Unit I

1 Introduction to Algorithm and Data Structures

1.1 : From Problem to Program and Concept of Data Structures

Q.1 Define the term Data structure.

Ans. : A data structure is a particular way of organizing data in a computer so that it can be used effectively.

For example - Consider set of elements which can be stored in array data structure. The representation of array containing set of elements is as follows -

0	1	2	3	4
10	20	30	40	50

Q.2 Give the difference between Data and Information.

Ans. :

Sr. No.	Data	Information
1.	Data is in raw form.	Information is processed form of data.
2.	Data may or may not be meaningful.	Information is always meaningful.
3.	Data may not be in some specific order.	Information generally follows specific ordering.
4.	For example - Each student's marks in exam is one piece of data.	For example - The average score of a class is information that can be derived from given data.

1.2 : Abstract Data Structures

Q.3 What is ADT ? Write ADT for an arrays.

☞ **[SPPU : May-10,11, Dec.-11, Marks 6, May-12, Marks 4]**

Ans. : The abstract data type is a triple of D - Set of domains, F - Set of functions, A - axioms in which only what is to be done is mentioned but how is to be done is not mentioned.

In ADT, all the implementation details are hidden.

AbstractDataType Array

{

Instances : An array A of some size, index i and total number of elements in the array n.

Operations :

Store () – This operation stores the desired elements at each successive location.

display () – This operation displays the elements of the array.

}

Q.4 Define (1) ADT (2) Data Structure. ☞ **[SPPU : May-19, Marks 2]**

Ans. : Refer Q.3 and Q.1.

1.3 : Data Structure Classification

Q.5 Write short note on Linear and Non-Linear data structure with example.

☞ **[SPPU : Dec.-19, Marks 4]**

OR Differentiate between linear and non linear data structure with example.

☞ **[SPPU : May-17, Marks 3]**

Ans. : Linear data structures are the data structures in which data is arranged in a list or in a straight sequence.

For example : arrays, list.

Non linear data structures are the data structures in which data may be arranged in hierarchical manner. **For example :** trees, graphs

Q.6 Explain static and dynamic data structures with examples.

☞ **[SPPU : Dec.-18, Marks 4]**

Ans. : The static data structures are data structures having fixed size memory utilization. One has to allocate the size of this data structure before using it.

For example (Static Data Structure) :

Arrays in C is a static data structure. We first allocate the size of the arrays before its actual use. Sometimes what ever size we have allocated for the arrays may get wasted or we may require larger than the one which is existing. Thus the data structure is static in nature.

The dynamic data structure is a data structure in which one can allocate the memory as per his requirement. If one does not want to use some block of memory, he can deallocate it.

For example (Dynamic Data Structure) :

Linked list, the linked list is a collection of many nodes. Each node consists of data and pointer to next node. In linked list user can create as much nodes (memory) as he wants, he can dellocate the memory which cannot be utilized further.

The advantage of dynamic data structure over the static data structure is that there is no wastage of memory.

1.4 : Problem Solving

Q.7 State a reason why each of the six problem solving steps is important in developing the best solution for a problem. Give one reason for each step. ☞ **[SPPU : May-10, Marks 8]**

OR Consider any one problem and solve that problem using six steps of problem solving. Explain each step in detail.

☞ **[SPPU : Dec.-10, Marks 8]**

OR Describe the six steps in problem solving with example.

☞ **[SPPU : May-11, Marks 8]**

Ans. : Problem solving is based on the decisions that are taken. Following are the **six steps of problem solving -**

1. **Identify the problem :** Identifying the problem is the first step in solving the problem. Problem identification is very essential before solving any problem.

2. **Understand the problem** : Before solving any problem it is important to understand it. There are three aspects based on which the problem can be understood.
   - **Knowledgebase** : While solving the problem the knowledgebase can be related to **a person or a machine**. If the problem is to be solved for the person then it is necessary to know **what the person knows**. If the problem is to be solved for the machine then its **instruction set** must be known. Along with this the problem solver can make use of his/her own instruction set.
   - **Subject** : Before solving the problem the subject on which the problem is based must be known. For instance : To solve the problem involving Laplace, it is necessary to know about the Laplace transform.
   - **Communication** : For understanding the problem, the developer must communicate **with the client**.
3. **Identify the alternative ways to solve the problem** : The alternative way to solve the problem must be known to the developer. These alternatives can be decided by **communicating** with the customer.
4. **Select the best way to solve the problem from list of alternative solutions** : For selecting the best way to solve the problem, the merits and demerits of each problem must be analysed. The criteria to evaluate each problem must be predefined.
5. **List the instructions using the selected solution** : Based on the knowledgebase (created/used in step 2) the step by step instructions are listed out. Each and every instruction must be understood by the person or the machine involved in the problem solving process.
6. **Evaluate the solution** : When the solution is evaluated then - i) Check whether the solution is correct or not ii) Check whether it satisfies the requirements of the customer or not.

1.5 : Difficulties in Problem Solving

Q.8 State and explain any four difficulties with problem solving.

