

INTRODUCTION TO ALGORITHMS

By Dr. N. Subhash Chandra

Fundamentals of Algorithm Analysis


CONTENTS

- 1) Introduction
- 2) Definition of algorithm
- 3) Algorithmic Problem solving
- 4) Methods of Specifying an Algorithm
- 5) Steps for writing an algorithm(Pseudo code)
- 6) Sample Problems and Algorithms
- 7) Quiz
- 8) Important Questions

Algorithm Background

- ***Algorithm:*** The word algorithm came from the name of a Persian mathematician Abu Jafar Mohammed Ibn Musa Al Khowarizmi (ninth century).
- An algorithm is simply a set of rules used to perform some calculations either by hand or more usually on a machine (computer).

Definition of Algorithm



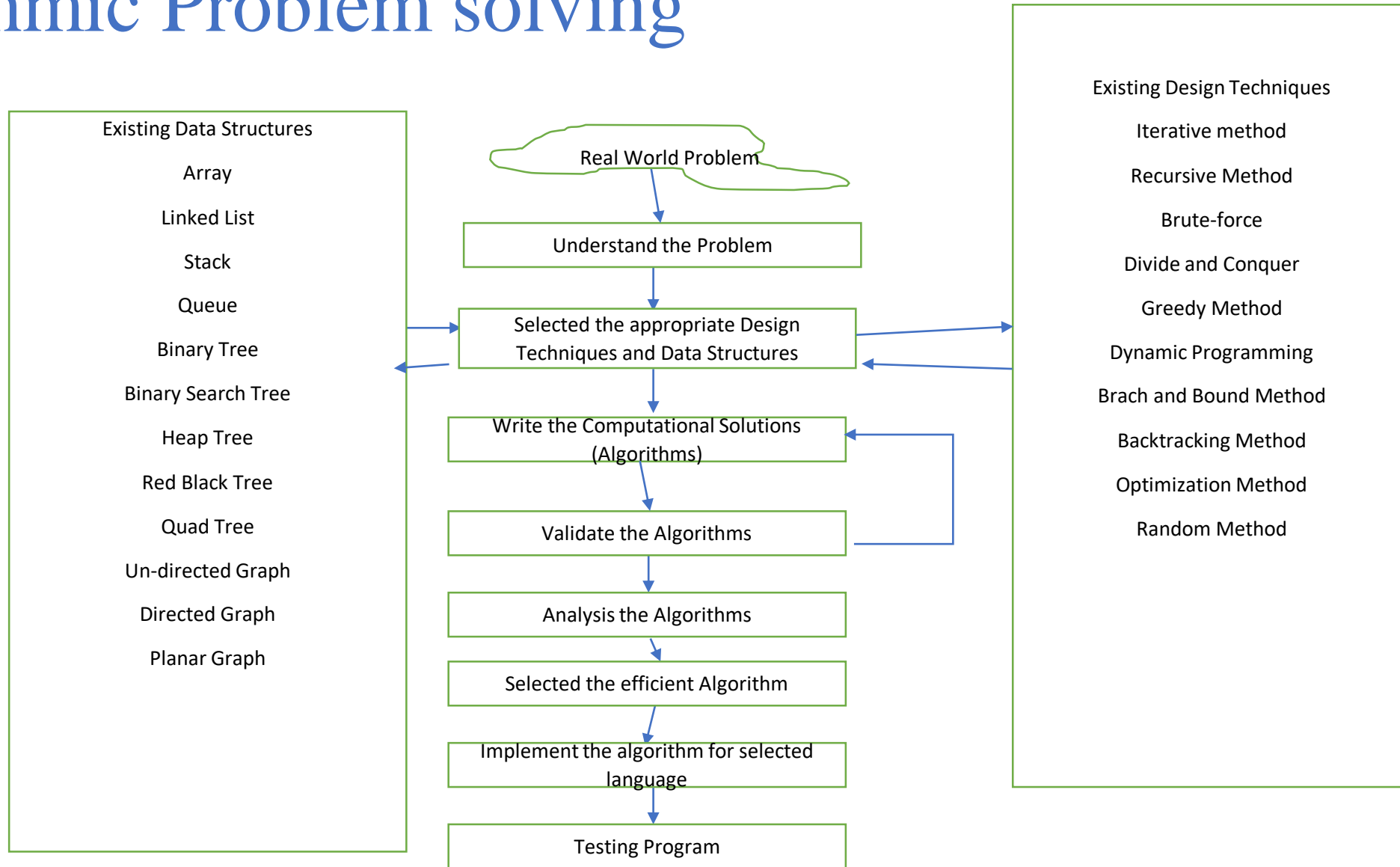
Definition:

An *algorithm* is a finite sequence of unambiguous instructions for solving a particular problem. It must satisfy the specific characteristics.

Characteristics of an algorithm.

- **Input** : Zero / more quantities are externally supplied.
- **Output**: At least one quantity is produced.
- **Definiteness**: Each instruction is clear and unambiguous.
- **Finiteness**: If the instructions of an algorithm is traced then for all cases the algorithm must terminates after a finite number of steps.
- **Efficiency**: Every instruction must be very basic and runs in short time with effective results better than human computations.

Algorithmic Problem solving



Issues for Algorithm

- There are various issues in the study of algorithms;
 1. How to devise algorithms: The creation of an algorithm is a logical activity which may never be fully automated.
 2. How to express algorithms: We shall express all of our algorithms using the best principles of structuring.
 3. How to validate algorithms: After creation of algorithms is to validate algorithms. The process of checking an algorithm computes the correct answer for all possible legal inputs is called algorithm validation. The purpose of validation of algorithm is to find whether algorithm works properly without being dependent upon programming languages.
 4. How to analyze algorithms: Analysis of algorithm is a task of determining how much computing time and storage is required by an algorithm. Analysis of algorithms is also called performance analysis. The behavior of algorithm in best case, worst case and average case needs to be obtained.
 5. How to test a program: Testing a program really consists of two phases:
 - i) *Debugging*: While debugging a program, it is checked whether program produces faulty results for valid set of input and if it is found then the program has to be corrected.
 - ii) *Profiling or performance measuring*: Profiling is a process of measuring time and space required by a corrected program for valid set of inputs.

How to devise algorithms?

Creating an algorithm is an art which may never be fully automated. In this research we want to study more design techniques and select appropriate design techniques to devise the new useful algorithm. The most important design techniques are

- (a) Brute-force or exhaustive search
- (b) Divide and Conquer
- (c) Greedy Algorithms
- (d) Dynamic Programming
- (e) Branch and Bound Algorithm
- (f) Randomized Algorithm
- (g) Backtracking

