (B) of UGC Act, 1956)

19UCB206 - INTRODUCTION TO DATA STRUCTURES & ALGORITHMS (R-2019)

Mrs.P.Suganthi, Assistant Professor(SG)

DEPARTMENT OF COMPUTER SCIENCE AND BUSNIESS SYSTEMS

UNIT I BASIC TERMINOLOGIES &INTRODUCTION TO ALGORITHM AND DATA ORGANISATION

Algorithm specification, Recursion, Performance analysis, Asymptotic Notation - The Big-O, Omega and Theta notation, Programming Style, Refinement of Coding - Time-Space Trade Off, Testing, Data Abstraction

DATA STRUCTURES

What is the "Data Structure" ?

- ***Ways to represent data***.
- It is the representation of the ***logical relationship existing between individual elements of data.***
- It is a specialized format for ***organizing and sorting data*** in memory that considers not only the elements sorted but also their relationship to each other.

Why data structure ?

- To design and ***implement large-scale computer system***
- ***Have proven correct algorithms***
- The art of programming

How to master in data structure ?

- ***practice, discuss, and think***

ALGORITHM SPECIFICATION

Definition

An ***algorithm*** is a ***finite set of instructions*** that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

(1)***Input***. There are ***zero or more quantities*** that are externally supplied.

(2)***Output***. ***At least one quantity*** is produced.

(3)***Definiteness***. Each instruction is ***clear and unambiguous***.

(4)***Finiteness***. If we trace out the instructions of an algorithm, then for all cases, the algorithm ***finish or terminates after a finite number of steps.***

(5)***Effectiveness***. All operations to be accomplished must be sufficiently basic that they can be ***done exactly and in finite length***

DESCRIBING ALGORITHMS

Natural language

English, Chinese

Instructions must be ***definite and effectiveness***

Graphic representation

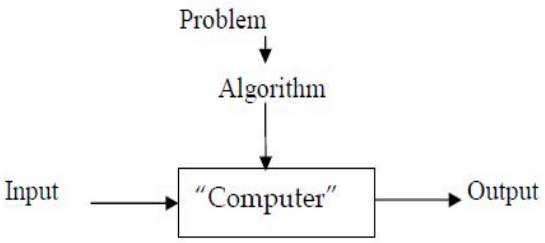
Flowchart

work well only if the algorithm is ***small and simple***

Pseudo language

Readable

Instructions must be ***definite and effectiveness***



TRANSLATING A PROBLEM INTO AN ALGORITHM

Problem

Devise a program that sorts a set of *n>= 1* integers

Step I – Concept

From those integers that are currently unsorted, find the smallest and place it next in the sorted list

Step II – Algorithm

for (i= 0; i< n; i++){
Examine *list[i]* to *list[n-1]* and suppose that the smallest integer is *list[min]*;
Interchange *list[i]* and *list[min]*;
}

TRANSLATING A PROBLEM INTO AN ALGORITHM(CONT.)

Step III - Coding

```
void sort(int *a, int n)
{
   for (i= 0; i< n; i++)
    {
    int j= i;
    for (int k= i+1; k< n; k++){
        if (a[k ]< a[ j]) j= k;
    int temp=a[i]; a[i]=a[ j]; a[ j]=temp;
   }
}
```

RECURSIVE ALGORITHMS

In ***recursion, a function or method has the ability of calling itself*** to solve the problem.

Types

Direct recursion

Functions ***call themselves***

Indirect recursion

Functions ***call other functions that invoke the calling function again***

When is recursion an appropriate mechanism?

The problem itself is defined recursively

Statements: ***if-else and while*** can be written recursively

Why recursive algorithms ?

which you can ***use recursion or iteration***

Powerful, express an complex process very clearly

PERFORMANCE ANALYSIS

Performance analysis - prior

- an important branch of CS, ***complexity theory***
- estimate ***time and space***
- machine ***independent***

Performance measurement -posterior

- The ***actual time*** and ***space requirements***
- machine ***dependent***

Space and time

- Does the ***program efficiently use primary and secondary storage***?
- Is the ***program's running time acceptable for the task?***

PERFORMANCE ANALYSIS

Evaluate a program generally

- Does the program ***meet*** the original ***specifications*** of the task?
- Does it ***work correctly***?
- Does the program contain ***documentation*** that show ***how to use it and how it works?***
- Does the program ***effectively use functions*** to create logical units?
- Is the program's code ***readable?***

Evaluate a program

MWGWRERE

- **M**eet specifications, **W**ork correctly,
  Good user-interface, **W**ell-documentation,
  Readable, **E**ffectively use functions,
  Running time acceptable, **E**fficiently use space

How to achieve them?

