3.1 Introduction

This chapter describes a very interesting algorithmic strategy called divide and conquer. As the name suggests in this strategy the big problem is broken down into **smaller sub problems** and solution to these sub problems is obtained. To understand this strategy we will discuss various applications such as binary search, merge sort, quick sort, matrix multiplication. The time complexities of these applications are also discussed along with the method.

3.2 General Method

- In divide and conquer method, a given problem is,
  - i) Divided into smaller sub problems.
  - ii) These sub problems are solved independently.
  - iii) Combining all the solutions of sub problems into a solution of the whole.
- If the sub problems are large enough then divide and conquer is reapplied.
- The generated sub problems are usually of same type as the original problem. Hence recursive algorithms are used in divided and conquer strategy.
- A **control abstraction** for divide and conquer is as given below - using control abstraction a flow of control of a procedure is given.

```
Algorithm DC(P)
{
    if P is too small then
    return solution of P.
    else
    {
        Divide (P) and obtain P1,P2,.....Pn
        where n ≥ 1
        Apply DC to each subproblem
        return combine (DC(P1),DC(P2)... (DC(Pn));
    }
}
```

- The computing time of above procedure of divide and conquer is given by the recurrence relation.

$$T(n) = \begin{cases} g(n) & \text{if n is small} \\ T(n_1)+T(n_2)+.....T(n_r)+F(n) & \text{when n is sufficiently large} \end{cases}$$

Where T(n) is the time for divide and conquer of size n. The g(n) is the computing time required to solve small inputs. The F(n) is the time required in dividing problem P and combining the solutions to sub problems. Let us now discuss some applications of this method.

The divide and conquer technique is as shown by Fig. 3.2.1.

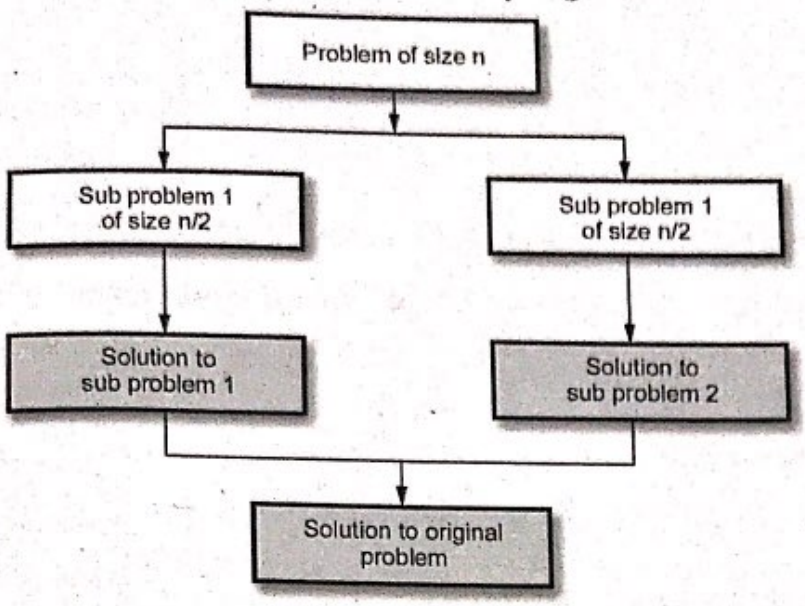


Fig. 3.2.1 Divide and conquer technique

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

For example, if we want to compute sum of n numbers then by divide and conquer we can solve the problem as

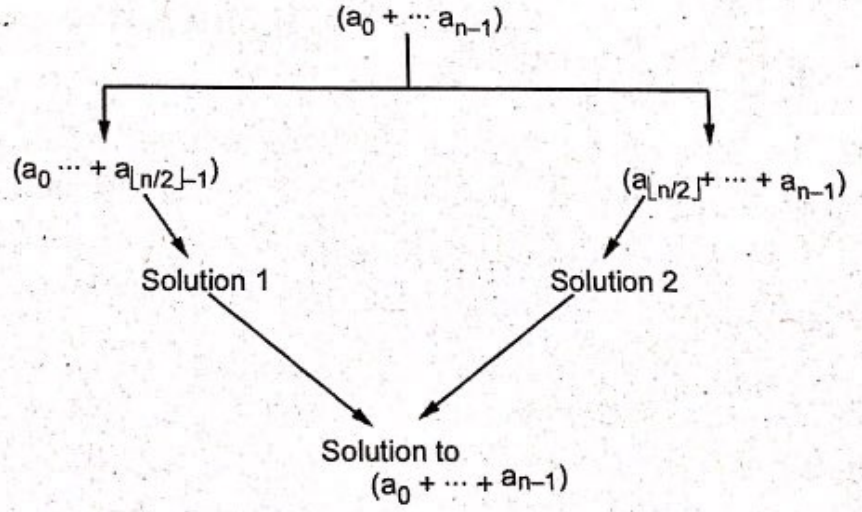


Fig. 3.2.2

3.2.1 Efficiency Analysis of Divide and Conquer

Let recurrence relation be

$$T(n) = aT(n/b) + f(n)$$

Consider $a \geq 1$ and $b \geq 2$. Assume $n = b^k$, where $k = 1, 2, \ldots$.

$$T(b^k) = aT(b^k/b) + f(b^k)$$

$$= aT(b^{k-1}) + f(b^k)$$

$$= a[aT(b^{k-2}) + f(b^{k-1})] + f(b^k)$$

$$= a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

Now substituting $T(b^{k-2})$ by using back substitution,

$$= a^2[aT(b^{k-3}) + f(b^{k-2})] + af(b^{k-1}) + f(b^k)$$

$$= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

Continuing in this fashion we get,

$$= a^kT(b^{k-k}) + a^{k-1}f(b^1) + a^{k-2}f(b^2) + \ldots + a^0 f(b^k)$$

$$= [a^kT(1) + a^{k-1}f(b) + a^{k-2}f(b^2) + \ldots + a^0 f(b^k)]$$

This can also be written as,

$$= a^kT(1) + \frac{a^k}{a}f(b) + \frac{a^k}{a^2}f(b^2) + \ldots + \frac{a^k}{a^k}f(b^k)$$

Taking a^k as common factor

$$= a^k\left[T(1) + \frac{f(b)}{a} + \frac{f(b^2)}{a^2} + \ldots + \frac{f(b^k)}{a^k}\right]$$

$$= a^k\left[T(1) + \sum_{j=1}^{k} \frac{f(b^j)}{a^j}\right]$$

By property of logarithm,

$$a^{\log_b x} = x^{\log_b a}$$

Hence we can write a^k as

$$a^k = a^{\log_b n} = n^{\log_b a}$$

We can rewrite the equation

$$T(n) = a^k\left[T(1) + \sum_{j=1}^{k} f(b^j)/a^j\right]$$

$$T(n) = n^{\log_b a}\left[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j\right]$$

Thus order of growth of T(n) depends upon values of constants a and b and order of growth of function f(n).

3.3 Problem Solving using Divide and Conquer

Various problems that can be solved using divide and conquer strategy are -

1. Binary Search
2. Quick Sort
3. Merge Sort
4. Matrix Multiplication
5. Exponential.

Review Question

1. *Explain the general method for divide and conquer algorithm. Also give the efficiency analysis of it using the recurrence relation.*

3.4 Multiplying Large Integers Problems

GTU : June-11, Dec.-11, Winter-14, Marks 7

We are familiar with the multiplication performed using positional numeral system. This method of multiplying two numbers has taught us in schooling. In this method multiply the multiplicand by each digit of multiplier and then add up all the properly shifted results. This method is also called **grade-school multiplication.**

For example :

$$\begin{array}{r} 42 \\ \times\ 34 \\ \hline 168 \\ +\ 1260 \\ \hline 1428 \end{array} \quad \leftarrow \text{ padding 0 at units position}$$

But this method is not convenient for performing multiplication of large integers. Hence let us discuss an interesting algorithm of multiplying large integers. For example : Consider multiplication of two integers 42 and 34. First let us represent these numbers according to positions.

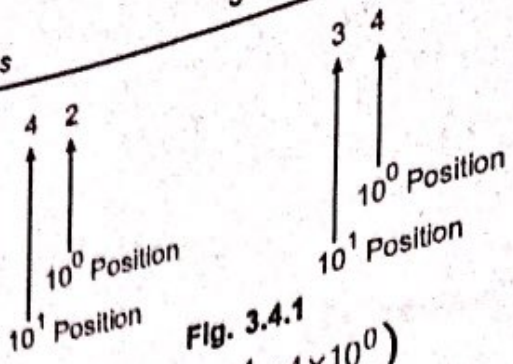


Fig. 3.4.1

i.e. $42\times34 = (4\times10^1+2\times10^0)*(3\times10^1+4\times10^0)$

$= (4\times3)10^2+(4\times4+2\times3)10^1+(2\times4)10^0$

$= 1200 + 220 + 8$

$= 1428$

Let us formulate this method -

Let, $c = a*b$

$c = c_2\,10^2+c_1\,10^1+c_0$ ... (3.4.1)

where $c_2 = a_1*b_1$

$c_0 = a_0*b_0$

$c_1 = (a_1+a_0)*(b_1+b_0)-(c_2+c_0)$

The 2 digit numbers are

$a = a_1\,a_0$

$b = b_1\,b_0$

Let perform multiplication operation with the help of formula given in equation (3.4.1).

$c = a*b$

$= 42\times34$

where $a_1 = 4,\ a_0 = 2$

$b_1 = 3,\ b_0 = 4$

Let us obtain c_0, c_1, c_2 values

$c_2 = a_1*b_1$

$= 4*3$

$c_2 = 12$

$c_0 = a_0*b_0$

$= 2*4$

$c_0 = 8$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$
$$= (4+2) * (3+4) - (12+8)$$
$$= 6 * 7 - 20$$
$$c_1 = \mathbf{22}$$
$$\therefore \quad a * b = c_2\, 10^2 + c_1\, 10^1 + c_0$$
$$= 12 * 10^2 + 22 * 10^1 + 8$$
$$= 1200 + 220 + 8$$
$$a * b = \mathbf{1428}$$

Consider a multiplication of 981 * 1234, as 981 us a three digit number, we will make it four digit by padding a zero to it. Now 0981 * 1234 makes both the operands as 4 digit values.

We will write

$$c = a * b$$
$$= 0981 * 1234$$

Divide the numbers in equal halves.

$$\therefore \quad a_1 = 09 \qquad a_0 = 81$$
$$b_1 = 12 \qquad b_0 = 34$$

Let us obtain c_0, c_1, c_2 Values

$$c_2 = a_1 * b_1$$
$$= 09 * 12$$
$$c_2 = \mathbf{108}$$
$$c_0 = a_0 * b_0$$
$$= 81 * 34$$
$$c_0 = \mathbf{2754}$$
$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$
$$= (9+81) * (12+34) - (108+2754)$$
$$= 4140 - 2862$$
$$c_1 = \mathbf{1278}$$

$$\therefore \quad a * b = c_2 10^4 + c_1 10^2 + c_0$$
$$= 108 * 10000 + 1278 * 100 + 2754$$
$$= 1080000 + 127800 + 2754$$

0981 * 1234 = 1210554

We can generalize this formula as -

$c = a * b$

$c = c_2\ 10^n + c_1\ 10^{n/2} + c_0$

where, n is total number of digits in the integer

$c_2 = a_1 * b_1$

$c_0 = a_0 * b_0$

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

Clearly, this algorithm can be implemented using recursion. This recursion can stopped when n reaches to 0.

Example 3.4.1 *What is divide and conquer technique ? Apply this method to find multiplication on integers 2101 and 1130.* **GTU : IT, Winter-14, Marks 7**

Solution : Refer section 3.2.

We will write

$$c = a * b$$
$$= 2101 * 1130$$

Divide the numbers in equal halves.

$$\therefore \quad a_1 = 21,\ a_0 = 01$$
$$b_1 = 11,\ b_0 = 30$$

Let us obtain c_0, c_1, c_2 values

$$c_2 = a_1 * b_1$$
$$= 21 * 11$$
$$\mathbf{c_2 = 231}$$
$$c_0 = a_0 * b_0$$
$$= 01 * 30$$
$$\mathbf{c_0 = 30}$$
$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$
$$= (21 + 01) * (11 + 30) - (231 + 30)$$

$= (22*41) - 261$

$c_1 = 641$

$a*b = c_2 10^4 + c_1 10^2 + c_0$

$= 231*10^4 + 641*10^2 + 30$

$= 2374130$

Analysis

In this method there are 3 multiplication operations of 1 digit numbers.

i.e. $c_2 = a_1 * b_1$ → Multiplication 1

$c_0 = a_0 * b_0$ → Multiplication 2

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ → Multiplication 3

This is the case for two digit multiplication. Now if there are n digits to be multiplied then multiplication of n digits requires 3 multiplications of n/2 digit numbers. Hence recurrence relation can be given as -

$C(n) = 3\,C(n/2)$ when $n > 1$ ... (3.4.2)

And $C(n) = 1$ when $n = 1$ ... (3.4.3)

Now put $n = 2^k$ in equation (3.4.2) we get,

$C(2^k) = 3\,C(2^k / 2)$

$C(2^k) = 3\,C(2^{k-1})$

$= 3[3\,C(2^{k-2})]$

$= 3[3(3\,C(2^{k-3}))]$

:

$= 3^k (C(2^{k-k}))$

$= 3^k (C(2^0))$

$= 3^k\,C(1)$ $\because\ 2^0 = 1$

Using equation (3.4.3), $C(1) = 1$ ... (3.4.4)

$\therefore \quad C(2^k) = 3^k$

As $n = 2^k$ we get $k = \log_2 n$, equation (3.4.4) becomes

$\because \; a \log_b c = c \log_2 a$

$$C(n) = 3 \log_2 n$$
$$= n \log_2 3$$

$$\boxed{C(n) = n^{1.58}}$$

Review Questions

1. *Explain how multiplication of large integers can be done efficiently by using divide and conquer technique ?* **GTU : June-11, Marks 4**
2. *Explain how divide and conquer method help multiplying two large integers.* **GTU : Dec.-11, Marks 4**

3.5 Binary Search

GTU : Dec.-10,11, June-12, Winter-14, Marks 7

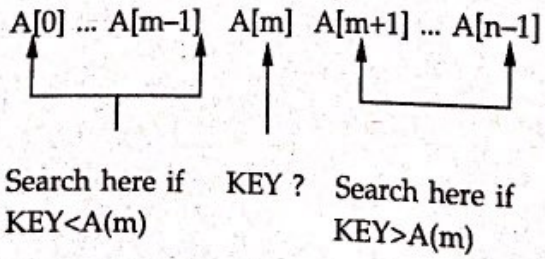
Binary search is an efficient searching method. While searching the elements using this method the most essential thing is that the elements in the array should be sorted one.

An element which is to be searched from the list of elements stored in array A[0...n – 1] is called KEY element.

Let A[m] be the mid element of array A. Then there are three conditions that needs to be tested while searching the array using this method

1. If KEY=A[m] then desired element is present in the list.
2. Otherwise if KEY<A[m] then search the left sub list
3. Otherwise if KEY>A[m] then search the right sub list.

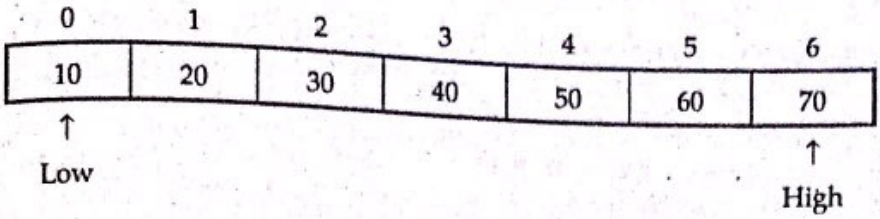
This can be represented as



Let us take one example to understand this method

Example 3.5.1 *Apply binary search method to search 60 from the list 10, 20, 30, 40, 50, 60, 70.*

Solution : Consider a list of elements stored in array A as



The KEY element (i.e. the element to be searched) is 60.

Now to obtain middle element we will apply a formula :

$$m = (low + high)/2$$

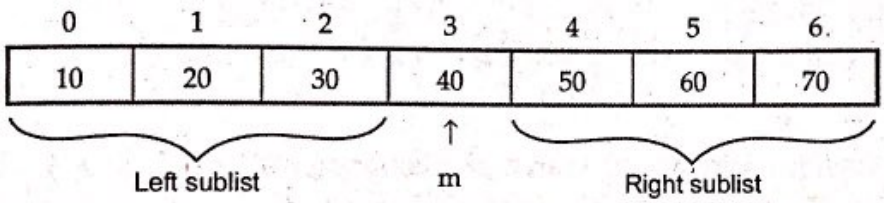
$$m = (0 + 6)/2$$

$$m = 3$$

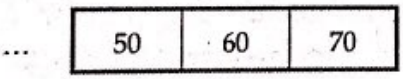
Then Check $A[m] \stackrel{?}{=} KEY$

i.e. $A[3] \stackrel{?}{=} 60$ NO $A[3] = 40$ and $40 < 60$

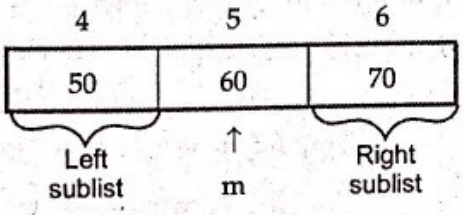
∴ Search the right sublist.



The right sublist is



Now we will again divide this list and check the mid element.



$m = (low + high)/2$

$m = (4 + 6)/2$

$\therefore m = 5$

is A[m] $\stackrel{?}{=}$ KEY

i.e. A[5] $\stackrel{?}{=}$ 60 Yes, i.e. the number is present in the list.

Thus we can search the desired number from the list of elements.