☞ **[SPPU : Dec.-10, Marks 4]**

OR What are the difficulties in problem solving ? Explain any four steps in problem solving suitable example.

☞ **[SPPU : May-16, Marks 4]**

Ans. : Various difficulties in solving the problem are -

- People do **not know** how to solve particular problem.
- Many times people get **afraid of** taking decisions
- While following the problem solving steps people complete one or two **steps inadequately**.
- People do **not define** the problem statements correctly.
- They do not generate the **sufficient list of alternatives**. Sometimes good alternatives might get eliminated. Sometimes the merits and demerits of the alternatives is defined hastily.
- The **sequence of solution** is not defined **logically**, or the focus of the design is sometimes on **detailed work** before the framework solution.
- When solving the problems on **computer** then **writing the instructions** for the computer is most crucial task. With **lack of knowledgebase** one can not write the proper instruction set for the computers.

1.6 : Introduction to Algorithm

Q.9 Define Algorithm and its characteristics.

☞ **[SPPU : Dec.-16, Marks 4, May-19, Marks 2]**

Ans. : Definition of Algorithm : An algorithm is a finite set of instructions for performing a particular task. The instructions are nothing but the statements in simple English language.

1. Each algorithm is supplied with **zero or more inputs.**
2. Each algorithm must **produce** at least one **output**.
3. Each algorithm should have **definiteness** i.e. each instruction must be **clear and unambiguous.**
4. Each algorithm should have **finiteness** i.e. if we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after finite number of steps.

5. Each algorithm should have **effectiveness** i.e. every instruction must be **sufficiently** basic that it can in principal be carried out by a person using only pencil and paper. Moreover each instruction of an algorithm must also be feasible.

1.7 : Algorithm Design Tools

Q.10 What do you mean by flow chart ? Give the meaning of each symbol used in flowchart. Draw flowchart to compute the sum of elements from a given integer array. ☞ **[SPPU : May-10, Marks 8]**

Ans :

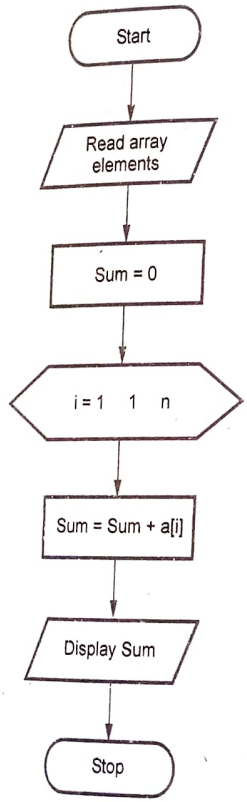


Fig. Q.10.1

Q.11 Design and explain an algorithm to find the sum of the digits of an integer number. ☞ **[SPPU : Dec.-10, Marks 6]**

Ans. :

```
Read N
Remainder=0
Sum=0
Repeat
  Remainder=N mod 10
  Sum=Sum+remainder
  N=N/10
Until N<0
Display Sum
End
```

Q.12 What is a difference between flowchart and algorithm ? Convert the algorithm for computing factorial of given number into flowchart. ☞ **[SPPU : May-11, Marks 8]**

Ans. : Algorithm is a set of instructions written in natural language or pseudo code and flowchart is a graphical representation of an **algorithm**.

```
Read N
Set i and F to 1
While i<=N
  F=F * i
  Increase the value of i by 1
Display F
End
```

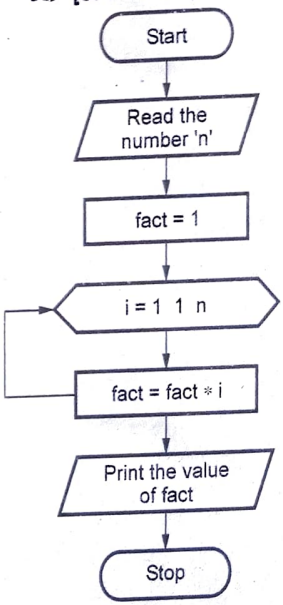


Fig. Q.12.1 Flowchart for factorial

1.8 : Step Count Method

Q.13 What is step count method ? Explain it with example.

Ans. : Definition : The frequency count is a count that denotes how many times particular statement is executed.

For Example : Consider following code for counting the frequency count

```
void fun()
{
 int a;
 a=10; .....1
 printf("%d",a); .....1
}
```

The frequency count of above program is **2.**

1.9 : Complexity of Algorithm

Q.14 What is time complexity ? How is time complexity of an algorithm computed ? ☞ **[SPPU : Dec.-11, Marks 6]**

Ans. : The amount of time required by an algorithm to execute is called the time complexity of that algorithm.

For determining the time complexity of particular algorithm following steps are carried out -

1. Identify the basic operation of the algorithm.
2. Obtain the frequency count for this basic operation.
3. Consider the order of magnitude of the frequency count and express it in terms of big oh notation.