- ***Good programming style, experience, and practice***
- ***Discuss and think***

SPACE COMPLEXITY

Definition

- ***Space complexity*** is the ***amount of space required*** to solve an algorithm.
- The ***space complexity*** of a program ***is the amount of memory that it needs to run to completion.***

The space needed is the ***sum of Fixed space and Variable space***

Fixed space

- Includes the ***instructions, variables, and constants***
- Independent of the ***number and size of I/O***

Variable space

- Includes ***dynamic allocation, functions' recursion***

Total space of any program

- ***S(P)= c+ Sp(Instance)***

c is a fixed space *, which is independent of input and output . It is a constant.*

where ***sp(Instance) is variable space*** *and it depends on Instance.*

TIME COMPLEXITY

- Definition
  - ***Time complexity*** is the ***amount of time taken by an algorithm to execute a task.***
  - The ***time complexity, T(p),*** taken by a ***program P is the sum of the compile time and the run time***

- Total time
  - T(P) = compile time + ***run*** (or ***execution***) ***time***
    = ***c*** + ***t****p(instance characteristics)*
    Compile time does not depend on the instance characteristics

- How to evaluate?
  - Use the ***system clock***
  - Number of ***steps*** performed
    - ***machine-independent***

- Definition of a program step
  - ***A program step*** is a ***syntactically or semantically meaningful program segment*** whose ***execution time is independent*** of the ***instance characteristics***
    (10 additions can be one step, 100 multiplications can also be one step)

EXAMPLES OF DETERMINING STEPS

*the **first method: count by a program***

```
float rsum(float list[], int n)
{
  count ++;  /* for if condition  */
  if (n) {
    count++; /* for return and rsum invocation */
    return rsum(list, n-1)+ list[n-1];
  }
  count++; //return
  return list[0];
}
```

~~2n+ 2~~

rsum(0) = 2
rsum(n) = 2 + rsum(n-1)
= 2 + 2 + rsum(n-2)
= 2 + 2 + rsum(n-3)
= …
= 2n + rsum(0)= 2n+2

```
void add(int a[][MaxSize], int b[][MaxSize],
                int c[][MaxSize], int rows, int cols)
{
  int i, j;
  for (i=0; i< rows; i++)
    for (j=0; j< cols; j++)
      c[i][j]= a[i][j] + b[i][j];
  }
```

~~2rows*cols+ 2rows+ 1~~

EXAMPLES OF DETERMINING STEPS

- The ***second method: build a table to count***

Statement	*s/e*	*Frequency*	*Total Steps*
void add(int a[][MaxSize], . . .	*0*	*0*	*0*
{	*0*	*0*	*0*
int i, j;	*0*	*0*	*0*
for (i=0; i< rows; i++)	*1*	*rows+ 1*	*rows+ 1*
for (j=0; j< cols; j++)	*1*	*rows*(cols+1)*	*rows*cols+ rows*
c[i][j]= a[i][j] + b[i][j];	*1*	*rows*cols*	*rows*cols*
}	*0*	*0*	*0*
Total			***2rows*cols+2rows+1***

REMARKS OF TIME COMPLEXITY

- Difficulty: the time complexity is ***not dependent*** solely on the ***number of inputs or outputs***
- To determine the step count
  - **Best case, Worst case,** and **Average**

Best Case:
The ***amount of time*** the algorithm takes on ***best set of inputs.***

Worst Case:
The ***amount of time*** the algorithm takes on the ***worst possible set of inputs***.

Average Case:
The ***amount of time*** the algorithm takes on an ***average set of inputs.***

DATA ABSTRACTION

Types of data

- All ***programming language provide*** at least ***minimal set of predefined data type, plus user defined types***

Data types of C

- Char, int, float, and double
  - may be modified by short, long, and unsigned
- Array, struct, and pointer

DATA TYPE

Definition

- A ***data type*** is a ***collection of objects*** and a ***set of operations*** that act on those objects

Example of "int"

- Objects: 0, +1, -1, ..., Int_Max, Int_Min
- Operations: *arithmetic*(+, -, *, /, and %), *testing*(equality/inequality), *assigns*, *functions*

Define operations

- Its ***name***, possible ***arguments and results*** must be ***specified***

The design strategy for representation of objects

- ***Transparent*** to the user

ABSTRACT DATA TYPE

Definition

- An ***abstract data type(ADT) is a data type*** that is organized in such a way that the ***specification of the objects*** and the ***specification of the operations on the objects is separated*** from the representation of the objects and the implementation of the operation.

Why abstract data type ?

- implementation-independent

CLASSIFYING THE FUNCTIONS OF A DATA TYPE

Creator/constructor:

- Create a ***new instance*** of the designated type.

Transformers

- Also ***create an instance*** of the designated type by using ***one or more other instances.***

Observers/reporters

- Provide information about an instance of the type, but they ***do not change the instance***

Notes

- An ***ADT definition will include at least one function*** from each of these three categories