Algorithm

```
Algorithm BinSearch(A[0...n-1],KEY)
//Problem Description: This algorithm is for searching the
//element using binary search method.
//Input: An array A from which the KEY element is to be
//searched.
//Output: It returns the index of an array element if it is
//equal to KEY otherwise it returns -1
low ← 0
high ← n-1
while (low<high) do
{
m ← (low+high)/2      //mid of the array is obtained
      if (KEY=A[m])then
  return m
else if(KEY<A[m])then
  high ← m-1          //search the left sub list
else
  low ← m+1           //search the right sub list
}
return -1             //if element is not present in the list
```

Analysis

The basic operation in binary search is comparison of search key (i.e. KEY) with the array elements. To analyze efficiency of binary search we must count the number of times the search key gets compared with the array elements. The comparison is also called a three way comparison because algorithm makes the comparison to determine whether KEY is smaller, equal to or greater than A[m].

In the algorithm after one comparison the list of n elements is divided into n/2 sublists. The worst case efficiency is that the algorithm compares all the array elements for searching the desired element. In this method one comparison is made and based on this comparison array is divided each time in n/2 sub lists. Hence the worst case time complexity is given by

$$C_{worst}(n) = \underbrace{C_{worst}(n/2)}_{\text{Time required to compare left sublist or right sublist}} + \underbrace{1}_{\text{One comparison made with middle element}} \quad \text{for } n > 1$$

Also, $C_{worst}(1) = 1$

But as we consider the rounded down value when array gets divided the above equations can be written as

$$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1 \quad \ldots (3.5.1)$$

$$C_{worst}(1) = 1 \quad \ldots (3.5.2)$$

The above recurrence relation can be solved further. Assume $n = 2^k$ the equation (3.5.1) becomes,

$$C_{worst}(2^k) = C_{worst}(2^k/2) + 1$$

$$C_{worst}(2^k) = C_{worst}(2^{k-1}) + 1 \quad \ldots (3.5.3)$$

Using backward substitution method, we can substitute

$$C_{worst}(2^{k-1}) = C_{worst}(2^{k-2}) + 1$$

Then equation (3.5.3) becomes

$$C_{worst}(2^k) = \left[C_{worst}(2^{k-2}) + 1\right] + 1$$

$$= C_{worst}(2^{k-2}) + 2$$

Then $C_{worst}(2^k) = \left[C_{worst}(2^{k-3}) + 1\right] + 2$

$$\therefore \quad C_{worst}(2^k) = C_{worst}(2^{k-3}) + 3$$

...

...

...

$$C_{worst}(2^k) = C_{worst}(2^{k-k}) + k$$

$$= C_{worst}(2^0) + k$$

$$C_{worst}(2^k) = C_{worst}(1) + k \quad \ldots (3.5.4)$$

But as per equation (3.5.2),

$C_{worst}(1) = 1$ then we get equation (3.5.4),

$C_{worst}(2^k) = 1 + k$

$\therefore \quad C_{worst}(n) = 1 + \log_2 n$

$\therefore \quad C_{worst}(n) = \log_2 n$

for $n > 1$

As we have assumed

$n = 2^k$

Taking logarithm (base 2) on both sides

$\log_2 n = \log_2 2^k$

$\log_2 n = k \cdot \log_2 2$

$\log_2 n = k(1) \qquad \because \log_2 2 = 1$

$\therefore \quad k = \log_2 n$

The worst case time complexity of binary search is $\Theta(\log_2 n)$.

As $C_{worst}(n) = \log_2 n + 1$ we can verify equation (3.5.1) with this value.

Consider equation (3.5.2),

$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1$

L.H.S.

Assume $n = 2^i$

$\therefore$ substitute

$C_{worst}(n) = \log_2 n + 1$

$= \log_2(2i) + 1$

$= \log_2 2 + \log_2 i + 1$

$= 1 + \log_2 i + 1$

L.H.S. $= \log_2 i + 2$

R.H.S.

Assume $n = 2^i$

$\therefore$ substitute

$C_{worst}(n/2) + 1$

$C_{worst}(2i/2) + 1$

$= C_{worst}(i) + 1$

As $C_{worst}(n) = \log_2 n + 1$

in the same manner

$C_{worst}(i) + 1 = (\log_2 i + 1) + 1$

R.H.S. $= \log_2 i + 2$

Average case

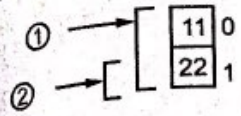
To obtain average case efficiency of binary search we will consider some samples of input n.

If n = 1

i.e. only element 11 is there

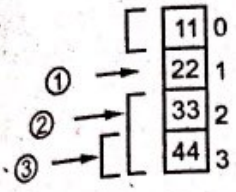
$1 \rightarrow [11]$ Only 1 search is required to search some KEY.

If n = 2 and search key = 22



Two comparisons are made to search 22.

If n = 4 and search key = 44



m = (0 + 3)/2 = 1

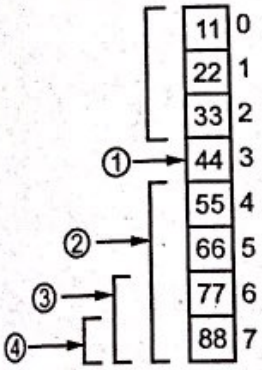
As A[1] = 22 and 22 < 44

Search right sublist, again

m = (2 + 3)/2 = 2

Again A[2] = 33 and 33 < 44 we divide list. In right sublist A[4] = 44 and key is 44. Thus total 3 comparisons are made to search 44.

If n = 8 and search key = 88



m = (0 + 7)/2 = 3

As A[3] = 44 and 44 < 88 search right sublist, again.

m = (4 + 7)/2 = 5

Again A[5] = 66 and 66 < 88 search right sublist, again,

m = (6 + 7)/2 = 6

But A[6] = 77 and 77 < 88.

Then search right sublist

m = (7 + 7)/2 = 7

A[m] = A[7] = 88. Is A[7] $\stackrel{?}{=}$ 88.

Yes, the desired element is present.

Thus total 4 comparisons are made to search 88.

To summarize the above operations.

n	Total comparison (c)
1	1
2	2
4	3
8	4
16	5
⋮	⋮

Observing the above given table we can write

$$\log_2 n + 1 = c$$

For instance if n = 2 then

$$\log_2 2 = 1$$

Then,

$$c = 1 + 1$$

$$c = 2$$

Similarly if n = 8, then

$$c = \log_2 n + 1$$

$$= \log_2 8 + 1$$

$$= 3 + 1$$

$$\therefore \quad c = 4$$

Thus we can write that

Average case time complexity of binary search is Θ (log n).

Advantage of binary search :

1. Binary search is an optimal searching algorithm using which we can search the desired element very efficiently.

Disadvantage of binary search :

1. This algorithm requires the list to be sorted. Then only this method is applicable.

Applications of binary search :

1. The binary search is an efficient searching method and is used to search desired record from database.
2. For solving nonlinear equations with one unknown this method is used.

Time complexity of binary search

Best case	Average case	Worst case
Θ(1)	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$

C Program

There are two types of implementation possible

- Recursive Binary Search Program
- Non Recursive Binary Search Program

Let us understand both the implementations.

```
/*****************************************************************
        Implementation of recursive Binary Search algorithm
*****************************************************************/
#include<stdio.h>
#include<conio.h>
#define SIZE 10
int n;
void main()
{
    int A[SIZE],KEY,i,flag,low,high;
    int BinSearch(int A[SIZE],int KEY,int low,int high);
    clrscr();
    printf("\n How Many elements in an array?");
    scanf("%d",&n);
    printf("\n Enter The Elements");
    for(i=0;i<n;i++)
    scanf("%d",&A[i]);
    printf("\n Enter the element which is to be searched");
    scanf("%d",&KEY);
    low=0;
    high=n-1;
    flag=BinSearch(A,KEY,low,high);
    printf("\n The element is at A[%d] location",flag);
    getch();
}
int BinSearch(int A[SIZE],int KEY,int low,int high)
{
 int m;
 m=(low+high)/2;       //mid of the array is obtained
 if(KEY==A[m])
        return m;
 else if(KEY<A[m])
        BinSearch(A,KEY,low,m-1);//search the left sub list
 else
        BinSearch(A,KEY,m+1,high);//search the right sub list
}
```

Output

```
How Many elements in an array? 5

Enter The Elements 10 20 30 40 50

Enter the element which is to be searched 20

The element is at A[1] location
```

Non Recursive Binary Search Program

```
/*****************************************************************
Implementation of non recursive Binary Search algorithm
*****************************************************************/
#include<stdio.h>
#include<conio.h>
#define SIZE 10
int n;
void main()
{
  int A[SIZE],KEY,i,flag;
  int BinSearch(int A[SIZE],int KEY);
  clrscr();
  printf("\n How Many elements in an array?");
  scanf("%d",&n);
  printf("\n Enter The Elements");
  for(i=0;i<n;i++)
       scanf("%d",&A[i]);
  printf("\n Enter the element which is to be searched");
  scanf("%d",&KEY);
  flag=BinSearch(A,KEY);
  if(flag==-1)
       printf("\n The Element is not present");
  else
       printf("\n The element is at A[%d] location",flag);
  getch();
}
int BinSearch(int A[SIZE],int KEY)
{
  int low,high,m;
  low=0;
  high=n-1;
  while(low<=high)
  {
            m=(low+high)/2;//mid of the array is obtained
            if(KEY==A[m])
                 return m;
            else if(KEY<A[m])
                 high=m-1;        //search the left sub list
            else
                 low=m+1;         //search the right sub list
       }
       return-1; //if element is not present in the list
}
```

Output (Run 1)

```
How Many elements in an array? 6

Enter The Elements
10
20
30
40
50
60

Enter the element which is to be searched 50

The element is at A[4] location
```

(Run 2)

```
How Many elements in an array? 5

Enter The Elements
10
20
30
40
50
Enter the element which is to be searched 80

The Element is not present
```

Sr. No.	Sequential technique	Binary search technique
1.	This is the simple technique of searching an element.	This is the efficient technique of searching an element.
2.	This technique does not require the list to be sorted.	This technique requires the list to be sorted. Then only this method is applicable.
3.	Every element of the list may get compared with the key element.	Only the mid element of the list is compared with key element.
4.	The worst case time complexity of this technique is O(n).	The worst case time complexity of this technique is O(log n).

Review Questions

1. *Explain the use of divide and conquer technique for binary search method. Give the algorithm for binary search method. What is the complexity of binary search method ?* **GTU : IT, Dec.-10, Marks 7**
2. *Explain binary search using divide and conquer method and compute its worst case running time.* **GTU : IT ,Dec.-11, Marks 7**
3. *What is divide and conquer technique? Give the use of it for binary searching method.* **GTU : IT, June-12, Marks 7**
4. *Explain Binary search algorithm with divide and conquer strategy and use the recurrence tree to show that the solution to the binary search recurrence $T(n) = T(n/2) + \Theta(1)$ is $T(n) = \Theta(\lg n)$.* **GTU : IT, Winter-14, Marks 7**

3.6 Max-Min Problem

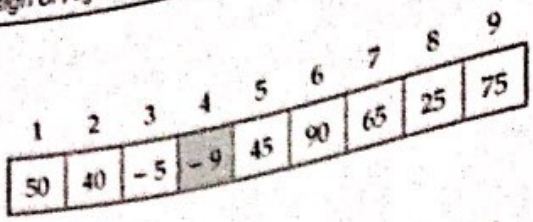
First of all we will discuss an algorithm of finding the minimum element from the set of n numbers.

```
Algorithm Minimum_val (A[1.....n])
{
// Problem Description : This algorithm is to
// find minimum value from array A of n
// elements
        min ← A[1] // Assuming first element as min.
for (i←2 to n) do
{
 if (min > A[i]) then
        min ← A[i] //set new min value
}
 return min
}
```

Thus we require total (n–1) comparison to obtain minimum value. Similarly we can obtain the maximum value from an array A in (n–1) comparison. Here is an algorithm for finding maximum element.

```
Algorithm Maximum_val (A[1.....n])
{
// Problem Description : This algorithm is to find maximum value from
// array A of n elements
      max ← A[1]
      for (i ← 2 to n) do
      {
            if [A[i] > max) then
                  max ← A[i]
      }
      return max
```

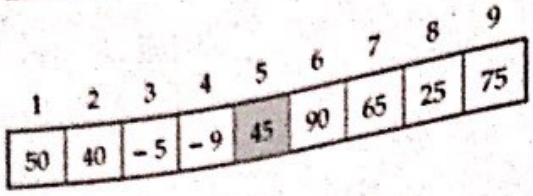
Step 4 :



Min = – 9

Max = 50

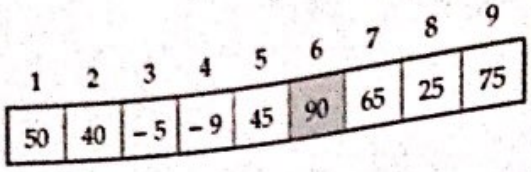
Step 5 :



Min = – 9

Max = 45

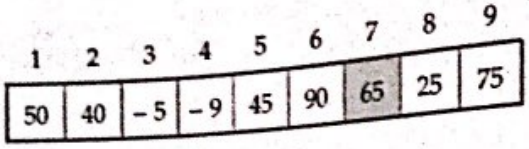
Step 6 :



Min = – 9

Max = 90

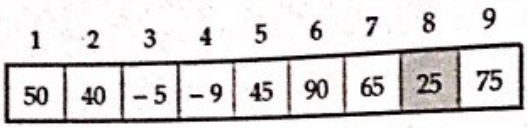
Step 7 :



Min = – 9

Max = 90

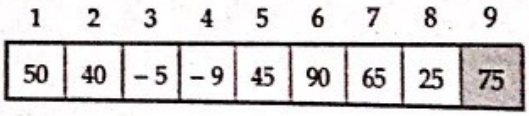
Step 8 :



Min = – 9

Max = 90

Step 9 :



Min = – 9

Max = 90

Step 10 :

As we have reached at n^{th} location of the array we will terminate the procedure of finding minimum and maximum values and print minimum value as – 9 and maximum value as 90.

Analysis

The above algorithm takes $\Theta(n)$ run n time. This is because the basic operation of comparing array element with Min or Max value is done within a **for** loop.

The above algorithm is a straightforward algorithm of finding minimum and maximum. But we can obtain them using recursive method. This recursive algorithm makes use of divide and conquer strategy.

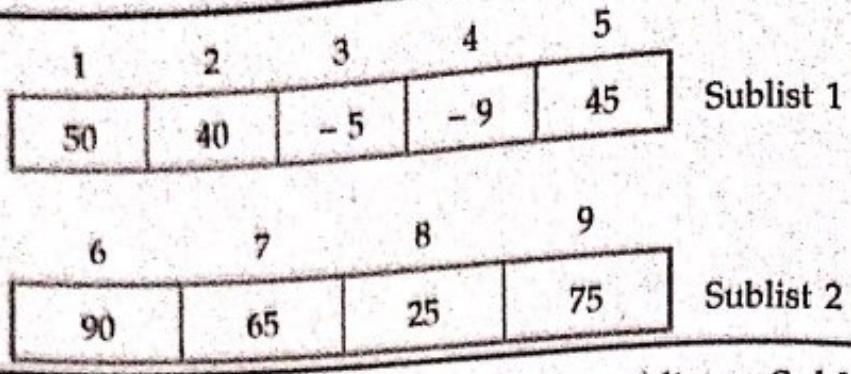
In this algorithm, the list of elements is divided at the mid in order to obtain two sublists. From both the sublist maximum and minimum elements are chosen. Two maxima and minima are compared and from them real maximum and minimum elements are determined. This process is carried out for entire list. The algorithm is as given below.

```
Algorithm Max_Min_Val (i, j, max, min)
//Problem Description : Finding min, max elements recursively.
//Input : i, j are integers used as index to an array A. The max and min will
//contain maximum and minimum value elements.
//Output : None
if (i = = j) then
{
    max ← A [i]
    min ← A [j]
}
else if (i = j–1) then
{
    if (A [i] < A [j]) then
    {
        max ← A [j]
        min ← A [i]
    }
    else
    {
        max ← A [i]
        min ← A [j]
    }           //end of else
}               //end of if
else
{
    mid ← (i+j) / 2                    //divide the list handling two lists separately
    Max Min Val (i, mid, max, min)
    Max Min val (mid + 1, j, max new, min new)
    if (max < max new) then
        max ← max new                  //combine solution
    if (min > min_new) then
        min ← min_new                  //combine solution
}
```

Example : Consider a list of some elements from which maximum and minimum element can be found out.

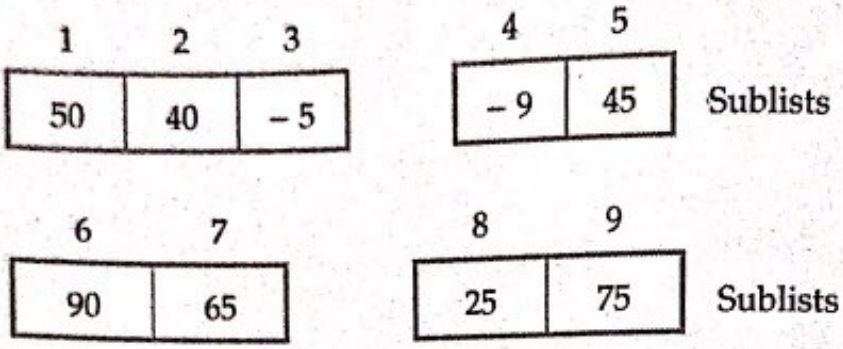
1	2	3	4	5	6	7	8	9
50	40	– 5	– 9	45	90	65	25	75

Step 1 :



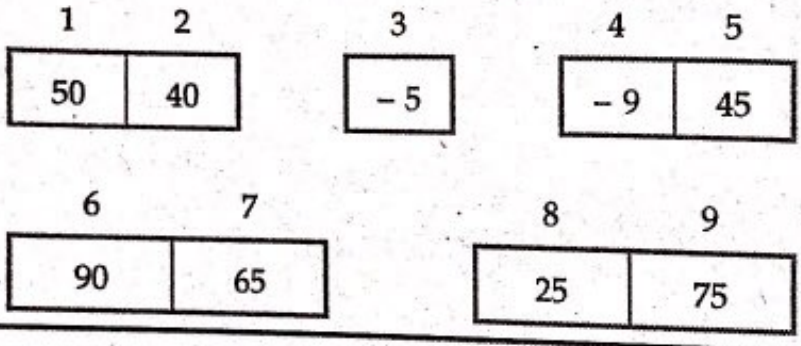
We have divided the original list at mid and two sublists : Sublist 1 and sublist 2 are created. We will find min and max values respectively from each sublist.