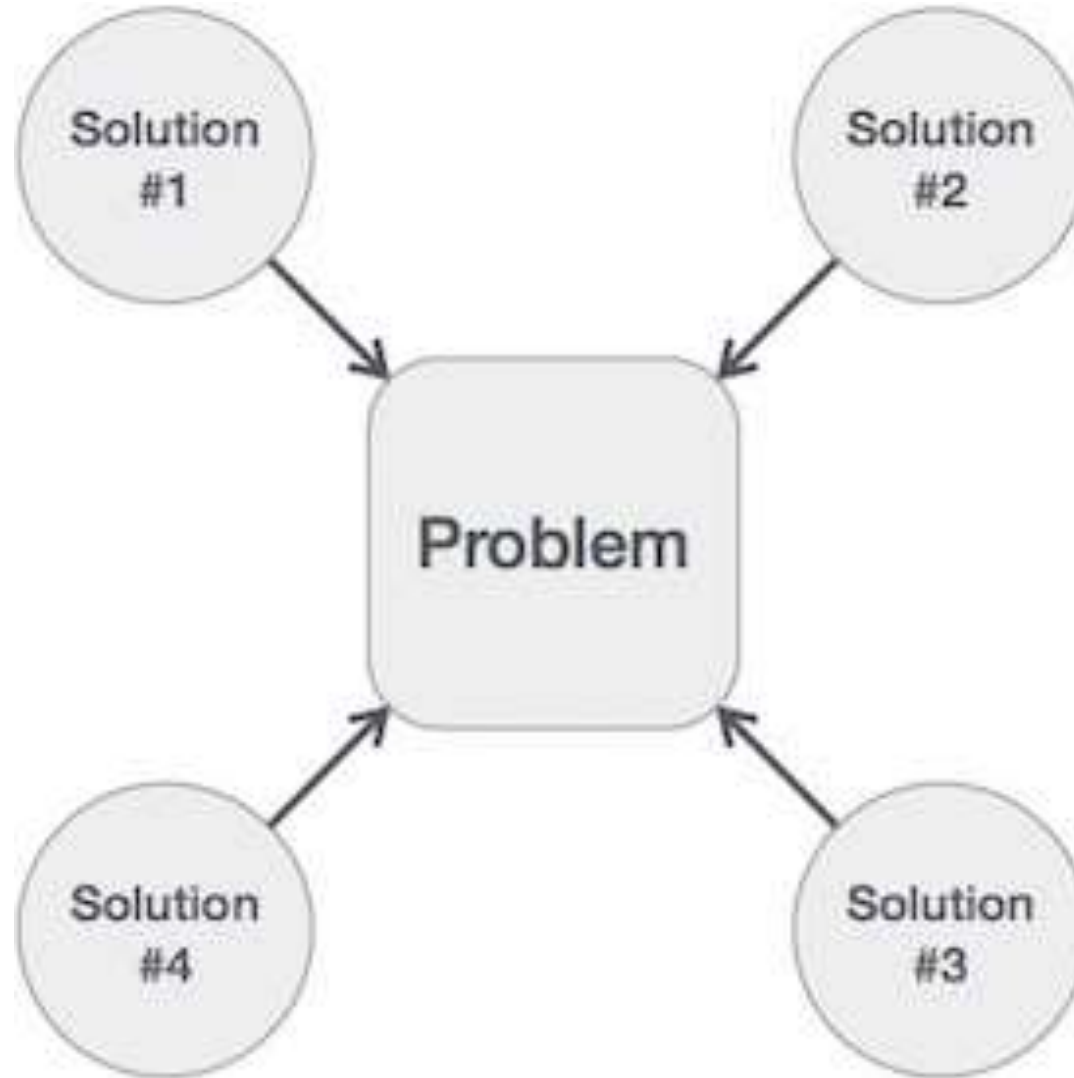
How to validate algorithms?

- ❖ Once an algorithm is devised, it is necessary to show that it computes the correct answer for all possible legal inputs.
- ❖ It helps to the next process for writing program code in a particular programming language.
- ❖ This phase is also called program proving or program verification.
- ❖ The proof of correctness requires that the solution be stated in two forms.
- ❖ First form, In this form a program which is **annotated** by a set of **assertations** about the input and output variables of the program. Then these assertions are often expressed in the **predicate calculus**.
- ❖ Second form is called specification, and this expressed in the **predicate calculus**.

How to analysis algorithms?

- Efficiency of an algorithm can be analyzed at two different stages, before and after implementation.
- Before implementation(priori analysis), analysis of algorithms refers to the task of determining how much computing time and storage space an algorithm require.
- This is the challenging area which require great mathematical skill.(Quantitative judgement about the algorithms)
- An algorithm perform in the best case, the worst case or on the average case are typical.

Algorithm Solutions for a Given problem



How to test a program?

After implementation (posterior analysis), Testing a program consists of two phases:

- (a) Debugging
- (b) Profiling

Debugging is the process of executing programs on sample data sets to determine whether the faulty results occur if so to correct them.

Profiling or performance measurement is the process of executing a correct program on data sets and measuring time and space it takes to compute the results.

4. Methods of Specifying an Algorithm

There are three ways to specify an algorithm. They are:

- a. Natural language
- b. Pseudocode
- c. Flowchart

Pseudocode and flowchart are the two options that are most widely used nowadays for specifying algorithms.

Natural Language

It is very simple and easy to specify an algorithm using natural language. But many times, specification of algorithm by using natural language is not clear and thereby we get brief specification.