Q.15 Write an algorithm to find smallest element in a array of integers and analyze its time complexity. ☞ **[SPPU : May-13, Marks 8]**

Ans. :

```
Algorithm MinValue(int a[n])
{
   min_element=a[0];
```



```
   for(i=0;i<n;i++)
   {
      if (a[i]<min_element) then
         min_element=a[i];
   }
   Write(min_element);
}
```

We will first obtain the frequency count for the above code

Statement	Frequency Count
min_element=a[0]	1
for(i=0;i<n;i++)	n+1
if (a[i]<min_element) **then**	n
Write(min_element);	1
Total	2n+3

By neglecting the constant terms and by considering the order of magnitude we can express the frequency count in terms of Omega notation as O(n). Hence the frequency count of above code is O(n).

Q.16 What is space complexity of an algorithm ? Explain its importance with example. ☞ **[SPPU Dec.-10, Marks 4]**

Ans. : The space complexity can be defined as amount of memory required by an algorithm to run.

To compute the space complexity we use two factors : **Constant and instance characteristics**. The space requirement S(p) can be given as

$$S(p) = C + Sp$$

where **C** is a constant i.e. fixed part and it denotes the space of inputs and outputs. This space is an amount of space taken by instruction, variables and identifiers. And **Sp** is a space dependent upon instance characteristics. This is a variable part whose space requirement depends on particular problem instance.

There are two types of components that contribute to the space complexity - Fixed part and variable part.

The **fixed part** includes space for

- Instructions
- Variables
- Array size
- Space for constants

The **variable part** includes space for

- The variables whose size is dependent upon the particular problem instance being solved. The control statements (such as for, do, while, choice) are used to solve such instances.
- Recursion stack for handling recursive call.

Q.17 What is complexity of algorithm ? Explain with an example.

[SPPU : Dec.-19, Marks 3]

Ans. : Complexity represents the amount of memory or time required for a program to execute. There are two types of complexities - time complexity and space complexity.

Refer Q.14 and Q.16.

1.10 : Asymptotic Notations

Q.18 Explain asymptotic notations Big O, Theta and Omega with one example each.

[SPPU : Dec.-16, 17, May-18, Marks 6, Dec.-19, Marks 3]

OR What is complexity analysis of an algorithm ? Explain the notations used in complexity analysis. **[SPPU : May-19, Marks 6]**

Ans. : Using asymptotic notations we can give time complexity as "fastest possible", "slowest possible" or "average time".

Various notations such as Ω, Θ and O used are called **asymptotic notations.**

1. **Big oh Notation :** The **Big oh** notation is denoted by 'O'. It is a method of representing the **upper bound of** algorithm's running time. Using big oh notation we can give longest amount of time taken by the algorithm to complete.

Definition

Let f(n) and g(n) be two non-negative functions.

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that $c > 0$. We can write

$$f(n) \leq c*g(n)$$

then f(n) is big oh of g(n). It is also denoted as $f(n) \in O(g(n))$. In other words f(n) is less than g(n) if g(n) is multiple of some constant c.

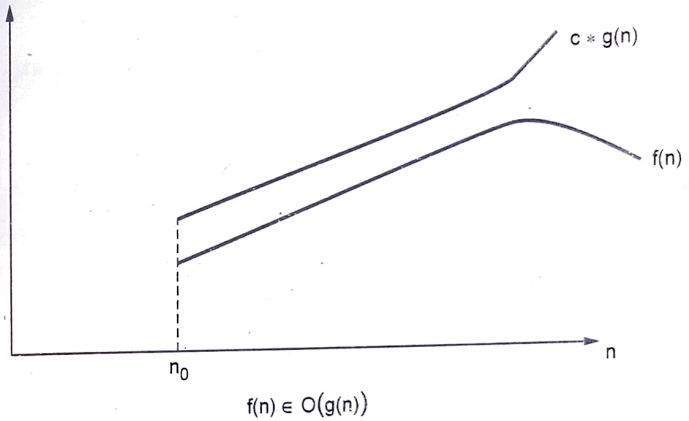


Fig. Q.18.1 Big oh notation

2. **Omega Notation :** Omega notation is denoted by 'Ω'. This notation is used to represent the **lower bound** of algorithm's running time. Using omega notation we can denote shortest amount of time taken by algorithm.

Definition

A function f(n) is said to be in Ω (g(n)) if f(n) is bounded below by some positive constant multiple of g(n) such that

$$f(n) \geq c * g(n)$$

For all $n \geq n_0$

It is denoted as $f(n) \in \Omega(g(n))$. Following graph illustrates the curve for Ω notation.