Step 2 :



Again divide each sublist and create further sublists. Then from each sublist obtain min and max values.

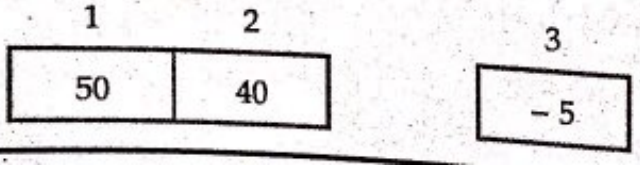
Step 3 :



It is possible to divide the list (50, 40, – 5) further. Hence we have divided the list into sublists and min, max values are obtained.

Step 4 :

Now further division of the list is not possible. Hence we start combining the solutions of min and max values from each sublist.



Combine (1, 2) and (3) Min = – 5, Max = 50

1	2	3
50	40	– 5

4	5
– 9	45

Combine (1, ... 3) and (4, 5) Min = – 9, Max = 50

Now we will combine (1, ... 3) and (4, 5) and the min and max values among them are obtained. Hence,

min value = – 9

max value = 50

Step 5 :

6	7
90	65

8	9
25	75

Combine (6, 7) and (8, 9) Min = 25, Max = 90

Step 6 :

1	2	3	4	5
50	40	– 5	– 9	45

6	7	8	9
90	65	25	75

Combine the sublists (1, ... 5) and (6, ... 9). Find out min and max values which are

min = – 9 max = 90

Thus the complete list is formed from which the min and max values are obtained. Hence final min and max values are

min = – 9 and max = 90

The tree for recursive calls will be -

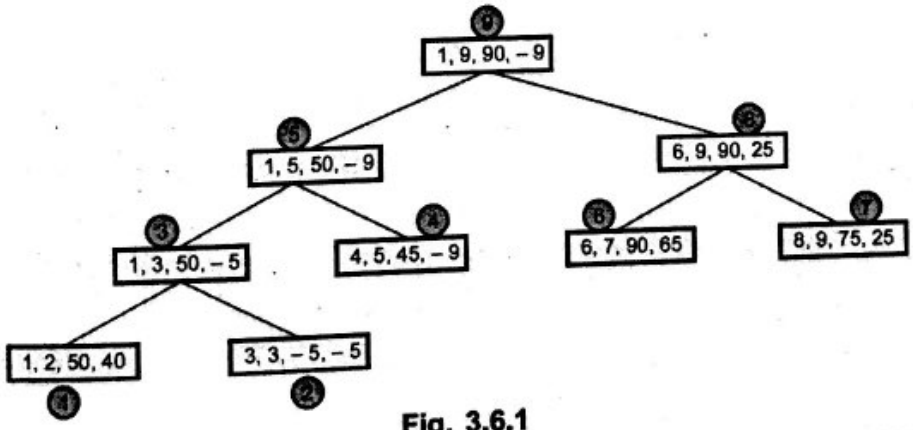


Fig. 3.6.1

Analysis

There are two recursive calls made in this algorithm, for each half divided sublist.

Hence, time required for computing min and max will be

$$T(n) = T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + 2 \quad \text{when } n > 2$$

$$T(n) = 1 \quad \text{when } n = 2$$

If single element is present in the list then $T(n) = 0$.

Now time required for computing min and max will be -

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

$$= 2[2T(n/4) + 2] + 2$$

$$= 2(2[2T(n/8) + 2] + 2) + 2 = 8T(n/8) + 10$$

Continuing in this fashion a recursive equation can be obtained. If we put $n = 2^k$ then

$$T(n) = 2^{k-1} T(2) + \sum_{i=1}^{k-1} 2^i = 2^{k-1} + 2^k - 2$$

$$T(n) = 3n/2 - 2$$

Neglecting the order of magnitude, we can declare the time complexity is O(n).

Example 3.6.1 *Consider the following algorithm :*

```
ALGORITHM Secret (A[0, ..., n–1])
//Input : An array A[0, ..., n–1] of n real numbers
    minval ← A[0];
    maxval ← A[0];
    for i ← 1 to n–1 do
        if A[i] < minval
            minval ← A[i];
        if A[i] > maxval
            maxval ← A[i];
        return maxval-minval;
```

i) What does this algorithm compute ? ii) What is the basic operation?

iii) How many times is the basic operation computed ?

iv) What is the efficiency class of this algorithm ?

v) Suggest an improvement or a better algorithm altogether and indicate the efficiency class. If you can't do it, prove that it can't be done.

AU : Dec.-08, Marks 16

Solution :

i) This algorithm finds the maximum and minimum element from given sequence of elements.

ii) The basic operation is to compare the array elements for finding minval and maxval.

iii) The basic operation is computed for n times inside a for loop.

iv) This algorithm has the time complexity of **O(n)**. The efficiency class for this algorithm is linear.

v) This algorithm can be improved by using divide an conquer the pseudo code for this is as given below -

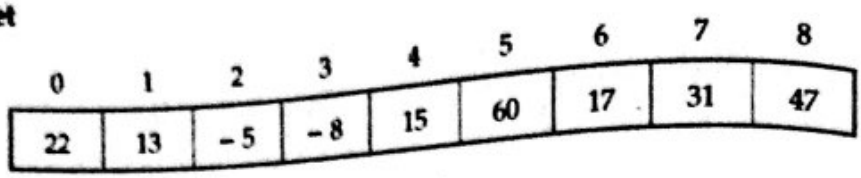
```
Algorithm Max_Min_Val (i, j, max, min)
//Problem Description : Finding min, max elements recursively.
//Input : i, j are integers used as index to an array A. The max and min will
//contain maximum and minimum value elements.
//Output : None
if (i = = j) then
{
        max ← A [i]
        min ← A [j]

}
else if (i = j–1) then
{
        if (A [i] < A [j]) then
        {
        max ← A [j]
        min ← A [i]
    }
    else
    {
        max ← A [i]
        min ← A [j]
    }           //end of else
                //end of if
}
else
{
        mid ← (i+j) / 2                 //divide the list handling two lists separately
        Max_Min_Val (i, mid, max, min)
        Max_Min_val (mid + 1, j, max_new, min_new)
        if (max < max_new) then
        max ← max_new                   //combine solution
    if (min > min_new) then
                min ← min_new  //combine solution
}
```

The efficiency class for this algorithm is linear i.e. **O(n)**.

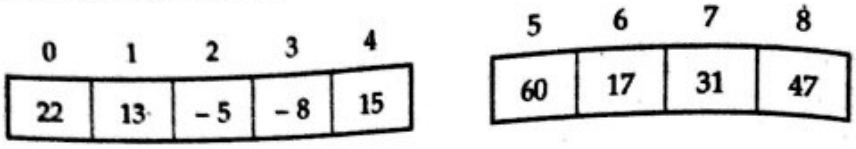
Example 3.6.2 *For the following list of elements trace the recursive algorithm for finding max and min and determine how many comparisons have been made.* **AU : Dec.-10, Marks 8**

22, 13, – 5, – 8, 15, 60, 17, 31, 47.

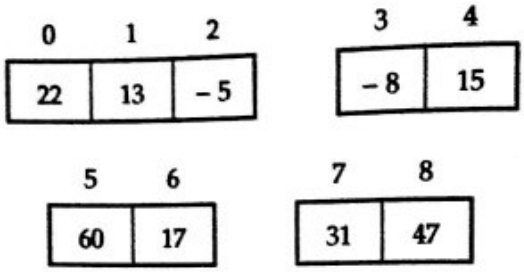
Solution : Let



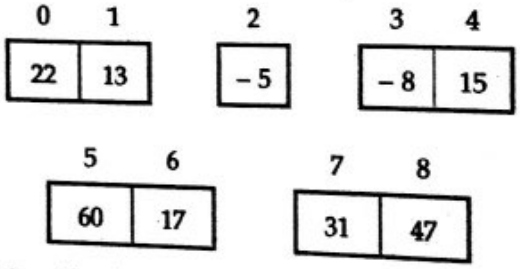
Step 1 : Divide the list into sub-lists



Step 2 : Further divide the sublists



Step 3 : Divide the list (0 .... 2) further into sublists



Step 4 : We will combine list (0 ... 1) and (2).

The min = – 5, max = 22 Combine the list (0 .... 2) and (3, 4)

min = – 8 and max = 22

Step 5 : Combine (5, 6) and 7, 8)

min = 17, max = 60

Step 6 : Combine (0 ... 4) and (5 ... 8)

min = – 8, max = 60

Review Question

1. *Using the divide and conquer approach to find the maximum and minimum in a set of 'n' elements. Also find the recurrence relation for the number of elements compared and solve the same.*

3.7 Merge Sort

GTU : Winter-14, Marks 7

The merge sort is a sorting algorithm that uses the divide and conquer 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 elements each.

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

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

Example : Consider the elements as

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

Now we will split this list into two sublists.

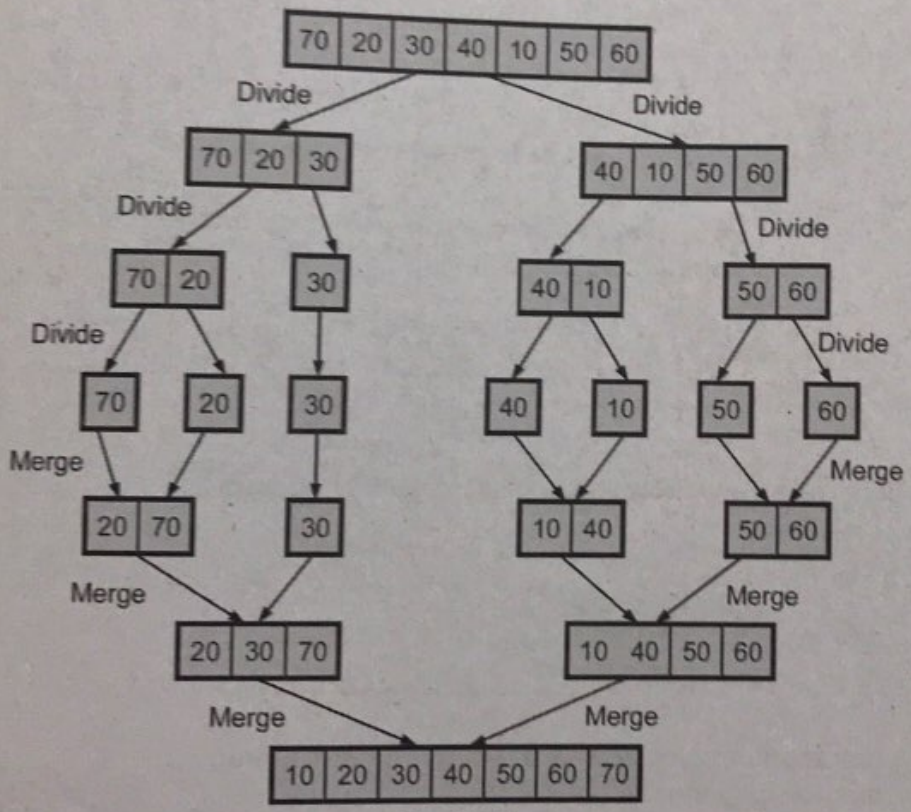


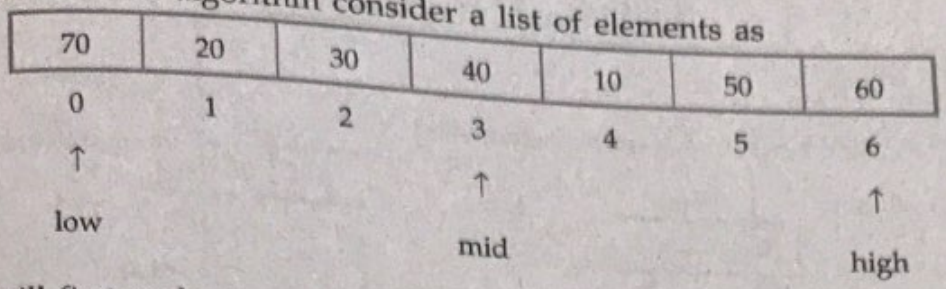
Fig. 3.7.1

Algorithm MergeSort(int A[0...n–1],low,high)
Problem Description: This algorithm is for sorting the elements using merge sort
Input: Array A of unsorted elements, low as beginning

```
//pointer of array A and high as end pointer of array A
//Output: Sorted array A[0...n - 1]
        if(low < high)then
        {
            mid ← (low+high)/2              // split the list at mid
            MergeSort(A,low,mid)            // first sublist
            MergeSort(A,mid+1,high) // second sublist
            Combine(A,low,mid,high) // merging of two sublists
        }
Algorithm Combine(A[0...n-1],low, mid, high)
{
        k ← low;   // k as index for array temp
        i ← low;          // i as index for left sublist of array A
        j ← mid+1 // j as index for right sublist of array A
        while(i <= mid and j <= high)do
        {
            if(A[i]<=A[j])then
        // if smaller element is present in left sublist
            {
                    // copy that smaller element to temp array
                    temp[k] ← A[i]
                    i ← i+1
                    k ← k+1
            }
            else  //smaller element is present in right sublist
            {
                //copy that smaller element to temp array
                temp[k] ← A[j]
                j ← j+1
                k ← k+1
            }
        }

    //copy remaining elements of left  sublist to temp
    while(i<=mid)do
    {
                temp[k] ← A[i]
                i ← i+1
                k ← k+1
    }
    //copy remaining elements of right sublist to temp
    while(j<=high)do
    {
            temp[k] ← A[j]
            j ← j+1
            k ← k+1
    }
```