Example: An algorithm to perform addition of two numbers.

Step 1: Read the first number, say a.

Step 2: Read the first number, say b.

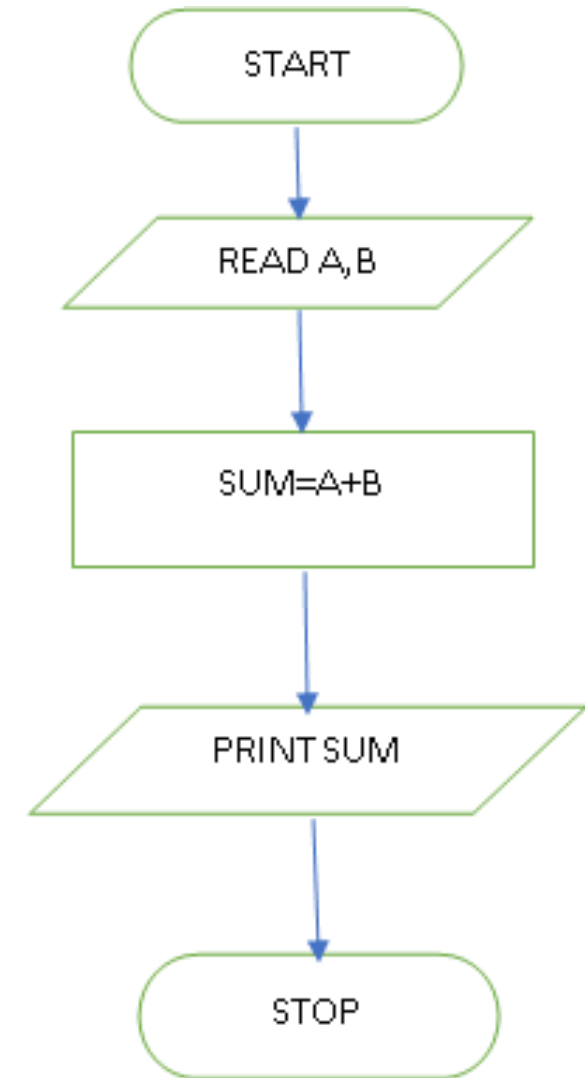
Step 3: Add the above two numbers and store the result in c.

Step 4: Display the result from c.

Such a specification creates difficulty while implementing it. Hence many programmers prefer specification of algorithm by means of Pseudocode.

Flowchart

In the earlier days of computing, the dominant method for specifying algorithms was a *flowchart*, this representation technique has proved to be inconvenient. *Flowchart* is a graphical representation of an algorithm. It is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.



Pseudocode

Pseudocode is a mixture of a natural language and programming language constructs. Pseudocode is usually more precise than natural language.

For Assignment operation left arrow “:=”, for comments two slashes “//”,if condition, for, while loops are used.

```
ALGORITHM Sum(a, b)
{
    //Problem Description: This algorithm performs addition of two numbers
    //Input: Two integers a and b
    //Output: Addition of two integers
    c:=a+b;
    return c;
}
```

This specification is more useful for implementation of any language.

Steps for writing an algorithm(Pseudo code):

1. An algorithm is a procedure. It has two parts; the first part is head and the second part is body.
2. The Head section consists of keyword Algorithm and Name of the algorithm with parameter list.

E.g. Algorithm name1(p1, p2,...,pn)


The head section also has the following:

//Problem Description:

//Input:

//Output:

3. In the body of an algorithm various programming constructs like if, for, while and some statements like assignments are used.
4. The compound statements may be enclosed with { and } brackets. if, for, while can be open and closed by {, } respectively. Proper indentation is must for block.
5. Comments are written using // at the beginning.

- 
6. The identifier should begin by a letter and not by digit. It contains alpha numeric letters after first letter. No need to mention data types.
 7. The left arrow “:=” used as assignment operator. E.g. v:=10
 8. Boolean operators (TRUE, FALSE), Logical operators (AND, OR, NOT) and Relational operators (<, <=, >, >=, =, ≠, <>) are also used.
 9. Input and Output can be done using read and write.
 10. Array[], if then else condition, branch and loop can be also used in algorithm.