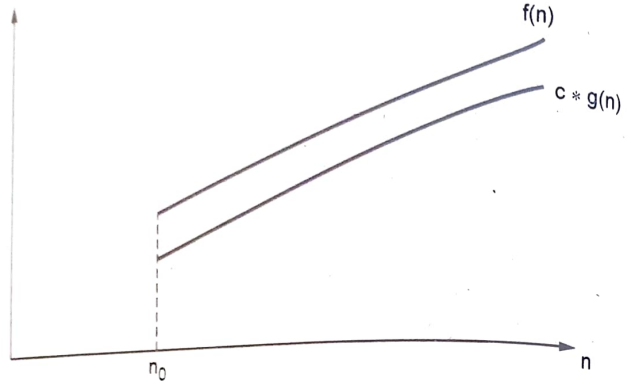


Fig. Q.18.2 Omega notation f(n) ∈ Ω (g(n))

3. Θ Notation : The theta notation is denoted by Θ. By this method the running time **is between upper bound and lower bound.**

Definition

Let f(n) and g(n) be two non negative functions. There are two positive constants namely c_1 and c_2 such that

$$c_1 \leq g(n) \leq c_2\, g(n)$$

Then we can say that

$$f(n) \in \Theta(g(n))$$

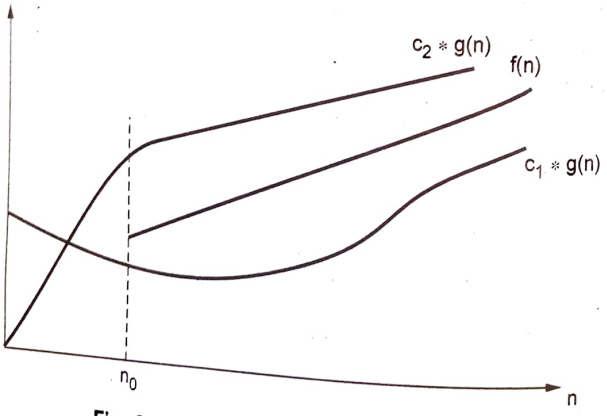


Fig. Q.18.3 Theta notation f(n) ∈ Θ (g(n))

1.11 : Analysis of Programming Constructs

Q.19 Find the frequency count (F.C.) of the given code : Explain each step.

```
double IterPow(double X, int N)
{
double Result = 1;
 while (N > 0)
 {
Result = Result *X;
N--;
}
return Result;
}
```

☞ **[SPPU : May-10, Marks 6]**

Solution :

Statement	Frequency Count
double Result=1	1
while(N>0)	N+1
Result=Result*X	N
N--	N
return Result	1
Total	**3N+3**

Q.20 What is frequency count of a statement ? Analyze time complexity of the following code :

i)

```
for(i = 1; i <= n; i++)
for(j = 1; j <= m; j++)
for(k = 1; k <= p; k++)
sum = sum + i;
```

ii)

```
i = n;
while(i ≥ 1)
 {i- -;}
```

☞ **[SPPU : Dec.-11, Marks 6, May-14, Marks 3]**

Solution : i)

Statement	Frequency Count
for(i=1;i<=n;i++)	n+1
for(j=1;j<=m;j++)	n(m+1)
for(k=1;k<=p;k++)	n.m.(p+1)
sum=sum+i	n.m.p
Total	2(n+nm+nmp)+1

ii)

Statement	Frequency Count
i=n	1
while(i>=1)	n+1
i- -	n
Total	2(n+1)

Q.21 Determine the frequency counts for all the statements in the following program segment.

```
i=10;
for(i=10;i<=n;i++)
  for(j=1;j<i;j++)
x=x+1;
```

☞ **[SPPU : May-13, Marks 6]**

Ans. : Assume (n–10+2)=m. Then the total frequency count is -

The outer loop will execute for m times. The inner loop is dependant upon the outer loop. Hence it will be (m+1)/2. Hence overall frequency count is $(1+m+m^2)(m(m+1)/2)$

If we consider only the order of magnitude of this code then overall time complexity in terms of big oh notation is $O(n^2)$.

Statement	Frequency Count
i=10;	1
for(i=10;i<=n;i++)	m (we assume the count of execution is m)
for(j=1;j<i;j++)	m((m+1)/2)
x=x+1;	m(m)

If an algorithm takes time O(logn) then it is faster than any other algorithm, for larger value of n. Similarly O(nlogn) is better than $O(n^2)$. Various computing time can be O(1), O(logn), O(n),O(nlogn), $O(n^2)$, $O(n^3)$ and $O(2^n)$.

1.12 : Recurrence Relation

Q.22 What is recurrence relation ? Explain with example.

☞ **[SPPU : Dec.-18, Marks 2]**

Ans. : Definition : The recurrence equation is an equation that defines a sequences resursively. It is normally in following form :

$$T(n) = T(n-1) + n \text{ for } n > 0 \quad (1)$$

Recurrence relation

$$T(0) = 0 \quad (2)$$

Initial condition

- The recurrence equation can have infinite number of sequence.
- **The general solution** to the recursive function specifies some formula.
- **For example :** Consider a recurrence relation

$$T(n) = 2T(n-1) + 1 \quad \text{for } n > 1$$

$$T(1) = 1$$

By solving this relation we will get $T(n) = 2^n - 1$ where $n > 1$.

1.13 : Solving Recurrence Relation

Q.23 Solve the following recurrence relation $T(n) = T(n-1) + 1$ with $T(0) = 0$ as initial condition. Also find big oh notation.