Logic Explanation

To understand above algorithm consider a list of elements as



Then we will first make two sublists as

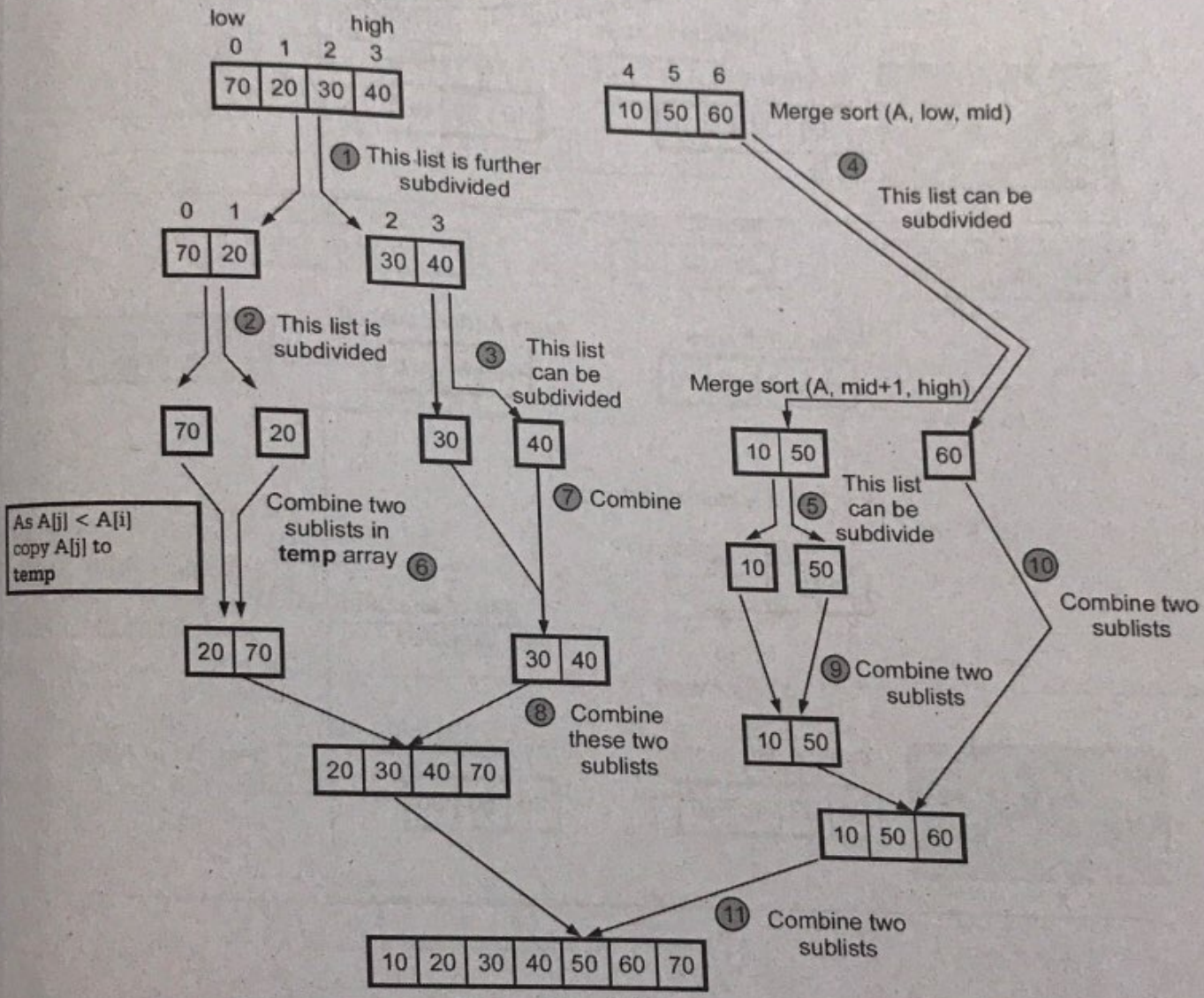
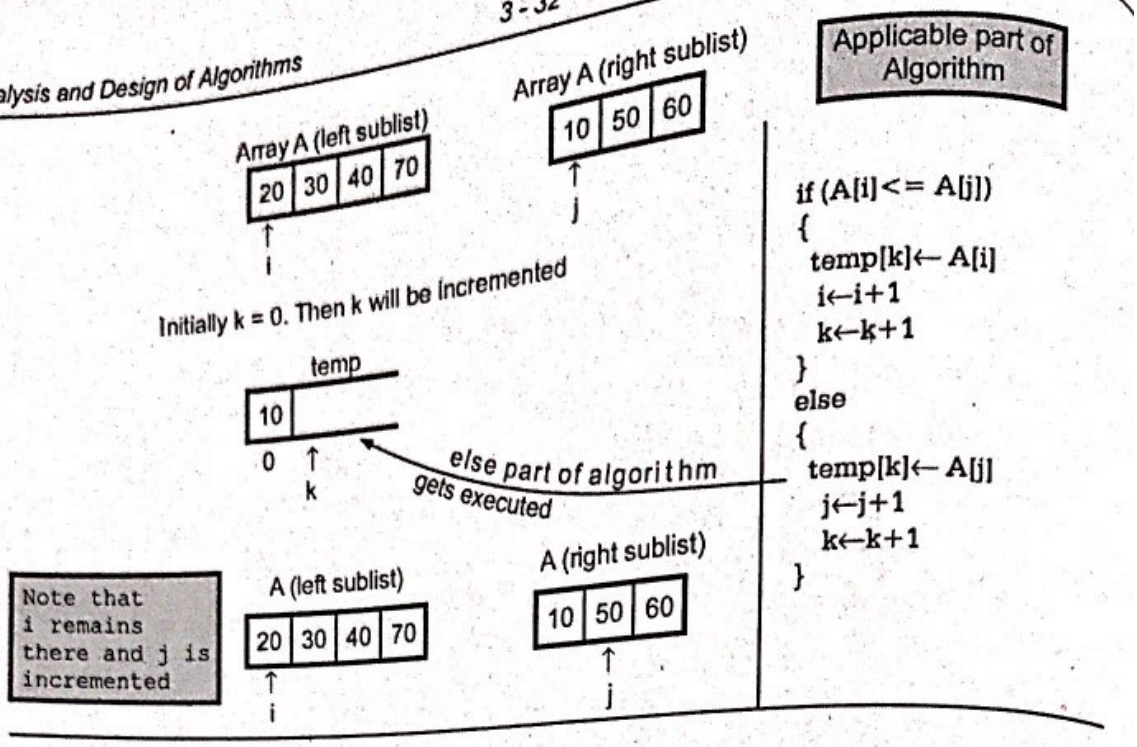
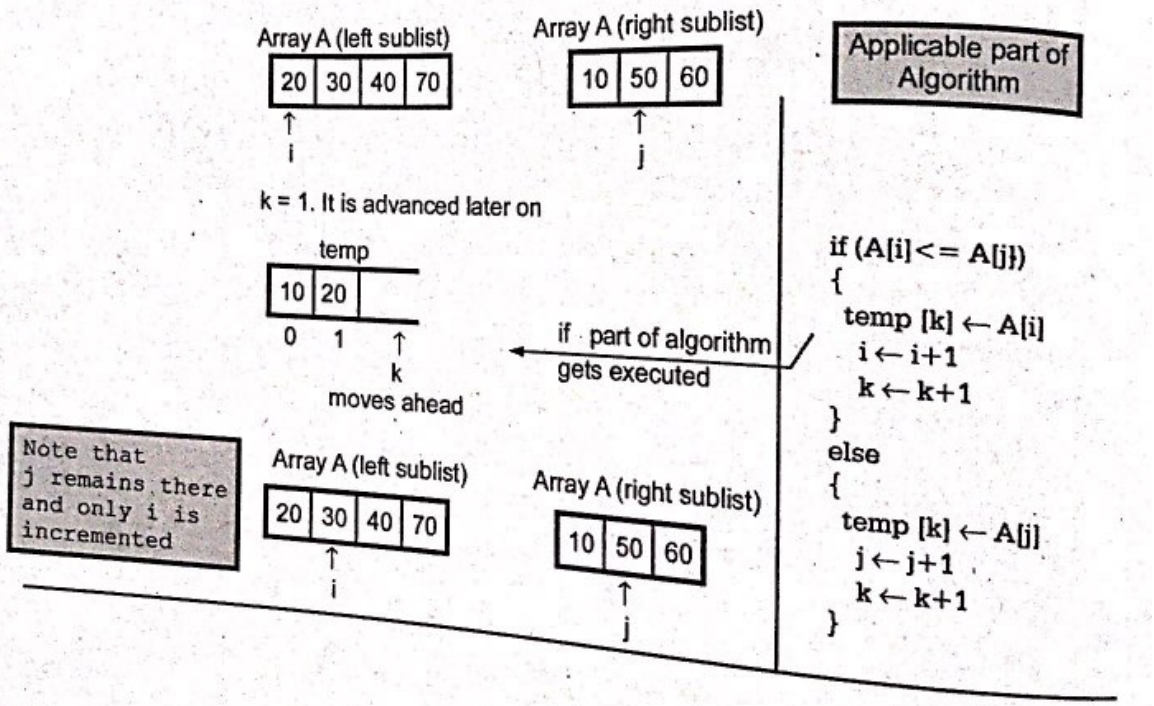


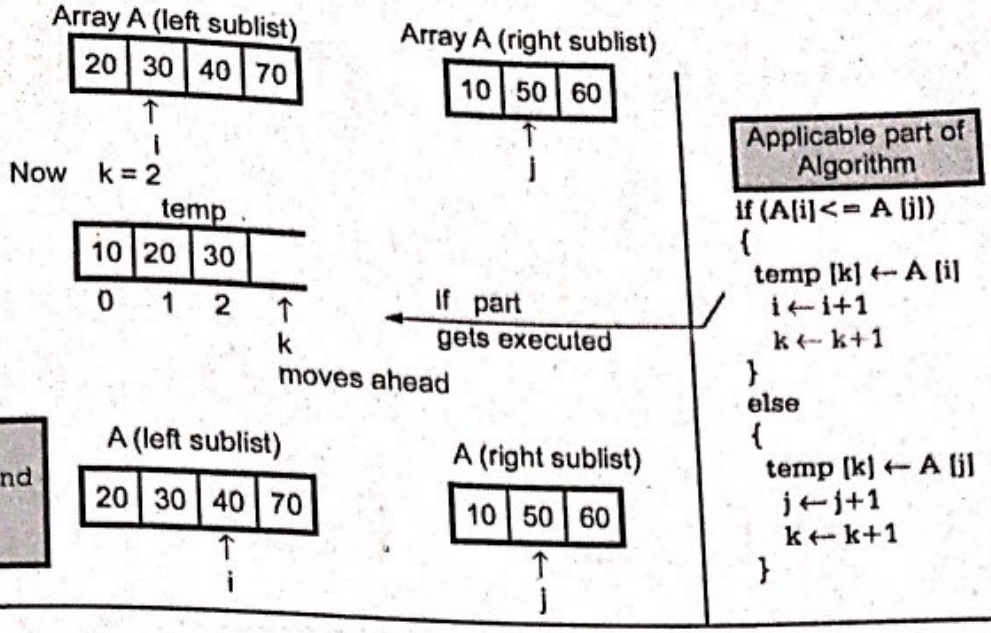
Fig. 3.7.2

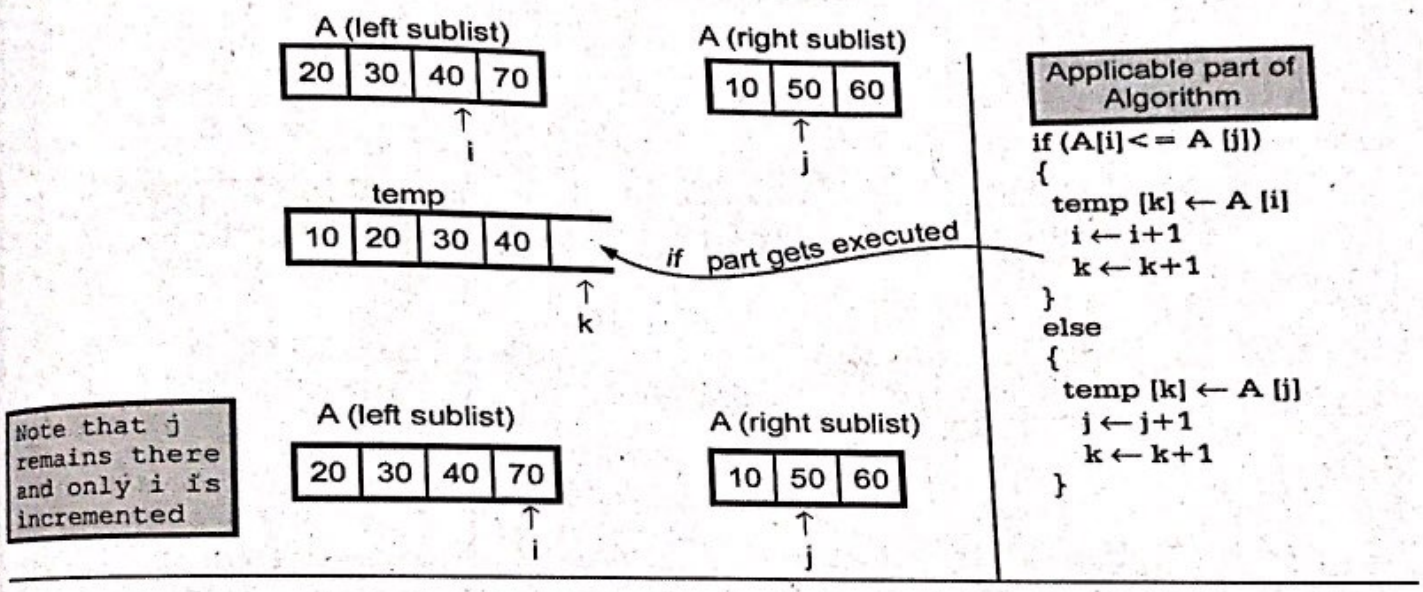
Let us see the **combine** operation more closely with the help of some example.

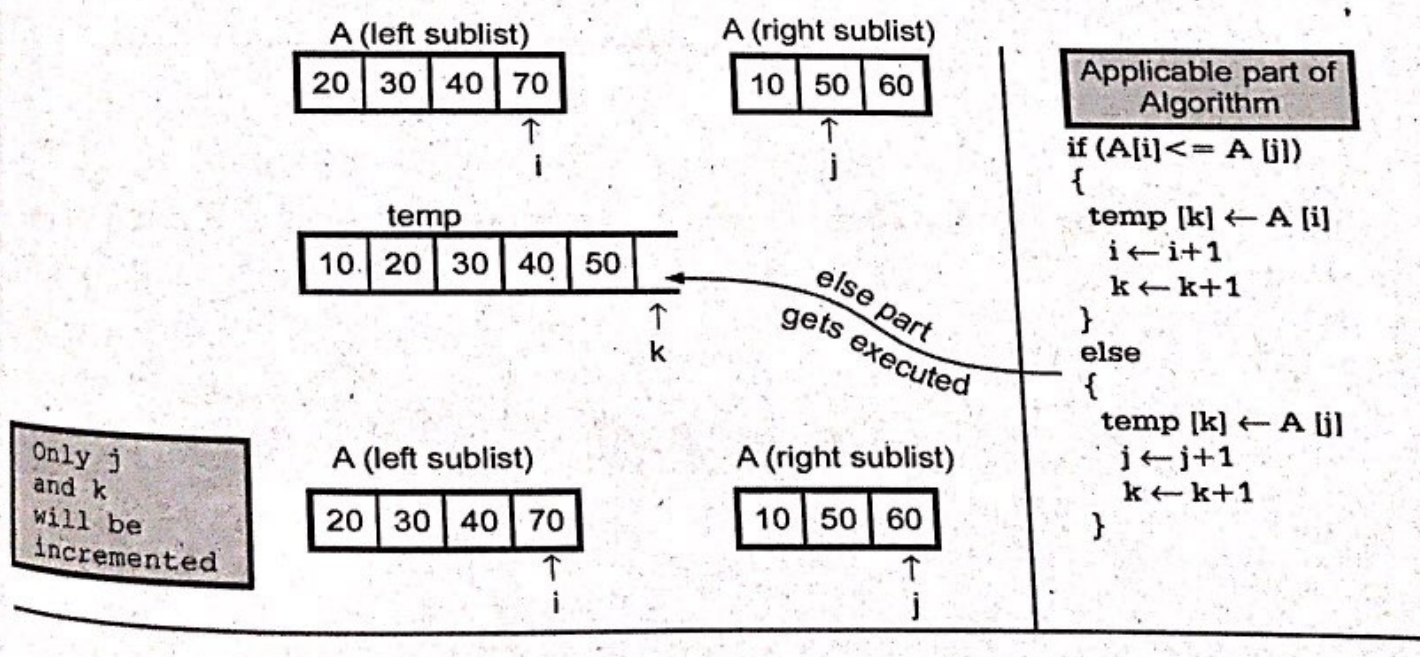
Consider that at some instance we have got two sublist 20, 30, 40, 70 and 10, 50, 60, then

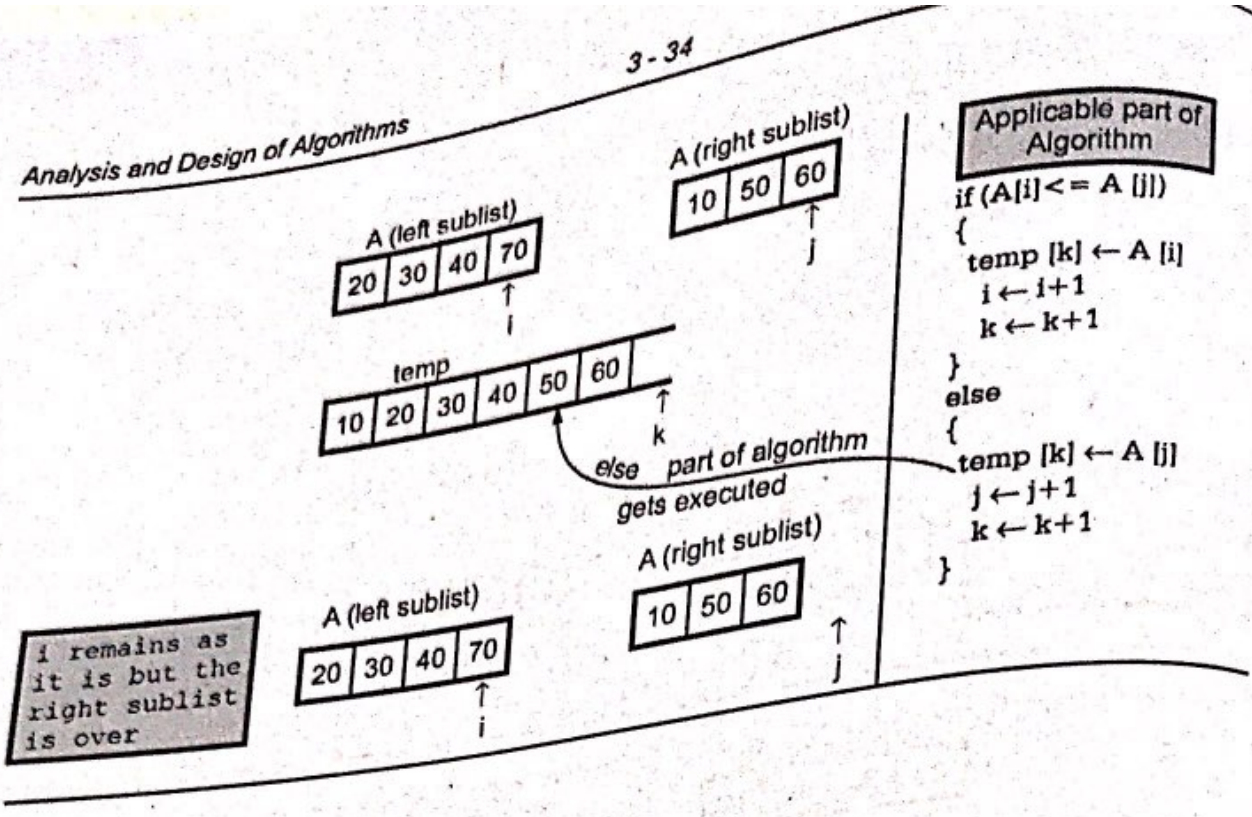


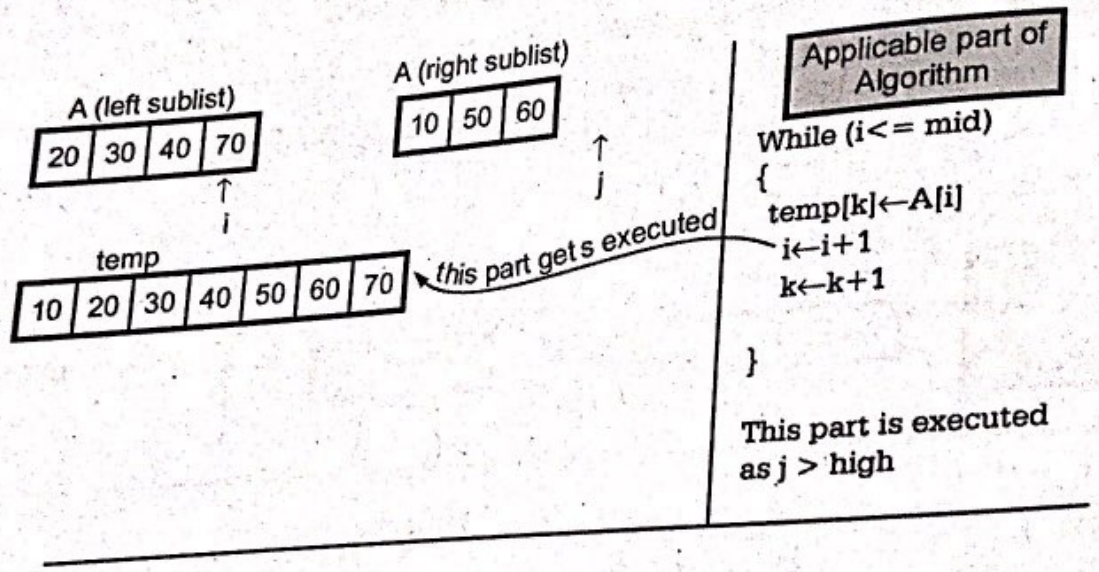












Finally we will copy all the elements of array **temp** *to array* **A.** *Thus array* **A** *contains sorted list.*