Operators and Conditional Statement(Pseudo code):

There are two Boolean values TRUE and FALSE. Logical **operators**: AND, OR, NOT.

- Relational operators: <, >, ≥, ≤, =, ≠. Arithmetic operators: +, -, *, /, %;

The conditional statement if-then or if-then-else is written in the following form. If (condition) then (statement)

- If (condition) then {Block-1 }else { Block-2}
- ‘If’ is a powerful statement used to make decisions based as a condition. If a condition is true the particular block of statements are execute.

Example

```
if(a>b) then
{
    write("a is big");
}
else
{
    write("b is big");
}
```

Case Statement(Pseudo code):

Case statement

```
case
{
  :(condition -1): (statement-1)
  :(condition -2): (statement-2)
  :(condition -n): (statement-n)
  .....
  .....
  else      :(statement n+1);
}
```

If condition -1 is true, statement -1 executed and the case statement is exited. If statement -1 is false, condition -2 is evaluated. If condition -2 is true, statement-2 executed and so on. If none of the conditions are true, statement $-(n+1)$ is executed and the case statement is exited. The else clause is optional.

for loop statement(Pseudo code):

Loop statements: For loop:

The general form of the for loop is

```
for variable:=value1 to value2 step value3 do  
{  
    Statement -1;  
    Statement -2;  
    ...  
    ...  
    Statement -n;  
}
```

Example 1:

```
for i:=1 to 10 do  
{  
    write(i); //displaying numbers from 1 to 10 i:=i+1;  
}
```

Example 2:

```
for i:=1 to 10 step 2 do  
{  
    write(i); //displaying numbers from 1 to 10 i:=i+2;  
}
```

While Statement(Pseudo code):

while loop:

The general form of while is-

while (condition) do

{

statement 1;

statement 2;

...

...

statement n;

}

Example

i:=1;

while(i<=n) do

{

write (i);

i:=i+1;

}

repeat-until statement(Pseudo code):

Repeat-until loop:

The general form of repeat-until is-

```
repeat  
{  
    statement 1;  
    statement 2;  
    ...  
    ...  
    statement n;  
}until (condition);
```

Example

```
i:=1;  
repeat  
{  
    write (i);  
    i:=i+1;  
}  
until (i>10);
```

Break, Array and function Statement(Pseudo code):

Break: this statement is exit from the loop.

Elements of array are accessed using [].

- For example, if A is an one-dimensional array, then i^{th} element can be accessed using A[i].
- If A is two-dimensional array, then $(i, j)^{\text{th}}$ element can be accessed using A[i,j].

Procedures (functions): There is only one type of procedure:

An algorithm consists of a heading and a body.

SAMPLE ALGORITHMS

Write an algorithm to find maximum element in an array with size 'n'.

Algorithm **Max**(A, n)

// This algorithm return the maximum element in the given array A with size n

```
{  
    maximum=A[1];  
    for i=2 to n do  
        if A[i]>maximum then  
            maximum=A[i];  
    return maximum;  
}
```



Microsoft Excel
37-2003 Worksheet

VERIFICATION

[illegible]

Write a linear sort algorithm to sort the data in an array with size 'n'.

Ans:

Algorithm LinearSort(*A*, *n*)

// This algorithm return the sorted(ascending order) array for the given array A with size n

```
{
  for i=1 to n-1 do{
    for j=i+1 to n do
      if A[i]>A[j] then
        {
          temp=A[i];
          A[i]=A[j];
          A[j]=temp;
        }
      }
  }
  return A;
}
```

VALIDATE THE ALGORITHM

VERIFY THE ALGORITHM

PASS 1: Fix the first minimum element in the first position

INDEX

A

I=1, J=2

I=1,J=3

I=1,J=4

I=1,J=5

1

23

12

12

7

7

2

12

23

23

23

23

3

18

18

18

18

18

4

7

7

7

12

12

5

20

20

20

20

20

RULE 1 : IF $A[I] > A[J]$ THEN INTERCHANGE($A[I]$, $A[J]$) ELSE NO-ACTION

3. Write an algorithm for finding max and min in an array of n elements.

Algorithm **MaxMin**(A, n)