Ans. : Let,

$$T(n) = T(n-1) + 1$$

By backward substitution,

$$T(n-1) = T(n-2) + 1$$

$$\therefore \quad T(n) = T(n-1) + 1$$

$$= (T(n-2) + 1) + 1$$

$$\mathbf{T(n) = T(n-2) + 2}$$

$$\text{Again } T(n-2) = T(n-2-1) + 1$$

$$= T(n-3) + 1$$

$$\therefore \quad T(n) = T(n-2) + 2$$

$$= (T(n-3) + 1) + 2$$

$$T(n) = T(n-3) + 3$$

$$\vdots$$

$$T(n) = T(n-k) + k \qquad \ldots (1)$$

If $k = n$ then equation (1) becomes

$$T(n) = T(0) + n$$

$$= 0 + n \qquad \because T(0) = 0$$

$$T(n) = n$$

$\therefore$ We can denote $T(n)$ in terms of big oh notation as

$$\mathbf{T(n) = O(n)}$$

Q.24 Solve the following recurrence relations -

i) $T(n) = 2T\left(\frac{n}{2}\right) + C \quad T(1) = 1$ **ii)** $T(n) = T\left(\frac{n}{3}\right) + C \quad T(1) = 1.$

Ans. : i) Let

$$T(n) = 2T\left(\frac{n}{2}\right) + C$$

$$= 2\left(2T\left(\frac{n}{4}\right) + C\right) + C$$

$$= 4T\left(\frac{n}{4}\right) + 3C$$

$$= 4\left(2T\left(\frac{n}{8}\right) + C\right) + 3C$$

$$= 8T\left(\frac{n}{8}\right) + 7C$$

$$= 2^3 T\left(\frac{n}{n^3}\right) + (2^3 - 1)\,C$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + (2^k - 1)\,C$$

If we $2^k = n$ then

$$T(n) = nT\left(\frac{n}{n}\right) + (n-1)\,C$$

$$= nT(1) + (n-1)\,C$$

$$\mathbf{T(n) = n + (n-1)\,C} \qquad \because T(1) = 1$$

ii) Let,

$$T(n) = T\left(\frac{n}{3}\right) + C$$

$$= \left(T\left(\frac{n}{9}\right) + C\right) + C$$

$$= T\left(\frac{n}{9}\right) + 2C$$

$$= \left[T\left(\frac{n}{27}\right) + C\right] + 2C$$

$$= T\left(\frac{n}{27}\right) + 3C$$

$$\vdots$$

$$= T\left(\frac{n}{3^k}\right) + kC$$

If we put $3^k = n$ then

$$= T\left(\frac{n}{n}\right) + \log_3 n \cdot C$$

$$= T(1) + \log_3 n \cdot C$$

$$T(n) = C \cdot \log_3 n + 1 \qquad \because T(1) =$$

Q.25 Consider the following code :

```
int sum (int a [ ], n)
{
   count = count + 1 ;
   if (n < = 0)
   {
      count = count + 1 ;
      return 0 ;
   }
   else
   {
      count = count + 1
      return sum (a, n – 1)+a[n] ;
   }
}
```

Write the recursive formula for the above code and solve this recurrence relation.

Ans. : In the above code the following statement gets executed at least twice

count = count + 1

In the else part, there is a recursive call to the function **sum,** by changing the value of n by n – 1, each time.

Hence the recursive formula for above code will be

$$T(0) = 2$$

We can solve this recurrence relation using backward substitution. It will be

$$T(n) = \underbrace{T(n-1)}_{\text{For recursive call to function sum in else part.}} + \underbrace{2}_{\text{For count statement}}$$

$$= [T(n-2) + 2] + 2$$

$$\therefore \quad = T(n-2) + 2(2)$$

$$= [T(n-3) + 2] + 2(2)$$

$$\therefore \quad T(n-2) = T(n-3) + 3 \cdot (2)$$

$$\vdots$$

$$T(n) = T(n-n) + n \cdot (2)$$

$$= T(0) + 2n$$

$$= 2 + 2 \cdot n \qquad \because T(0) = 2$$

$$T(n) = 2(n+1)$$

We can denote the time complexity in the form of big oh notation as **O(n).**

Q.26 What is Master's method ?

Ans. : We can solve recurrence relation using a formula denoted by Master's method.

$T(n) = aT(n/b) + F(n)$ where $n \geq d$ and d is some constant.

Then the Master theorem can be stated for efficiency analysis as -

If $F(n)$ is $\Theta(n^d)$ where $d \geq 0$ in the recurrence relation then,

1.	$T(n) = \Theta(n^d)$	if $a < b^d$
2.	$T(n) = \Theta(n^d \log n)$	if $a = b$
3.	$T(n) = \Theta(n^{\log_b a})$	if $a > b^d$

Q.27 Solve the following recurrence relation.