A

10	20	30	40	50	60	70

Analysis

In merge sort algorithm the two recursive calls are made. Each recursive call focuses on n/2 elements of the list. After two recursive calls one call is made to combine two sublists i.e. to merge all n elements. Hence we can write recurrence relation as

$$T(n) = \underbrace{T(n/2)}_{\text{Time taken by left sublist to get sorted}} + \underbrace{T(n/2)}_{\text{Time taken by right sublist to get sorted}} + \underbrace{cn}_{\text{Time taken for combining two sublists}}$$

where $n > 1$ $T(1) = 0$

We can obtain the time complexity of Merge Sort using two methods

1. Master theorem 2. Substitution method

1. Using master theorem :

Let, the recurrence relation for merge sort is

$$T(n) = T(n/2) + T(n/2) + cn$$

i.e. $$T(n) = 2T(n/2) + cn \quad \text{... (3.7.1)}$$

$$T(1) = 0 \quad \text{... (3.7.2)}$$

As per Master theorem

$$T(n) = \Theta(n^d \log n) \quad \text{if } a = b$$

As in equation (3.6.1),

$a = 2$, $b = 2$ and $f(n) = cn$ i.e. n^d with $d = 1$.

and $$a = b^d$$

i.e. $$2 = 2^1$$

This case gives us

$$T(n) = \Theta(n \log_2 n)$$

2. Using substitution method :

Let, the recurrence relation for Merge Sort be

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

$$T(n) = 2T(n/2) + cn \quad \text{... (3.7.3)}$$

i.e.

$$T(1) = 0 \quad \text{... (3.7.4)}$$

Let us apply substitution on equation (3.6.3). Assume $n = 2^k$.

$$T(n) = 2T(n/2) + cn$$

$$T(n) = 2T\left(\frac{2^k}{2}\right) + c \cdot 2^k \quad \text{... } \because n = 2^k$$

$$T(2^k) = 2T(2^{k-1}) + c \cdot 2^k$$

It we put $k = k - 1$ then,

$$T(2^k) = \underbrace{2T(2^{k-1}) + c \cdot 2^k}_{\downarrow}$$

$$= \overbrace{2[2T(2^{k-2}) + c \cdot 2^{k-1}]} + c \cdot 2^k$$

$$= 2^2 T(2^{k-2}) + 2 \cdot c \cdot 2^{k-1} + c \cdot 2^k$$

$$= 2^2 T(2^{k-2}) + 2 \cdot c \cdot \frac{2^k}{2} + c \cdot 2^k$$

$$= 2^2 T(2^{k-2}) + c \cdot 2^k + c \cdot 2^k$$

$$T(2^k) = 2^2 T(2^{k-2}) + 2c \cdot 2^k$$

Similarly we can write,

$$T(2^k) = 2^3 T(2^{k-3}) + 3c \cdot 2^k$$

$$= 2^4 T(2^{k-4}) + 4c \cdot 2^k$$

...

...

...

$$= 2^k T(2^{k-k}) + k \cdot c \cdot 2^k$$

$$= 2^k T(2^0) + k \cdot c \cdot 2^k$$

$$T(2^k) = 2^k T(1) + k \cdot c \cdot 2^k$$

But as per equation (3.6.4), $T(1) = 0$,

∴ Equation (3.6.5) becomes,

$$T(2^k) = 2^k \cdot 0 + k \cdot c \cdot 2^k$$

$$T(2^k) = k \cdot c \cdot 2^k$$

But we assumed $n = 2^k$, taking logarithm on both sides.

i.e. $\log_2 n = k$

$\therefore T(n) = \log_2 n \cdot cn$

$\therefore T(n) = \Theta(n \cdot \log_2 n)$

Hence the average and worst case time complexity of merge sort is $\Theta(n \log_2 n)$.

Time complexity of merge sort

Best case	Average case	Worst case
$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$

Example 3.7.1 *Is merge sort stable sorting algorithm ?* **Dec.-10, Marks 2**

Solution : Yes, merge sort is the stable sorting algorithm. A sorting algorithm is said t be stable if it preserves the ordering of similar (equal) elements after applying sortin method. And merge sort is a method which preserves this kind of ordering. Henc merge sort is a stable sorting algorithm.

Example 3.7.2 *Give the time efficiency and drawback of merge sort algorithm.*

Solution : Time efficiency : The best, worst and average case time complexity of merg sort is O (n log n).

Drawbacks :

i) This algorithm requires extra storage to execute this method.

ii) This method is slower than the quick sort method.

iii) This method is complicated to code.

C Program

```
/*****************************************************************
This program is for Merge Sort using divide and conquer strategy.
*****************************************************************/
#include<conio.h>
```

```
Enter list elements :
70
20
30
40
10
50
60

The Sorted Array Is ...
10    20    30    40    50    60    70
```

Review Question

1. *Explain merge sort problem using divide and conquer technique. Give an example.*

GTU : Winter-14, Marks 7

3.8 Quick Sort

GTU : Dec.-10,11, June-11,12, Summer-13,14, Winter-14,15, Marks 7

Quick sort is a sorting algorithm that uses the divide and conquer strategy. In this method division is dynamically carried out. The three steps of quick sort are as follows :

- **Divide :** Split the array into two sub arrays that each element in the left sub array is less than or equal the middle element and each element in the right sub array is greater than the middle element. The splitting of the array into two sub arrays is based on pivot element. All the elements that are less than pivot should be in left sub array and all the elements that are more than pivot should be in right sub array.

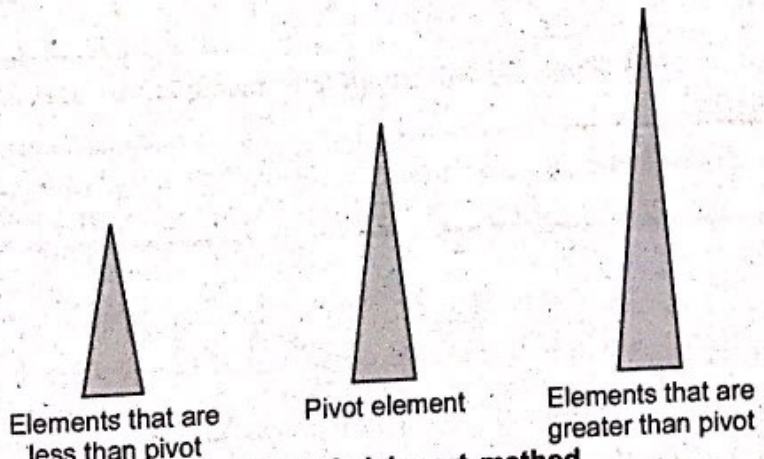
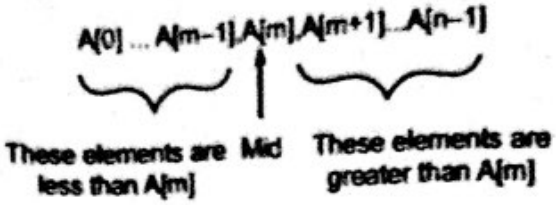


Fig. 3.8.1 Quick sort method

- **Conquer :** Recursively sort the two sub arrays.
- **Combine :** Combine all the sorted elements in a group to form a list of sorted elements.

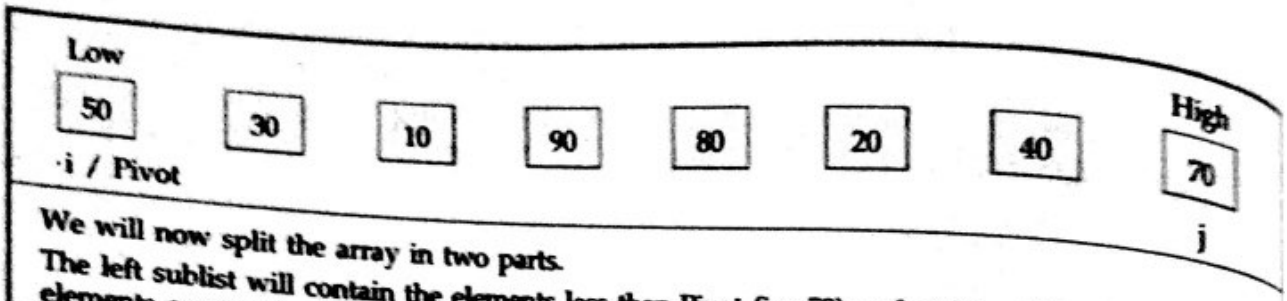
In merge sort the division of array is based on the positions of array elements, but in quick sort this division is based on actual value of the element. Consider an array A[i] where i is ranging from 0 to n – 1 then we can formulate the division of array elements as

Let us understand this algorithm with the help of some example.



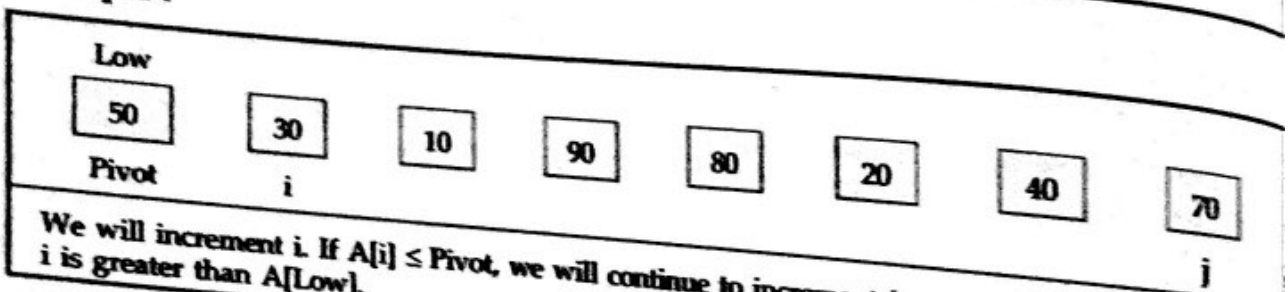
Example :

Step 1 :



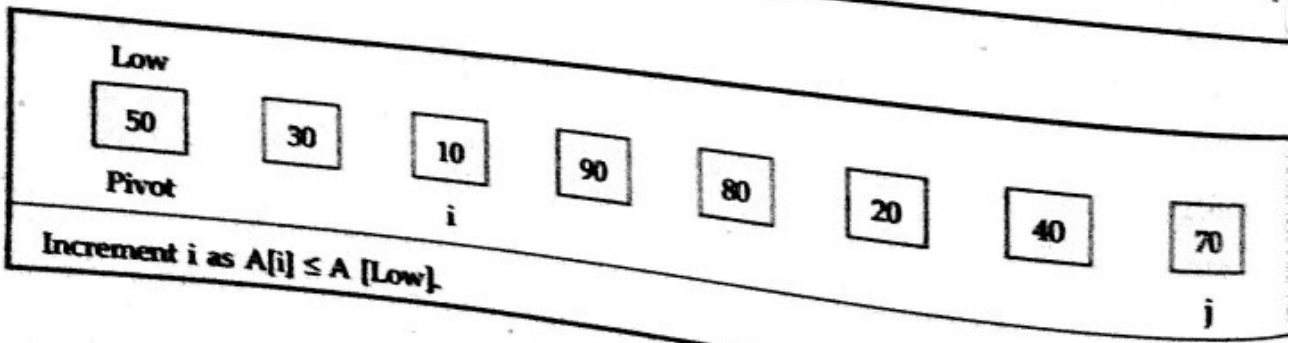
We will now split the array in two parts.
The left sublist will contain the elements less than Pivot (i.e. 50) and right sublist contains elements greater than Pivot.

Step 2 :



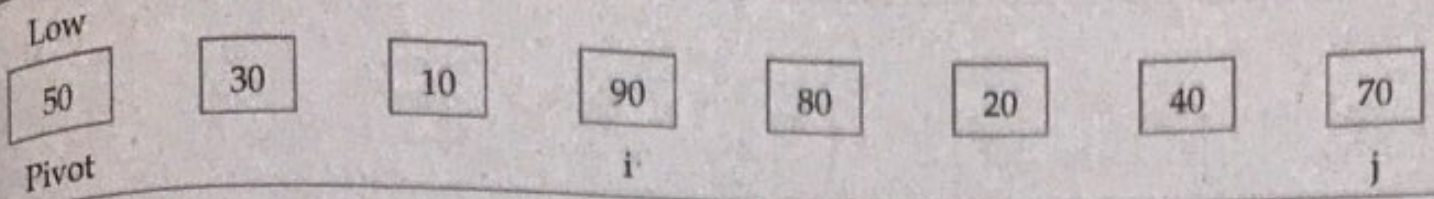
We will increment i. If A[i] ≤ Pivot, we will continue to increment it until the element pointed by i is greater than A[Low].

Step 3 :



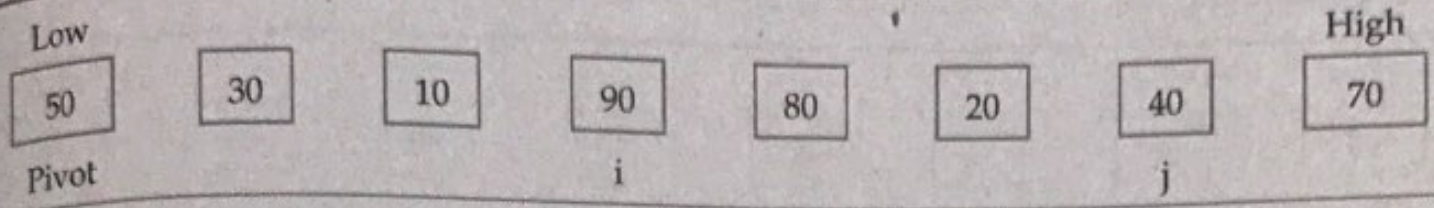
Increment i as A[i] ≤ A [Low].

Step



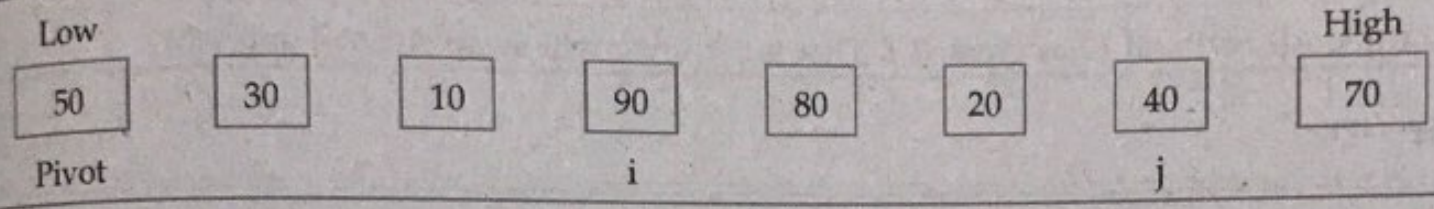
As A[i] > A [Low], we will stop incrementing i.

Step 5 :



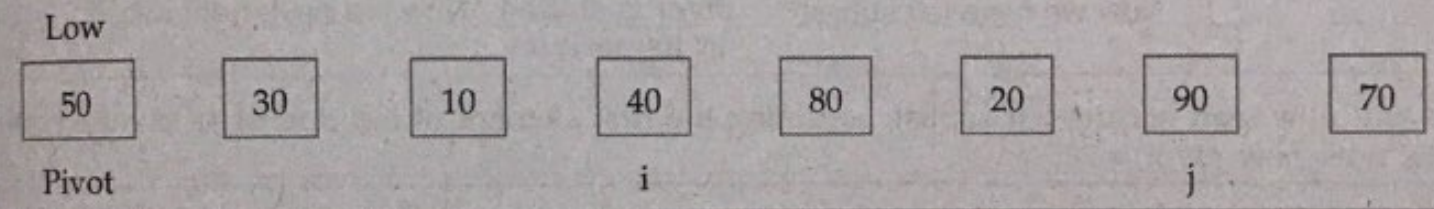
As A[j] > Pivot (i.e. 70 > 50). We will decrement j. We will continue to decrement j until the element pointed by j is less than A [Low].

Step 6 :



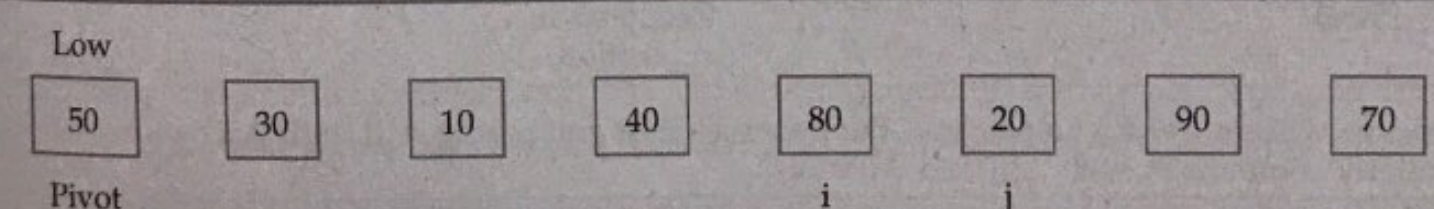
Now we can not decrement j because 40 < 50. Hence we will swap A[i] and A[j] i.e. 90 and 40.