// This algorithm return the max and min values in a given array A, n- size of the array.

```
{
    maximum:=A[1];
    minimum:=A[1];
    for i:=2 to n do
    {
        if A[i]<maximum then maximum:=A[i];
        if A[i]>minimum then minimum:=A[i];
    }
    return([minimum, maximum]);
}
```

4. Write an algorithm for Bubble Sort an array of n elements.

Algorithm **BubbleSort**(A, n)

// This algorithm return the sorted(ascending order) array for the given array A with size n

```
{  
    for i=1 to n-1 do{  
        for j=1 to n-i do  
            if A[j]>A[j+1] then {  
                temp=A[j];  
                A[j]=A[j+1];  
                A[j+1]=temp;  
            }  
    }  
}
```

BUBBLE SORT									
VERIFY THE ALGORITHM									
PASS 1: Fix the largest element at last position									
INDEX	KEY-A	I=1	I=2	I=3	I=4				
1	23	12	12	12	12				
2	12	23	18	18	18				
3	18	18	23	7	7				
4	7	7	7	23	20				
5	20	20	20	20	23				
RULE 1 : IF A[i]>A[i+1] THEN INTERCHANGE(A[I], A[J]) ELSE NO-ACTION									
PASS 2: Fix the second largest data in its position									
INDEX	KEY-A	I=1	I=2	I=3					
1	12	12	12	12					
2	18	18	7	7					
3	7	7	18	18					
4	20	20	20	20					
5	23	23	23	23					
RULE 1 : IF A[i]>A[i+1] THEN INTERCHANGE(A[I], A[J]) ELSE NO-ACTION									

PASS 3: Fix the third largest data in its position									
INDEX	KEY-A	I=1	I=2						
1	12	7	7						
2	7	12	12						
3	18	18	18						
4	20	20	20						
5	23	23	23						
RULE 1 : IF A[i]>A[i+1] THEN INTERCHANGE(A[I], A[J]) ELSE NO-ACTION									

PASS 4: Fix the fourth largest data in its position									
INDEX	KEY-A	I=1							
1	7	7							
2	12	12							
3	18	18							
4	20	20							
5	23	23							
RULE 1 : IF A[i]>A[i+1] THEN INTERCHANGE(A[I], A[J]) ELSE NO-ACTION									

5. Write an algorithm for Selection Sort an array of n elements.

Algorithm **SelectionSort**(A, n)

// This algorithm return the sorted(ascending order) array for the given array A with size n

```
{  
    for i:=1 to n-1 do{  
        pos:=i;  
        for j:=i+1 to n do{  
            if A[j]<A[pos] then pos:=j;  
        }  
        temp=A[pos]  
        A[pos]:=A[i];  
        A[i]:=temp  
    }  
    return(A);  
}
```

SELECTION SORT					
VERIFY THE ALGORITHM					
PASS 1: Fix the first min data in its positon					
INDEX	KEY-A	l=1,pos=2	l=1,pos=2	l=1,pos=4	l=1, pos=4
1	23	23	23	23	7
2	12	12	12	12	12
3	18	18	18	18	18
4	7	7	7	7	23
5	20	20	20	20	20
RULE 1 : Find the minimum element position and interchange A[i] and A[pos]					

PASS 2: Fix the second minimum data in its positon							
INDEX	KEY-A	l=2, pos=2	l=2,pos=2	l=2, pos=2			
1	7	7	7	7			
2	12	12	12	12			
3	18	18	18	18			
4	23	23	23	23			
5	20	20	20	20			

PASS 3: Fix the third minimum data in its positon							
INDEX	KEY-A	l=3,pos=3	i=3, pos=3				
1	7	7	7				
2	12	12	12				
3	18	18	18				
4	23	23	23				
5	20	20	20				

PASS 4: Fix the fourth minimum data in its positon							
INDEX	KEY-A	l=4, pos=5					
1	7	7					
2	12	12					
3	18	18					
4	23	20					
5	20	23					