$T(n) = 2T(n/2) + n \log n$

Ans. :

Here $f(n) = n \log n$

$a = 2, \quad b = 2$

$\log_2 2 = 1$

According to case 2 given in above Master theorem

$f(n) = \Theta(n^{\log_2 2} \log^1 n)$ i.e. $k = 1$

Then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

$$= \Theta(n^{\log_2 2} \log^2 n)$$

$$= \Theta(n^1 \log^2 n)$$

$$\therefore \quad T(n) = \Theta(n \log^2 n)$$

Q.28 Solve the following recurrence relation $T(n) = 9T(n/3) + n^3$.

Solution :

Here $a = 9$, $b = 3$ and $f(n) = n^3$

And $\log_3 9 = 2$

According to case 3 in above Master theorem

As $f(n)$ is $= \Omega(n^{\log_3 9 + \varepsilon})$

i.e. $\Omega(n^{2+\varepsilon})$ and we have $f(n) = n^3$.

Then to have $f(n) = \Omega(n)$. We must put $\varepsilon = 1$.

Then $T(n) = \Theta(f(n))$

Then $T(n) = \Theta(f(n))$

$$T(n) = \Theta(n^3)$$

Q.29 Find the complexity of the following recurrence relation. $T(n) = 9T(n/3) + n$

Solution : Let $T(n) = 9T(n/3) + n$

$\downarrow \quad \downarrow \quad \downarrow$

$a \quad b \quad f(n)$

$\therefore$ We get $a = 9$, $b = 3$ and $f(n) = n$.

$$n^{\log_b a} = n^{\log_3 9} = n^2$$

Now $f(n) = n$

i.e. $T(n) = f(n) = \Theta(n^{\log_b a - \varepsilon}) = \Theta(n^{2-\varepsilon})$

When $\varepsilon = 1$. That means case 1 is applicable. According to case 1, t time complexity of such equations is

$$T(n) = \Theta(n^{\log_b a})$$

$$= \Theta(n^{\log_3 9})$$

$$T(n) = \Theta(n^2)$$

Hence the complexity of given recurrence relation is $\Theta(n^2)$.

Q.30 Solve recurrence relation $T(n) = k \cdot T(n/k) + n^2$ when $T(1) = 1$, and k is any constant.

Ans. : Let,

$$T(n) = k \cdot T\left(\frac{n}{k}\right) + n^2$$

be a recurrence relation. Let us use substitution method

If we assume $k = 2$ then the equation becomes

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2 \qquad T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2$$

$$= 2\left[2 \cdot T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right] + n^2$$

$$= 4T\left(\frac{n}{4}\right) + \frac{n^2}{2} + n^2$$

$$= 4T\left(\frac{n}{4}\right) + \frac{3n^2}{2} \qquad T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2$$

$$= 4\left[2 \cdot T\left(\frac{n}{8}\right) + \frac{n^2}{16}\right] + \frac{3n^2}{2}$$

$$= 8T\left(\frac{n}{8}\right) + \frac{n^2}{4} + \frac{3n^2}{2}$$

$$= 8T\left(\frac{n}{8}\right) + \frac{7n^2}{4}$$

$$= 2^k T\left(\frac{n}{2^k}\right) + \frac{2^k - 1}{2^{k-1}} n^2$$

$$= 2^k T\left(\frac{n}{2^k}\right) + Cn^2 \qquad \because \frac{2^k - 1}{2^{k-1}} = C = \text{Constant}$$

If we put $\frac{n}{2^k} = 1$ then $2^k = n$ and $k = \log_2 n$

$$= nT(1) + Cn^2$$

$= n + Cn^2$ $\because T(1) = 1$

$T(n) = \Theta(n)^2$

Alternate Method

We can solve the given recurrence relation using Master's theorem also.

$$T(n) = k \cdot T\left(\frac{n}{k}\right) + n^2$$

$\downarrow \quad \downarrow \quad \downarrow$

$a \quad b \quad f(n)$

By Master's theorem $n^{\log_b a} = n^{\log_k k} = n^1$

For $T(n) = f(n)$ we need

$T(n) = n^{\log_k k + \varepsilon}$ Applying case 3

$= n^{1+1}$ when $\varepsilon = 1$

$= f(n)$ i.e. n^2

$\therefore \quad T(n) = \Theta(f(n))$

$\mathbf{T(n) = \Theta(n^2)}$

1.14 : Best, Worst and Average Case Analysis

Q.31 Explain Best case, worst case and average case analysis.

Ans. : If an algorithm takes minimum amount of time to run to completion for a specific set of input then it is called **best case** time complexity.

For example : While searching a particular element by using sequential search we get the desired element at first place itself then it is called best case time complexity.

If an algorithm takes maximum amount of time to run to completion for a specific set of input then it is called **worst case** time complexity.

For example : While searching an element by using linear searching method if desired element is placed at the end of the list then we get worst time complexity.

The time complexity that we get for certain set of inputs is as a average same. Then for corresponding input such a time complexity is called **average case** time complexity.