Step 7 :

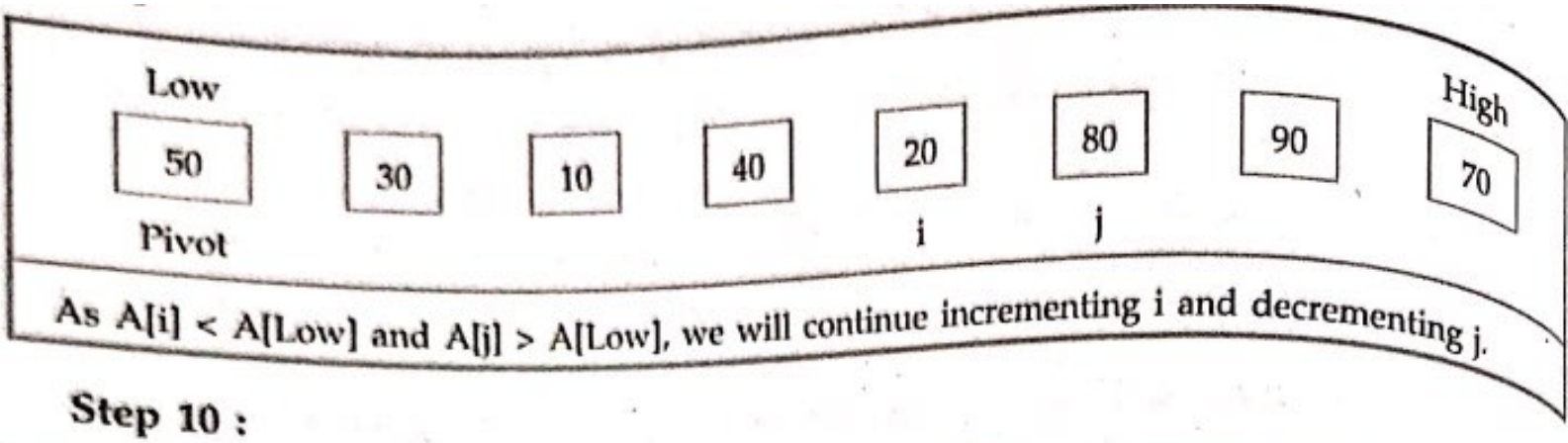


As A[i] is less than A[Low] and A[j] is greater than A[Low] we will continue incrementing i and decrementing j, until the false conditions are obtained.

Step 8 :

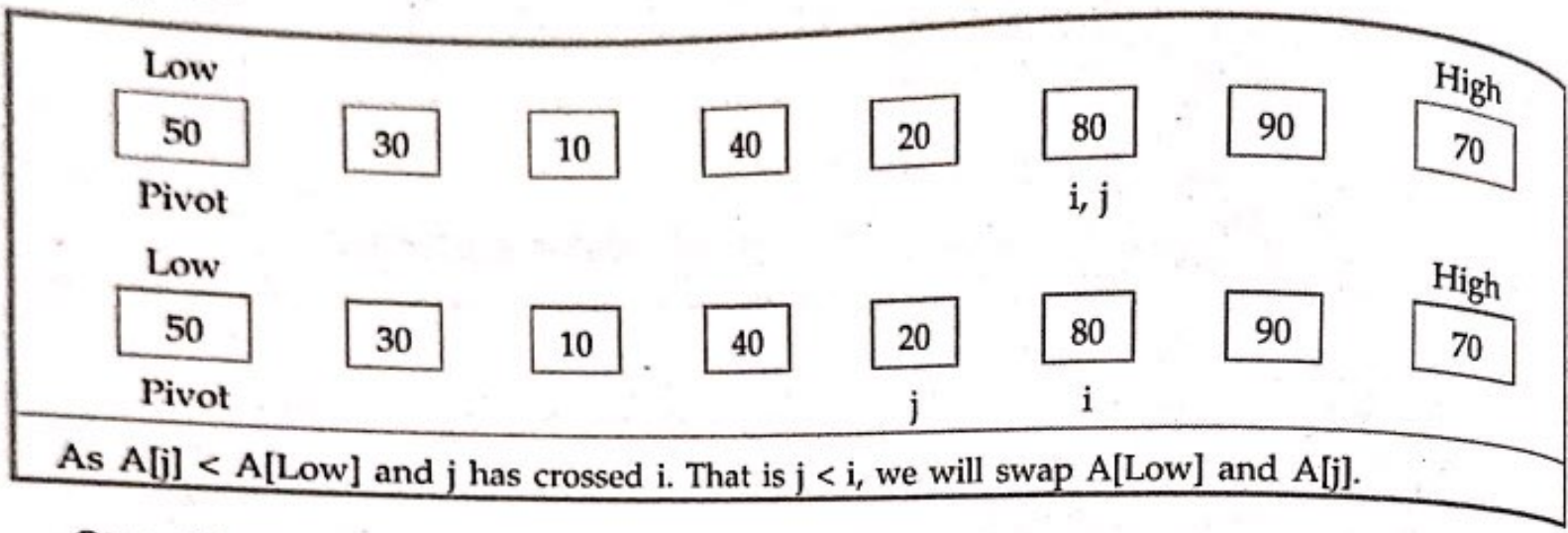


We will stop incrementing i and stop decrementing j. As i is smaller than j we will swap 80 and 20.



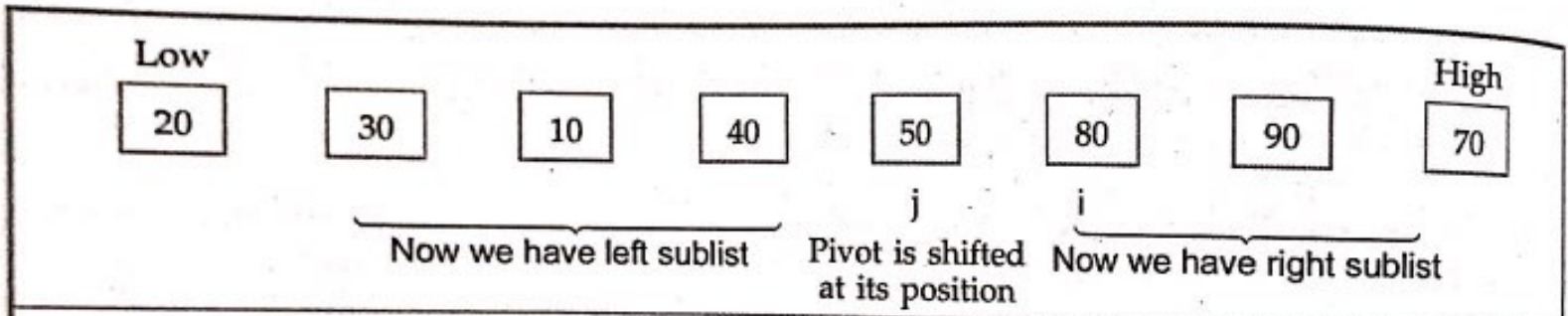
As A[i] < A[Low] and A[j] > A[Low], we will continue incrementing i and decrementing j.

Step 10 :



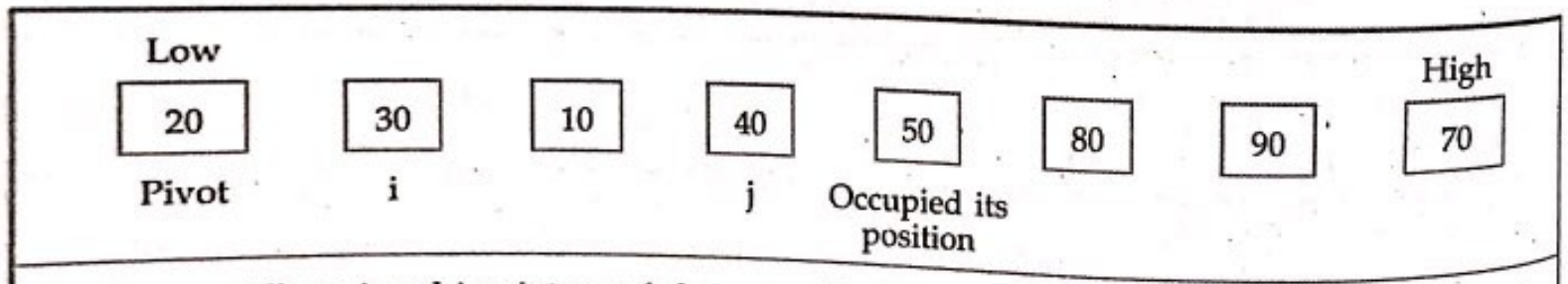
As A[j] < A[Low] and j has crossed i. That is j < i, we will swap A[Low] and A[j].

Step 11 :

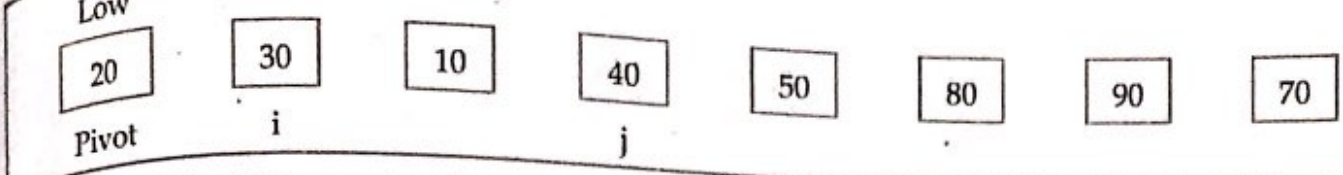


We will now start sorting left sublist, assuming the first element of left sublist as pivot element. Thus now new pivot = 20.

Step 12 :

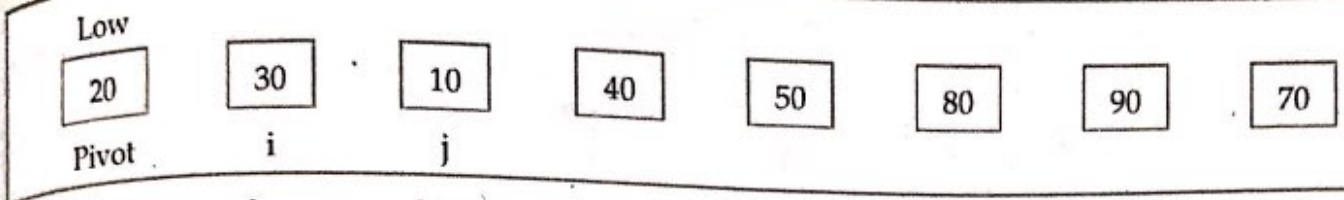


Now we will set i and j pointer and then we will start comparing A[i] with A[Low] or A[Pivot]. Similarly comparison with A[j] and A[Pivot].



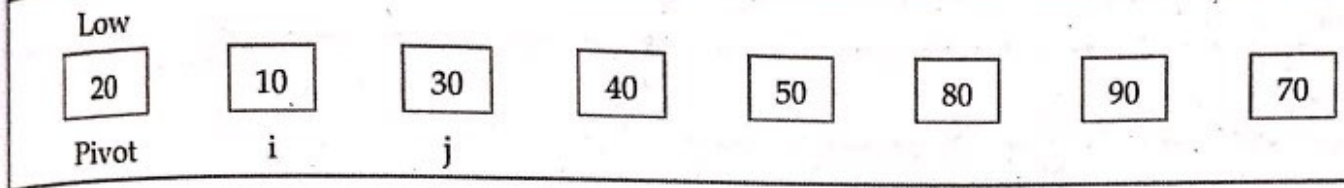
As A[i] > A[Pivot], hence stop incrementing i. Now as A[j] > A[Pivot], hence decrement j.

Step 14 :



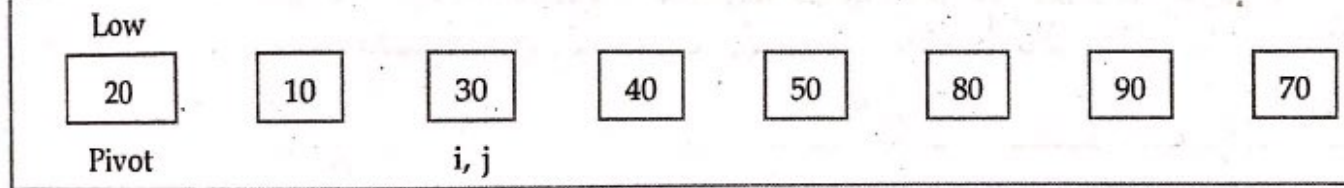
Now j cannot be decremented because 10 < 20. Hence we will swap A[i] and A[j].

Step 15 :



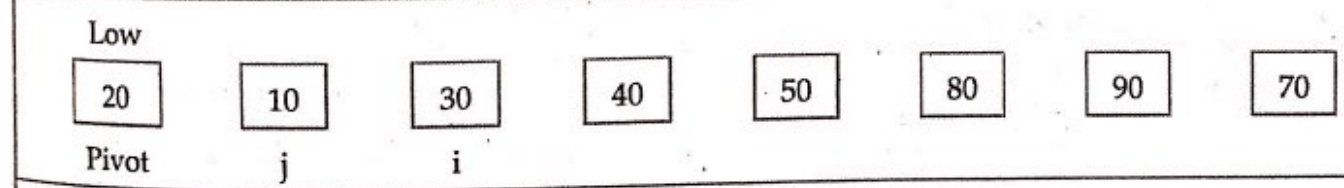
As A[i] < A[Low], increment i.

Step 16 :



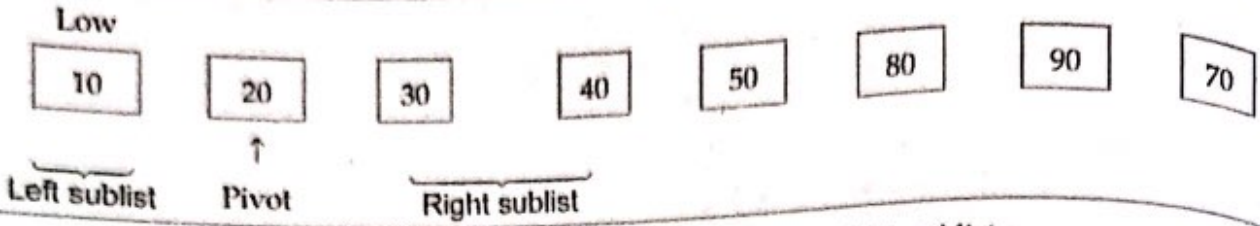
Now as A[i] > A[Low], or A[j] > A[Pivot] decrement j.

Step 17 :



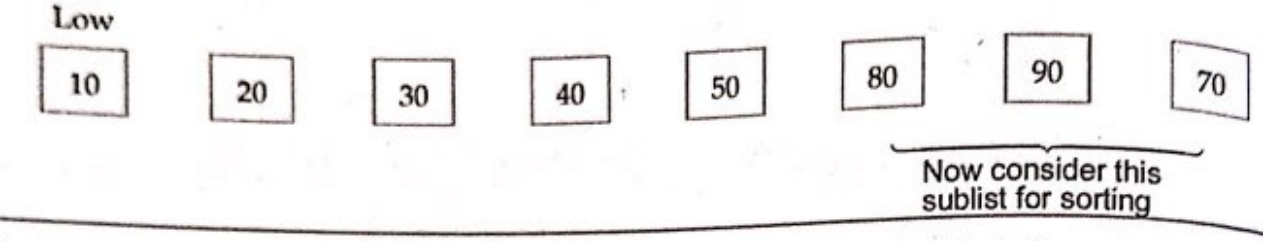
As A[j] < A[Low] we cannot decrement j now. We will now swap A[Low] and A[j] as j has crossed i and i > j.

Step 18 :



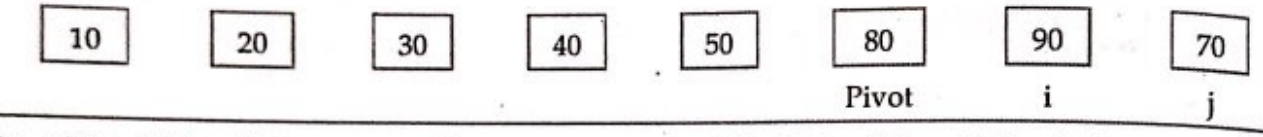
As there is only one element in left sublist hence we will sort right sublist.

Step 19 :



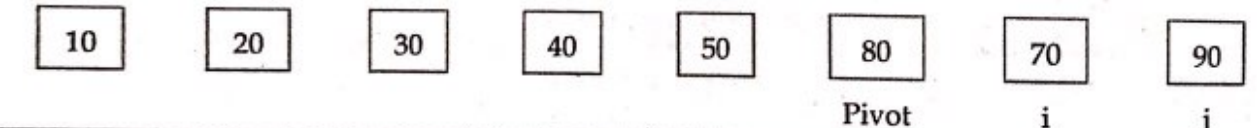
As left sublist is sorted completely we will sort right sublist, assuming first element of right sublist as Pivot.

Step 20 :



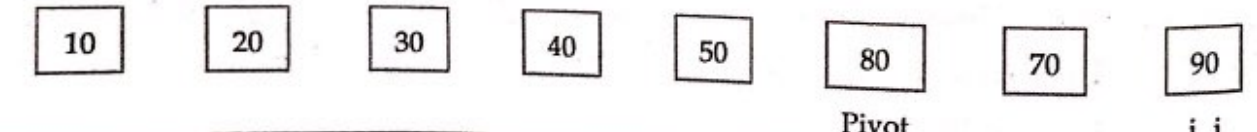
As A[i] > A[Pivot], hence we will stop incrementing i. Similarly A[j] < A[Pivot]. Hence we stop decrementing j. Swap A[i] and A[j].

Step 21 :



As A[i] < A[Pivot], increment i.

Step 22 :



As A[i] > A[Pivot], decrement j.

10	20	30	40	50	70	80	90

Pivot

The left sublist now contains 70 and right sublist contains only 90. We cannot further subdivide the list.

Hence list is

10	20	30	40	50	70	80	90

This is a sorted list.

Example 3.8.1 *Write the quick sort algorithm. Trace the same on data set - 4, 3, 1, 9, 8, 2, 4, 7.*

GTU : Summer-13, Marks 8

Solution : Consider the elements

4	3	1	9	8	2	4	7

Step 1 : We will assume first element as pivot. From second element onwards, we will compare all the elements with pivot value (i.e .4). All the elements less than pivot will occupy left sublist and all the elements greater than pivot will occupy right sublist.