6. Write an algorithm for Insertion Sort an array of n elements.

Algorithm **Insertion Sort**(A, n)

// This algorithm return the sorted(ascending order) array for the given array A with size n

```
{  
    for i:=2 to n do{  
        insertdata :=A[i];  
        j:=i-1;  
        while (j > 0 AND A[j] > insertdata){  
            A[j+1]=A[j];  
            j:=j-1;  
        }  
        A[j+1]:=insertdata;  
    }  
    return(A);  
}
```

Performance Analysis:

- Performance analysis or analysis of algorithms refers to the task of determining the efficiency of an algorithm
 - To judge an algorithm, particularly two things are taken into consideration

Space complexity

Time complexity

- Space Complexity

The space needed by an algorithm is the sum of a fixed part and a variable part.

- **Space complexity $S(P) = C + Sp(\text{Instance characteristics})$**

- The fixed part includes space for

- Instructions
 - Simple variables
 - Fixed size component variables
 - Space for constants, Etc..

- The variable part includes space for

- Component variables whose size is dependant on the particular problem instance being solved
 - Recursion stack space, Etc..

Examples

Algorithm NEC (float x, float y, float z)

```
{  
    Return (X + Y+Y*Z + (X + Y+Z)) /(X+ Y) + 4.0;  
}
```

In the above algorithm, there are no instance characteristics and the space needed by X, Y, Z is independent of instance characteristics, therefore we can write,

$$S(XYZ) = 3 + 0 = 3$$

One space each for X, Y and Z

Space complexity is $O(1)$.

Algorithm ADD (float [], int n)

```
{  
    sum := 0.0; for i:=1 to n do  
        sum:=sum +X[i]; return sum;  
}
```

Here, atleast n words since X must be large enough to hold the n elements to be summed.

Here the problem instances is characterized by n, the number of elements to be summed. So, we can write,

$$S(ADD) = 3 + n$$

3-one each for n, I and sum

Where n- is for

Time Complexity

The time complexity of a problem is

- The number of steps that it takes to solve an instance of the problem as a function of the size of the input (usually measured in bits), using the most efficient algorithm.
- **Priori analysis**
- **Posteriori analysis** or *run* (execution) *time*.

- **Time complexity $T(P) = C + TP(n)$**
- In general it can be two measured in ways:
 - **By Bruit force method Time= Compile Time +Run Time**
 - Example:
$$T_p(n) = C_{aADD}(n) + C_{sSUB}(n) + C_{mMUL}(n) + C_{dDIV}(n) + \dots$$
 - **By using step count method.**

Time complexity cases

- **Best Case:** Inputs are provided in such a way that the minimum time is required to process them. $O(1)$
- **Average Case:** The amount of time the algorithm takes on an average set of inputs. $O(n/2)$
- **Worst Case:** The amount of time the algorithm takes on the worst possible set of inputs. $O(n)$

Example: Linear Search

	3	4	5	6	7	9	10	12	15
A	1	2	3	4	5	6	7	8	9

Contd....

2. Global variable count method:

Statement	S/e	Frequency	Total steps
1. Algorithm Sum(a, n)	0	-	0
2. {	0	-	0
3. s:=0; Ex:	1	1	1
4. for i:=1 to n do	1	n+1	n+1
5. s:=s+a[i];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
Total			2n+3 steps

Statement	s/e	frequency	total steps
1 Algorithm Sum(a, n)	0	—	0
2 {	0	—	0
3 $s := 0.0$;	1	1	1
4 for $i := 1$ to n do	1	$n + 1$	$n + 1$
5 $s := s + a[i]$;	1	n	n
6 return s ;	1	1	1
7 }	0	—	0
Total			$2n + 3$