Consider the following algorithm

```
Algorithm Seq_search(X[0 ... n – 1 ],key)
// Problem Description: This algorithm is for searching the
//key element from an array X[0...n – 1] sequentially.
//Input: An array X[0...n – 1] and search key
//Output: Returns the index of X where key value is present
for i ← 0 to n – 1 do
 if(X[i]=key)then
  return i
```

Best case time complexity

Best case time complexity is a time complexity when an algorithm runs for short time. In above searching algorithm the element **key** is searched from the list of **n** elements. If the **key** element is present at first location in the list(X[0...n–1]) then algorithm runs for a very short time and thereby we will get the best case time complexity. We can denote the best case time complexity as

$$C_{best} = 1$$

Worst case time complexity

Worst case time complexity is a time complexity when algorithm runs for a longest time. In above searching algorithm the element **key** is searched from the list of **n** elements. If the **key** element is present at n^{th} location then clearly the algorithm will run for longest time and thereby we will get the worst case time complexity. We can denote the worst case time complexity as

$$C_{worst} = n$$

The algorithm guarantees that for any instance of input which is of size n, the running time will not exceed $C_{worst}(n)$. Hence the worst case time complexity gives important information about the efficiency of algorithm.

Average case time complexity

This type of complexity gives information about the behaviour of an algorithm on specific or random input. Let us understand some terminologies that are required for computing average case time complexity.

Let the algorithm is for sequential search and

P be a probability of getting successful search.

n is the total number of elements in the list.

The first match of the element will occur at i^{th} location. Hence probability of occurring first match is P/n for every i^{th} element.

The probability of getting unsuccessful search is (1 – P).

Now, we can find average case time complexity $C_{avg}(n)$ as -

$C_{avg}(n)$ = Probability of successful search

(for elements 1 to n in the list)

+ Probability of unsuccessful search

$$C_{avg}(n) = \left[1\cdot\frac{P}{n} + 2\cdot\frac{P}{n} + \ldots + i\cdot\frac{P}{n}\right] + n\cdot(1-P)$$

There may be n elements at which chances of 'not getting element' are possible. Hence n.(1 – P)

$$= \frac{P}{n}[1+2+\ldots+i\ldots n] + n(1-P)$$

$$= \frac{P}{n}\frac{n(n+1)}{2} + n(1-P)$$

$$C_{avg}(n) = \frac{P(n+1)}{2} + n(1-P)$$

Thus we can obtain the general formula for computing average case time complexity.

Suppose if P = 0 that means there is no successful search i.e. we have scanned the entire list of n elements and still we do not found the desired element in the list then in such a situation,

$$C_{avg}(n) = 0(n+1)/2 + n(1-0)$$

$$C_{avg}(n) = n$$

Thus the average case running time complexity becomes equal to n.

Suppose if P = 1 i.e. we get a successful search then

$$C_{avg}(n) = 1(n+1)/2 + n(1-1)$$

$$C_{avg}(n) = (n+1)/2$$

That means the algorithm scans about half of the elements from the list.

1.15 : Introduction to Algorithmic Design Strategies

Q.32 What is algorithmic design statregy ?

Ans. : Algorithm design strategy is a general approach by which many problems can be solved algorithmically. These problems may belong to different areas of computing. Algorithmic strategies are also called as **algorithmic techniques** or **algorithmic paradigm**.

Q.33 What is divide and conquer strategy ? Explain it with suitable example. ☞ **[SPPU : Dec.-19, Marks 3]**

Ans. : In divide and conquer method, a given problem is,

1) Divided into smaller sub problems.
2) These sub problems are solved independently.
3) If necessary, the solutions of the sub problems are combined to get a solution to the original problem.

- If the sub problems are large enough, then divide and conquer is reapplied.
- The divide and conquer technique is as shown in Fig. Q.33.1.

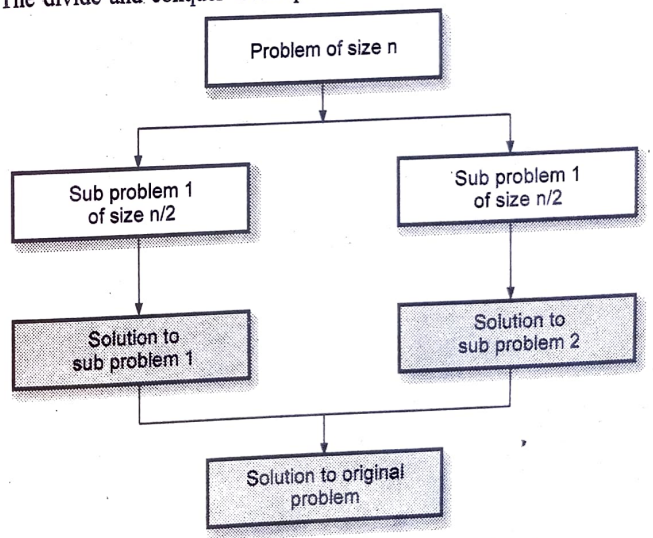


Fig. Q.33.1 Divide and conquer technique

- The generated sub problems are usually of same type as the origin problem. Hence sometimes recursive algorithms are used in divide a conquer strategy.