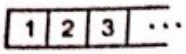
Thus pivot will occupy its proper position

3	1	2	4	4	9	8	7

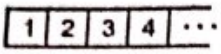
Left sublist Right sublist

Step 2 : The above procedure will be repeated for each sublists recursively and newer sub-sub lists will be generated.

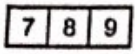
Step 3 :



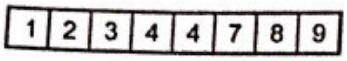
Step 4 :



Step 5 :



By combining each sorted sublist we get -

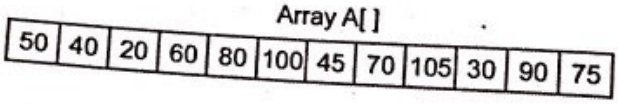


Sorted list

Example 3.8.2 *Sort the following list using quick sort algorithm : <50, 40, 20, 60, 80, 100, 45, 70, 105, 30, 90, 75>. Also discuss worst and best case of quick sort algorithm.*

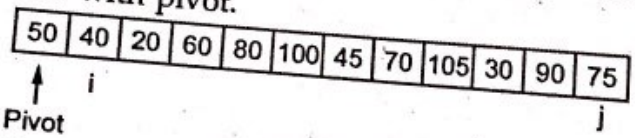
GTU : Summer-14, Marks 7

Solution : Consider the list in an array

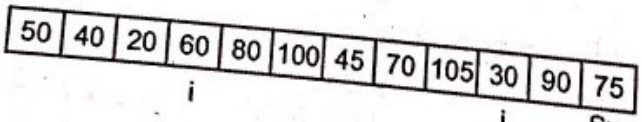


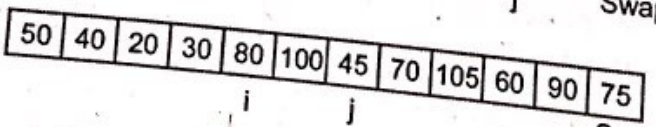
Step 1 :

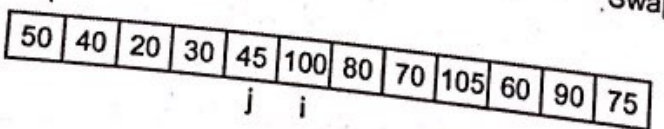
We will assume first element as pivot element. Second element onwards we will compare the elements with pivot.

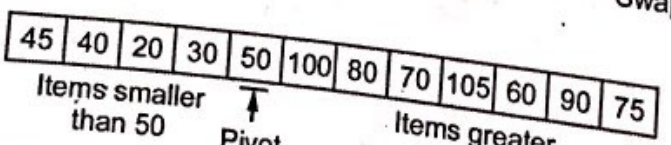


If A[i] < Pivot , increment i
If A[j] > Pivot , decrement j



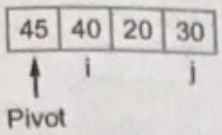


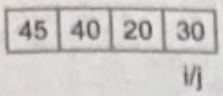




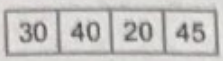
(left sublist)

Pass 2a



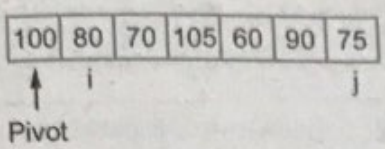


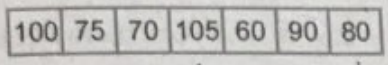
∴ Swap(A[j], Pivot)

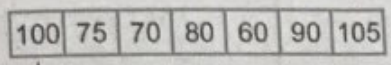


(Right sublist)

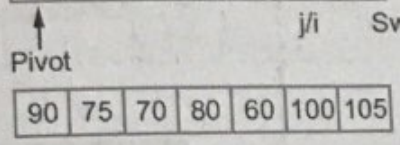
Pass 2b



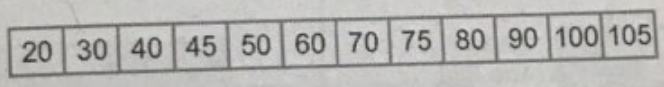




Swap(A[j], Pivot)



Thus continuing in this fashion, by sorting each sub-sublists finally we get the sorted list as

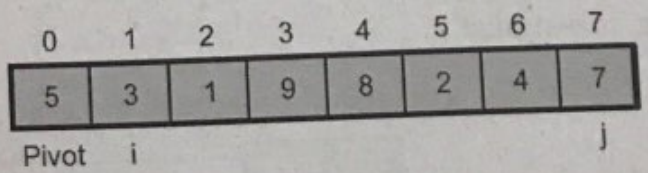


Example 3.8.3 *Write the Quick sort algorithm. Track the same on data set : 5, 3, 1, 9, 8, 2, 4, 7*

GTU : IT, Winter-14, Marks 7

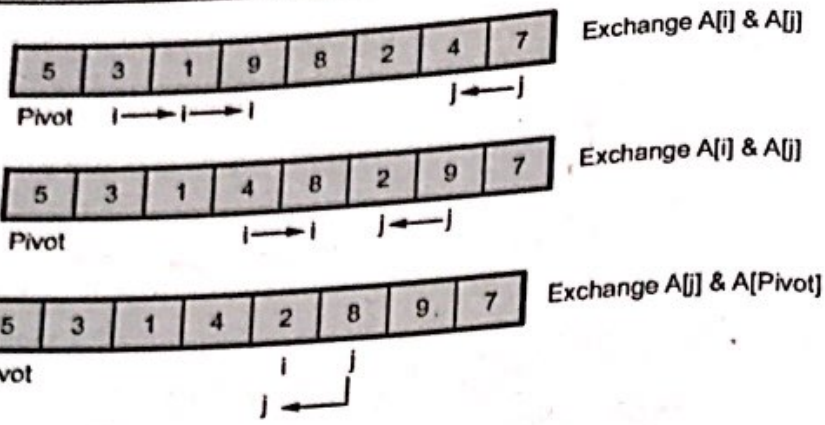
Solution : Arrange the elements in an array.

Step 1 :

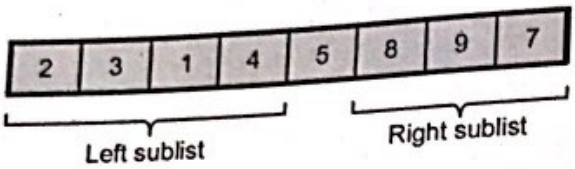


If A [i] < A [pivot], increment i.

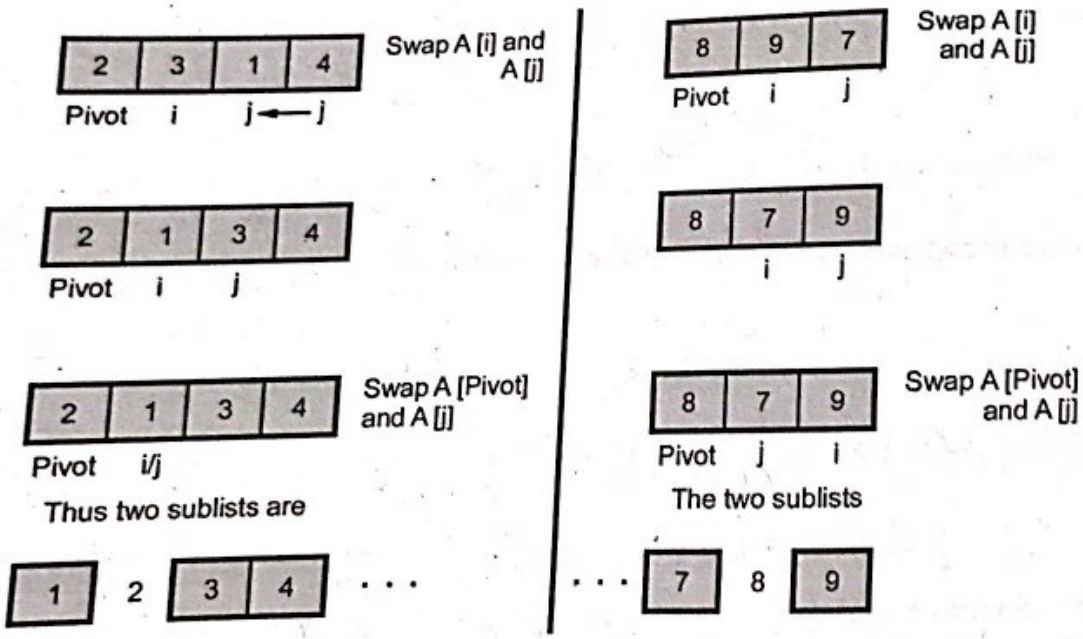
If A [j] > A[pivot], decrement j.



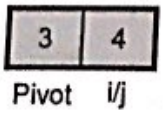
After pass 1



Step 2 : Now sorting two sublists using quick sort technique.



Step 3 : Now sorting sub-sublist



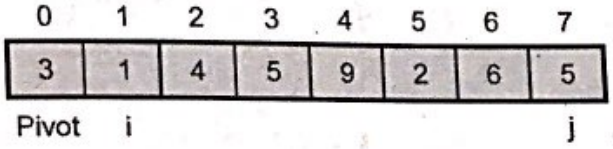
Example 3.8.4 *Write an algorithm for quick sort and derive best case, worst case using divide and conquer technique also trace given data (3, 1, 4, 5, 9, 2, 6, 5)*

GTU : Winter-15, Marks 7

Solution : Algorithm : Refer section 3.8.

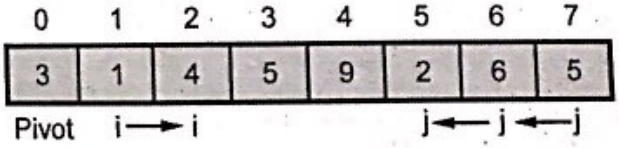
Derivation for best, worst case : Refer section 3.8.

Step 1 : Consider the elements in an array

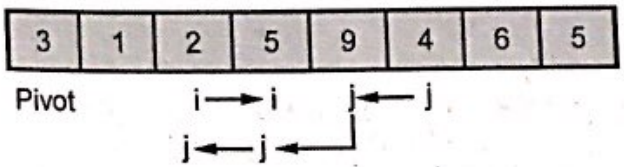


If A[Pivot] > A[i] increment i

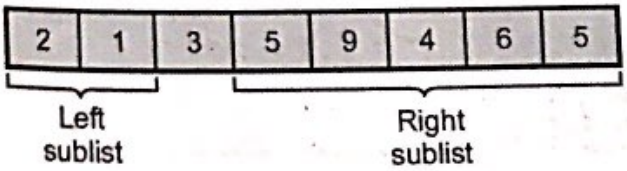
If A[Pivot] < A[j] increment j



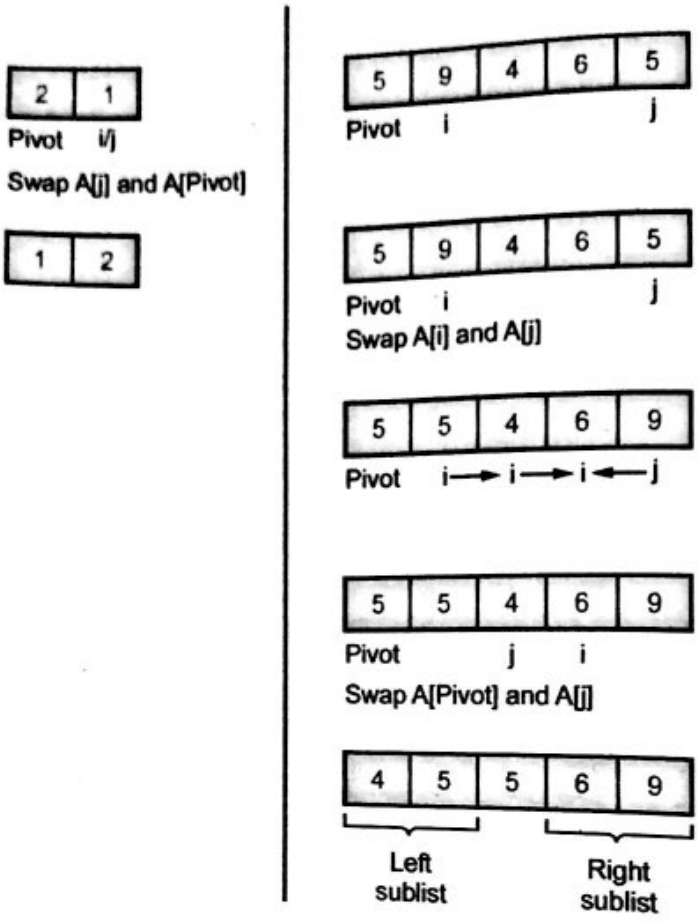
Now swap A [i] and A [j]



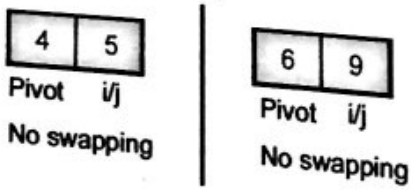
Swap A [j] and A [Pivot]



Step 2 : Now sorting two sublists using quick sort technique.

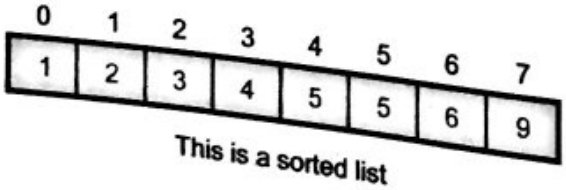


Step 3 : We will sort two sublists obtained in step 2.



Step 4 : We will combine all the sublists obtained in step 2 and step 3.

The resultant list will be



Algorithm

The quick sort algorithm is performed using following two important functions - *Quick* and *partition*. Let us see them-

```
Algorithm Quick(A[0...n–1], low,high)
//Problem Description : This algorithm performs sorting of
//the elements given in Array A[0...n–1]
//Input: An array A[0...n–1] in which unsorted elements are
//given. The low indicates the leftmost element in the list
//and high indicates the rightmost element in the list
//Output: Creates a sub array which is sorted in ascending
//order
if(low<high)then
//split the array into two sub arrays
m ← partition(A[low...high])// m is mid of the array
Quick(A[low...m–1])
Quick(A[mid+1...high])
```

In above algorithm call to partition algorithm is given. The *partition* performs arrangement of the elements in ascending order. The recursive *quick* routine is for dividing the list in two sub lists. The pseudo code for *Partition* is as given below -

```
Algorithm Partition (A[low...high])
//Problem Description: This algorithm partitions the
//subarray using the first element as pivot element
//Input: A subarray A with low as left most index of the
//array and high as the rightmost index of the array.
//Output: The partitioning of array A is done and pivot
//occupies its proper position. And the rightmost index of
//the list is returned
pivot ← A[low]
i ← low
j ← high+1
while(i<=j) do
{
 while(A[i]<=pivot) do
     i ← i+1
 while(A[j]>=pivot) do
     j ← j–1;
 if(i<=j) then
          swap(A[i],A[j])//swaps A[i] and A[j]
}
swap(A[low],A[j])//when i crosses j swap A[low] and A[j]
return j//rightmost index of the list
```

The Partition function is called to arrange the elements such that all the elements that are less than pivot are at the left side of pivot and all the elements that are greater than

pivot are all at the right of pivot. In other words pivot is occupying its proper position and the partitioned list is obtained in an ordered manner.

Analysis

Best case (split in the middle)

If the array is always partitioned at the mid, then it brings the best case efficiency of an algorithm.

The recurrence relation for quick sort for obtaining best case time complexity is,

$$C(n) = \underbrace{C(n/2)}_{\text{Time required to sort left sub array}} + \underbrace{C(n/2)}_{\text{Time required to sort right sub array}} + \underbrace{n}_{\text{Time required for partitioning the sub array}} \quad \text{... equation (3.8.1)}$$

and $C(1) = 0$

Method 1 : Using Master theorem

We will solve equation (3.7.1) using master theorem.

The master theorem is

If $f(n) \in \Theta(n^d)$ then

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

We get,

$$C(n) = 2\,C(n/2) + n$$

Here $f(n) \in n^1$ $\therefore d = 1$

Now, $a = 2$ and $b = 2$.

As from case 2 we get $a = b^d$ i.e. $2 = 2^1$, we get

$T(n)$ i.e. $C(n) = \Theta(n^d \log n)$

$\therefore$ $C(n) = \Theta(n \log n)$

Thus,

Best case time complexity of quick sort is $\Theta(n \log_2 n)$.

The best case of quick sort can be represented by Fig. 3.8.2.