Table 1.1 Step table for Algorithm 1.6

Statement	s/e	frequency		total steps	
		$n = 0$	$n > 0$	$n = 0$	$n > 0$
1 Algorithm RSum(a, n)	0	—	—	0	0
2 {					
3 if ($n \leq 0$) then	1	1	1	1	1
4 return 0.0;	1	1	0	1	0
5 else return					
6 RSum($a, n - 1$) + $a[n]$;	$1 + x$	0	1	0	$1 + x$
7 }	0	—	—	0	0
Total				2	$2 + x$

$$x = t_{\text{RSum}}(n - 1)$$

Time complexity of Matrix addition

Statement	s/e	frequency	total steps
1 Algorithm Add(a, b, c, m, n)	0	—	0
2 {	0	—	0
3 for $i := 1$ to m do	1	$m + 1$	$m + 1$
4 for $j := 1$ to n do	1	$m(n + 1)$	$mn + m$
5 $c[i, j] := a[i, j] + b[i, j];$	1	mn	mn
6 }	0	—	0
Total			$2mn + 2m + 1$

Time complexity of Fibonacci number

```
1  Algorithm Fibonacci(n)
2  // Compute the nth Fibonacci number.
3  {
4      if (n ≤ 1) then
5          write (n);
6      else
7          {
8              fnm2 := 0; fnm1 := 1;
9              for i := 2 to n do
10                 {
11                     fn := fnm1 + fnm2;
12                     fnm2 := fnm1; fnm1 := fn;
13                 }
14             write (fn);
15         }
16 }
```

Algorithm 1.14 Fibonacci numbers

Example: Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21 - - -

$$f_n = f_{n-1} + f_{n-2} \quad \therefore n \geq 2$$

total steps

Algorithm Fibonacci(n)

{ if (n ≤ 1) then

write (n);

else

fib2 := 0; fib1 := 1;

for i := 2 to n do

{

fib := fib1 + fib2;

fib2 := fib1;

fib1 := fib;

} write(fib);

}

	$n \leq 1$	$n > 1$
	1	0
	1	0
	0	0
	0	2
	0	$n +$
	0	n
	0	n
	0	n
	0	0
	0	1
	2	$4n + 3$

		Total steps	
		$n=0$	$n>0$
3. Algorithm Rsum(a, n)		0	0
{			
if (n ≤ 0) then		1	1
return(0);		1	0
else		0	0
return Rsum(a, (n-1)) + a(n);		0	1+2
}		0	0
		2	2+2

$$\therefore x = \text{Rsum}(n-1)$$

		Total steps
4. Algorithm Add(a, b, c, m, n)		0
{		0
for i := 1 to m do		m+1
for j := 1 to n do		(n+1)m
c[i, j] = a[i, j] + b[i, j];		mn
}		0
		2mn + 2m + 1

§ Examples on Time complexity:

1. Function for sum:

$$T(n) = n.$$

Algorithm SUM(a, n)

{

$s := 0;$ $c = c+1$ initialize

for $i := 1$ to n do $c = c+1$ for for.

$s := s + a[i];$

return $s;$

}

Total number of steps

$$= \underline{\underline{2n + 3}}$$

Total steps

2. Algorithm SUM(a, n)

{

$s := 0$

for $i := 1$ to n do

$s := s + a[i];$

return $s;$

}

Contd....

□ Step count method have two approaches:

1. Global variable count method:

Example:

Algorithm Sum(a, n)

```
{
s:=0;

for i:=1 to n do
{

s:=s+a[i];

}
returns;
}
```

Algorithm sum with count statement added

count:=0;

Algorithm Sum(a,n)

```
{
s:=0;
count:=count+1;
for i:=1 to n do
{
count:=count+1;
s:=s+a[i];
count:=count+1;
}
count:=count+1; //for last time of for loop
count:=count+1; //for return statement
return s;
}
```

Thus the total number of steps are $2n+3$

Important questions

- (1) Define algorithm
- (2) What is pseudo code?
- (3) How to device an algorithm?
- (4) How to validate an algorithm?
- (5) How to analysis an algorithm?
- (6) How to test a program?
- (7) What are the characteristics of an algorithm?
- (8) Write an algorithm for linear sort.
- (9) Write an algorithm for binary search
- (10) Write an algorithm for selection sort.
- (11) Write an algorithm for finding max and min in an array of n elements.
- (12) Write an algorithm to check the give positive integer is prime or not.
- (13) Write an Euclidean algorithm for finding GCD.