Example : Merge Sort

- The merge sort is a sorting algorithm that uses the divide and conqu strategy. In this method division is dynamically carried out.
- Merge sort on an input array with n elements consists of three steps :

Divide : partition array into two sub lists s1 and s2 with n/2 elemen each.

Conquer : Then sort sub list s1 and sub list s2.

Combine : merge s1 and s2 into a unique sorted group.

Consider the elements as

70, 20, 30, 40, 10, 50, 60

Now we will split this list into two sublists.

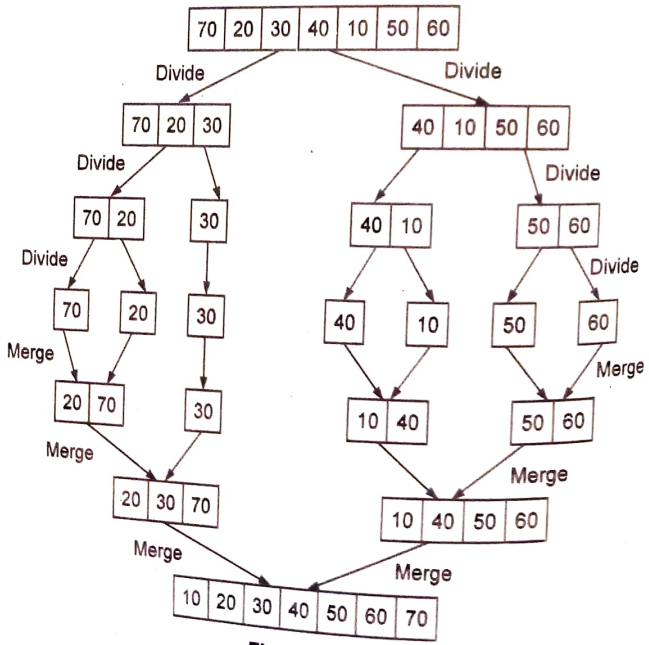


Fig. Q.33.2

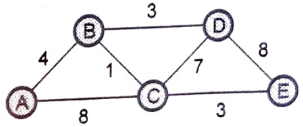
Q.34 Explain the greedy strategy with suitable example. Comment on its time complexity. ☞ **[SPPU : Dec.-17, Marks 6]**

Ans. : This method is popular for obtaining the **optimized solutions.**

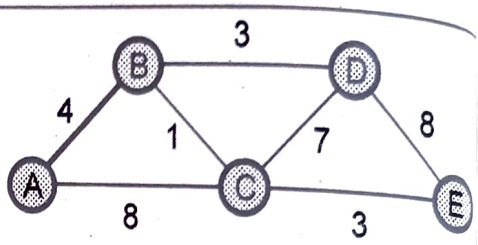
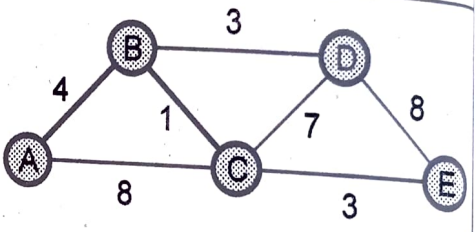
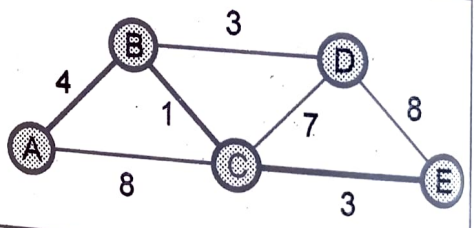
- In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
- In Greedy method following activities are performed.
  1. First we select some solution from input domain.
  2. Then we check whether the solution is feasible or not.
  3. From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function. Such a solution is called optimal solution.
  4. As Greedy method works in stages. At each stage only one input is considered at each time. Based on this input it is decided whether particular input gives the optimal solution or not.

Example of Greedy Method

- Dijkstra's Algorithm is a popular algorithm for finding shortest path using Greedy method. This algorithm is called **single source shortest path** algorithm.
- In this algorithm, for a given vertex called source the shortest path to all other vertices is obtained.
- In this algorithm the main focus is not to find only one single path but to find the shortest paths from any vertex to all other remaining vertices.
- This algorithm applicable to graphs with non-negative weights only.
- Consider a weighted connected graph as given below.



- Now we will consider each vertex as a source and will find the shortest distance from this vertex to every other remaining vertex. Let us start with vertex A.

Source vertex	Distance with other vertices	Path shown in graph
A	**A-B, path = 4** A-C, path = 8 A-D, path = ∞ A-E path = ∞	
B	**B-C, path = 4 + 1** B-D, path = 4 + 3 B-E path = ∞	
C	C-D, path = 5 + 7 = 12 **C-E, path = 5 + 3 = 8**	
D	D-E, path = 7 + 8 = 15	

- But we have one shortest distance obtained from A to E and that is A - B - C - E with path length = 4 + 1 + 3 = 8. Similarly other shortest paths can be obtained by choosing appropriate source and destination.
- Dijkstra's Shortest path algorithm take O(ElogV+VlogV) time, where E is number of edges and V is number of vertices of a graph.

END...