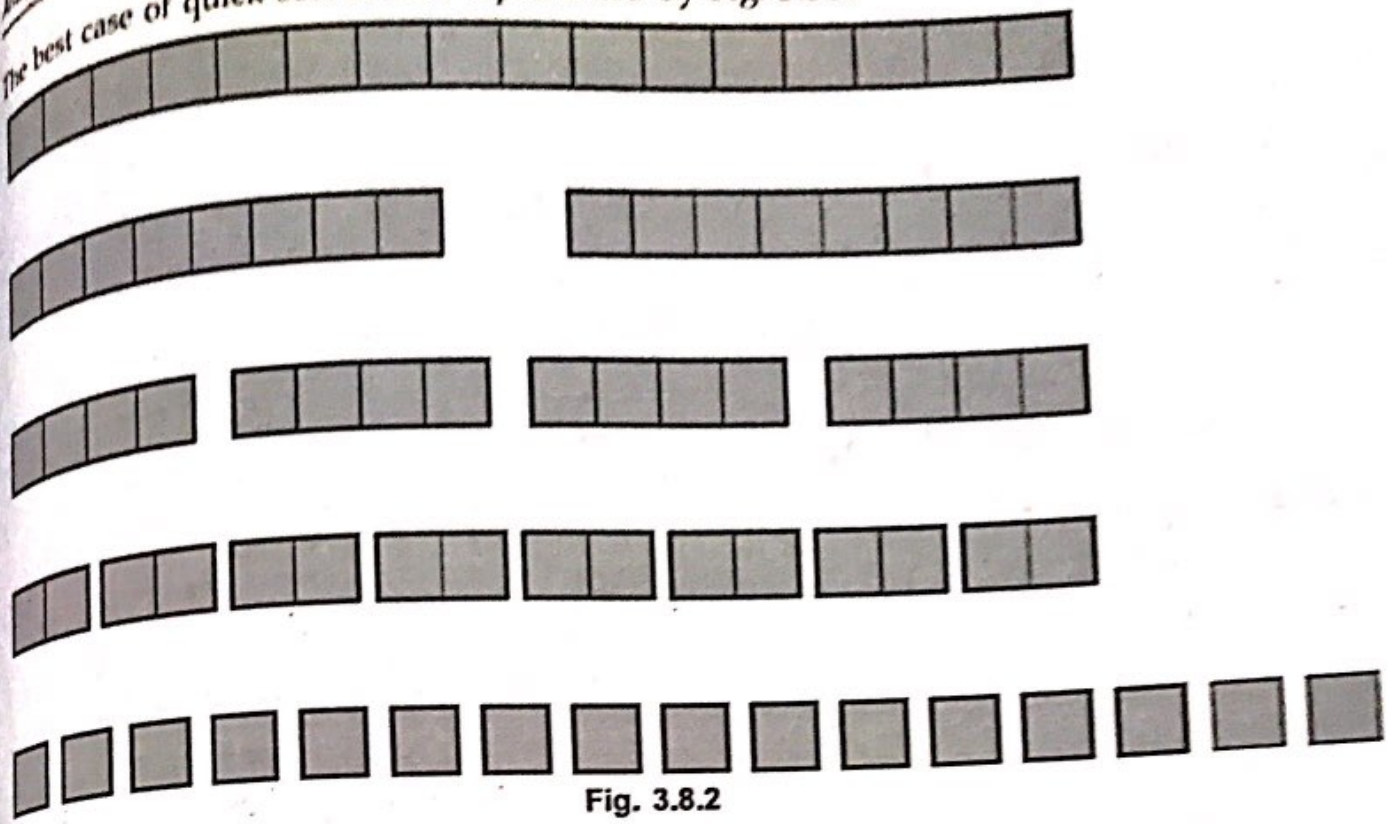


Fig. 3.8.2

Method 2 :

We can obtain the best case time complexity of quick sort using substitution method as well. Consider equation (1) once again

$$C(n) = C(n/2) + C(n/2) + n$$

$$C(n) = 2C(n/2) + n$$

We assume $n = 2^k$ since each time the list is divided into two equal halves. Then equation becomes,

$$C(2^k) = 2C(2^k / 2) + 2^k$$

$$= 2C(2^{k-1}) + 2^k$$

Now substitute $C(2^{k-1}) = 2C(2^{k-2}) + 2^{k-1}$

We get $C(2^k) = 2[2C(2^{k-2}) + 2^{k-1}] + 2^k$

$$C(2^k) = 2^2 C(2^{k-2}) + 2 \cdot 2^{k-1} + 2^k$$

$$= 2^2 C(2^{k-2}) + 2^k + 2^k$$

$$C(2^k) = 2^2 C(2^{k-2}) + 2 \cdot 2^k$$

If we substitute $C(2^{k-2})$ then,

$$C(2^k) = 2^2\, C(2^{k-2}) + 2 \cdot 2^k$$

$$= 2^2 [2C(2^{k-3}) + 2^{k-2}] + 2 \cdot 2^k$$

$$= 2^3 C(2^{k-3}) + 2^2 \cdot 2^{k-2} + 2 \cdot 2^k$$

$$= 2^3 C(2^{k-3}) + 2^k + 2 \cdot 2^k$$

$$C(2^k) = 2^3 C(2^{k-3}) + 3 \cdot 2^k$$

Similarly we can write

$$C(2^k) = 2^4 C(2^{k-4}) + 4 \cdot 2^k$$

...

...

...

$$= 2^k C(2^{k-k}) + k \cdot 2^k$$

$$= 2^k C(2^0) + k \cdot 2^k$$

$$C(2^k) = 2^k C(1) + k \cdot 2^k$$

But $C(1) = 0$ Hence the above equation becomes

$$C(2^k) = 2^k \cdot 0 + k \cdot 2^k$$

Now as we assumed $n = 2^k$ we can also say

$$k = \log_2 n \quad \text{[By taking logarithm on both sides]}$$

$$C(n) = n \cdot 0 + \log_2 n \cdot n$$

$$C(n) = n \cdot 0 + \log_2 \cdot n$$

Thus it is proved that best case time complexity of quick sort is $\Theta(n \log_2 n)$.

Worst case (sorted array)

The worst case for quick sort occurs when the pivot is a minimum or maximum of all the elements in the list. This can be graphically represented as

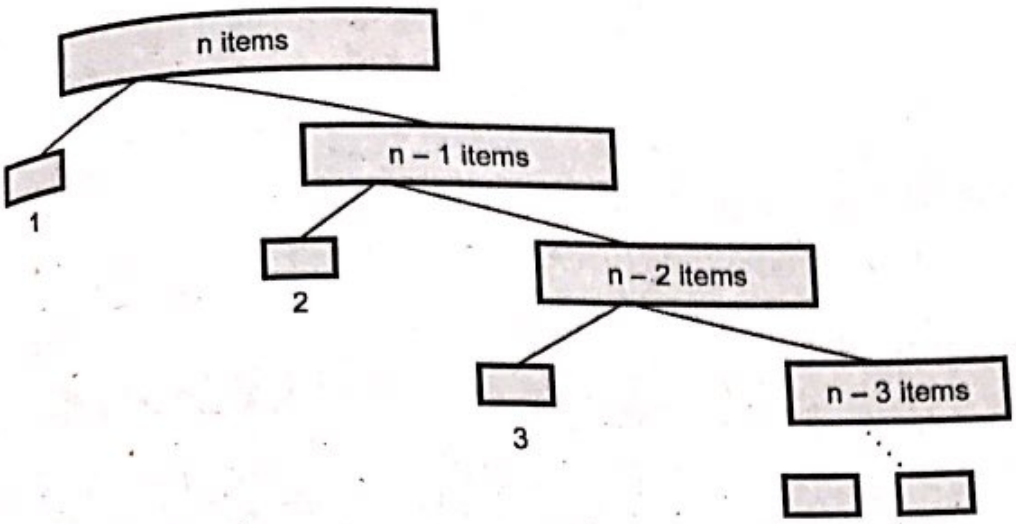


Fig. 3.8.3

We can write it as

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

or

$$C(n) = n + (n-1) + (n-2) + \ldots + 2 + 1$$

But as we know

$$1+2+3+\ldots+n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2.$$

$$\therefore \quad C(n) = \Theta(n^2)$$

The time complexity of worst case of quick sort is $\Theta(n^2)$.

Average case (random array)

Let, $C_{avg}(n)$ denotes the average time of quick sort (A[1 ... n]).

The recurrence relation for random input array is

$$C(n) = C(0) + C(n-1) + n$$

or

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

or

$$C(n) = C(2) + C(n-3) + n$$

or

$$C(n) = C(3) + C(n-4) + n$$

...

...

...

$$C(n) = C(n-1) + C(0) + n$$

The average value of C(n) is sum of all the possible values divided by n.

$$\therefore \quad C_{avg}(n) = \frac{2}{n}\,(C(1) + C(2) + \ldots + C(n-1)) + n$$

Multiplying both the sides by n we get,

$$n * C_{avg}(n) = 2\,(C(1) + C(2) + \ldots + C(n-1)) + n^2$$

Now we put n = n – 1 then,

$$(n-1) * C_{avg}(n-1) = 2\,[C_{avg}(1) + C_{avg}(2) + \ldots + C_{avg}(n-2)] + (n-1) * (n-1)$$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2[C_{avg}(1) + C_{avg}(2) + \ldots + C_{avg}(n-2)] + (n-1) * (n-1)$$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2C_{avg}\,(n-1) + (2n-1)$$

$$n * C_{avg}(n) = (n+1) * C_{avg}\,(n-1) + (2n-1) < (n+1) * C_{avg}\,(n-1) + 2n$$

If we assume

$$(n+1) * C_{avg}\,(n-1) + 2n = (n+1) * C_{avg}\,(n-1) + C'n$$

Then,

$$(n+1) * C_{avg}\,(n-1) < (n+1) * C_{avg}\,(n-1) + C'n$$

Divide this equation by n (n + 1) then

$$n * C_{avg}(n) < (n+1) * C_{avg}(n-1) + C'n$$

$$\frac{C_{avg}(n)}{(n+1)} < \frac{C_{avg}\,(n-1)}{n} + \frac{C'}{(n+1)}$$

If we assume

$$A(n) = \frac{C_{avg}(n)}{(n+1)} \text{ then}$$

$$A(n) < A\,(n-1) + \frac{C'}{(n+1)}$$

$$A(n-1) < A\,(n-2) + \frac{C'}{n}$$

$$A(n-2) < A\,(n-3) + \frac{C'}{n-1}$$

...

...

...

$$A(1) < A\,(0) + \frac{C'}{2}$$

Adding and canceling these equatio we get,

$$A(n) < C' * \left[\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} + \frac{1}{n+1}\right]$$

$$\therefore C_{avg}(n) = (n+1) * A(n)$$

i.e. $(n+1) * A(n) < C''(n+1) \log n$

$$\therefore C_{avg}(n) = \Theta\,(n \log n)$$

Hence average case time complexity of quick sort is $\Theta(n \log n)$.

Time complexity of quick sort

Best case	Average case	Worst case
$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n^2)$

C Function

```
int partition(int A[SIZE],int low,int high)
{
        int pivot=A[low],i=low,j=high;
        while(i<=j)
        {
         while(A[i]<=pivot)
              i++;
         while(A[j]>pivot)
              j--;
         if(i<j)
              swap(A,&i,&j);
        }
        swap(A,&low,&j);
        return j;
}
```

C Program

```
/*****************************************************************************
Program to sort the elements in ascending order using Quick Sort.
*****************************************************************************/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 10
void Quick(int A[SIZE],int,int);
int partition(int A[SIZE],int,int);
void swap(int A[SIZE],int *,int *);
int n;
int main()
{
        int i;
        int A[SIZE];
        clrscr();
        printf("\n\t\t Quick Sort Method \n");
        printf("\n Enter Total numbers to sort : ");
```

4 Dynamic Programming

Syllabus

Introduction, The principle of optimality, Problem solving using dynamic programming - Calculating the binomial coefficient, Making change problem, Assembly line-scheduling, Knapsack problem, All points shortest path, Matrix chain multiplication, Longest common subsequence.

Contents

4.1 Introduction

Dynamic programming is typically applied to optimization problem. This technique is invented by a U.S. Mathematician Richard Bellman in 1950. In the word dynamic programming the word **programming** stands for **planning** and it does not mean by computer programming.

- Dynamic programming is technique for solving problems with overlapping subproblems.
- In this method each subproblem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to the original problem.

4.1.1 General Method

Dynamic programming is typically applied to optimization problems.

For each given problem, we may get any number of solutions we seek for optimum solution (i.e. minimum value or maximum value solution). And such an optimal solution becomes the solution to the given problem.

4.2 Comparison of Dynamic Programming with Other Strategies

GTU : Dec.-10,11, May-12, Marks 4

4.2.1 Divide and Conquer and Dynamic Programming

Sr. No.	Divide and conquer	Dynamic programming
1.	The problem is divided into small subproblems. These subproblems are solved independently. Finally all the solutions of subproblems are collected together to get the solution to the given problem.	In dynamic programming many decision sequences are generated and all the overlapping subinstances are considered.
2.	In this method duplications in subsolutions are neglected. i.e., duplicate subsolutions may be obtained.	In dynamic computing duplications in solutions is avoided totally.
3.	Divide and conquer is less efficient because of rework on solutions.	Dynamic programming is efficient than divide and conquer strategy.
4.	The divide and conquer uses top down approach of problem solving (recursive methods).	Dynamic programming uses bottom up approach of problem solving (iterative method).
5.	Divide and conquer splits its input at specific deterministic points usually in the middle.	Dynamic programming splits its input at every possible split points rather than at a particular point. After trying all split points it determines which split point is optimal.

4.2.2 Steps of Dynamic Programming

Dynamic programming design involves 4 major steps :

1. Characterize the structure of optimal solution. That means develop a mathematical notation that can express any solution and subsolution for the given problem.
2. Recursively define the value of an optimal solution.
3. By using bottom up technique compute value of optimal solution. For that you have to develop a recurrence relation that relates a solution to its subsolutions, using the mathematical notation of step 1.
4. Compute an optimal solution from computed information.

4.2.3 Principle of Optimality

The dynamic programming algorithm obtains the solution using principle of optimality.

The principle of optimality states that "in an optimal sequence of decisions or choices, each subsequence must also be optimal."

When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the dynamic programming approach.

For example : Finding of shortest path in a given graph uses the principle of optimality.

4.2.4 Problem Solving using Dynamic Programming

Various problems that can be solved using dynamic programming are -

1. Calculating the Binomial coefficient
2. Making change problem
3. Assembly line scheduling
4. Knapsack problem
5. Shortest path
6. Matrix chain Multiplication

Review Questions

1. *Define : Principle of optimality.* **GTU : Dec.-10, Marks 2**
2. *Explain the difference between divide and conquer and dynamic programming.* **GTU : Dec.-11, Marks 3**

3. *What is principle of optimality? Explain its use in dynamic programming method.* **GTU : May-12, Marks 4**

4.3 Calculating the Binomial Coefficient

Computing binomial coefficient is a typical example of applying dynamic programming.

In mathematics, particularly in **combinatorics**, binomial coefficient is a coefficient of any of the terms in the expansion of $(a+b)^n$. It is denoted by C(n, k) or $\binom{n}{k}$ where $(0 \leq k \leq n)$.

The **formula** for computing binomial coefficient is,

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

and $$C(n, 0) = 1$$

$$C(n, n) = 1$$

where $n > k > 0$

Example 4.3.1 *Compute C (4, 2) using Dynamic programming.*

Solution : $n = 4, k = 2$

$$C(4, 2) = C(n-1, k-1) + C(n-1, k)$$

$$C(4, 2) = C(3, 1) + C(3, 2) \quad \ldots (4.3.1)$$

As there are two unknowns : C(3, 1) and C(3, 2) in above equation we will compute these sub instances of C(4, 2).

$\therefore$ $n = 3, k = 1$

$$C(3, 1) = C(2, 0) + C(2, 1)$$

As $C(n, 0) = 1$ We can write

$$C(2, 0) = 1$$

$\therefore$ $$C(3, 1) = 1 + C(2, 1)$$

Hence let us compute C(2, 1). $\ldots (4.3.2)$

$n = 2, k = 1$

$\therefore$ $$C(2, 1) = C(n-1, k-1) + C(n-1, k)$$

$$= C(1, 0) + C(1, 1)$$

But as $C(n, 0) = 1$ and $C(n, n) = 1$ we get

$C(1, 0) = 1$ and $C(1, 1) = 1$

$C(2, 1) = C(1, 0) + C(1, 1)$

$\therefore \quad = 1 + 1$

$$\boxed{C(2, 1) = 2} \qquad \text{... (4.3.3)}$$

Put this value in equation (4.3.2) and we get

$C(3, 1) = 1 + 2$

$$\boxed{C(3, 1) = 3} \qquad \text{... (4.3.4)}$$

Now to solve equation (4.3.1) we will first compute C(3, 2) with n = 3 and k = 2.

$C(3, 2) = C(n - 1, k - 1) + C(n - 1, k)$

$\therefore$

$C(3, 2) = C(2, 1) + C(2, 2)$

But as $C(n, n) = C(2, 2) = 1$, we will put values of C(2, 1) [obtained in equation (4.3.3) and C(2, 2) in C(3, 2) we get,

$C(3, 2) = C(2, 1) + C(2, 2)$

$= 2 + 1$

$$\boxed{C(3, 2) = 3} \qquad \text{... (4.3.5)}$$

Put equation (4.3.4) and (4.3.5) in equation (4.3.1), then we get

$C(4, 2) = C(3, 1) + C(3, 2)$

$= 3 + 3$

$\boxed{C(4, 2) = 6}$ is the final answer.

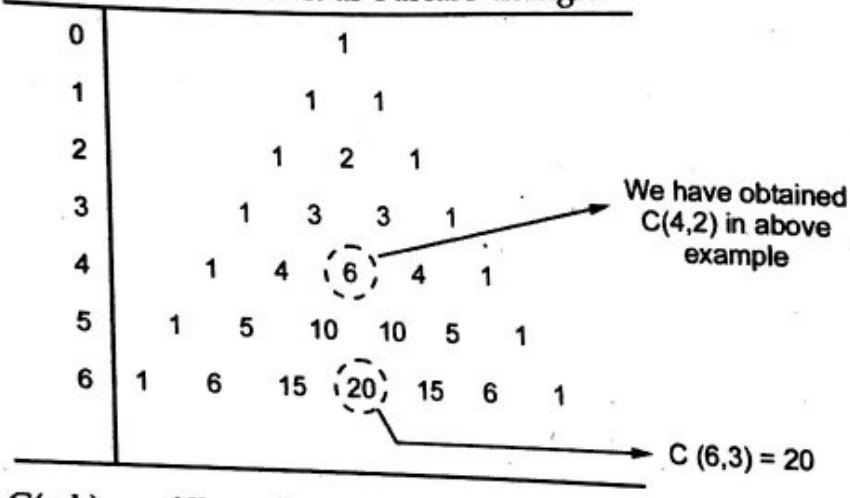
How Dynamic Programming approach is used ?

While computing C(n,k) the smaller overlapping sequences get generated by C(n - 1, k - 1) and C(n - 1, k). These overlapping, smaller instances of problem need to be solved first. The solutions which we obtained by solving these instances will ultimately generate the final solution. Thus for computing binomial coefficient dynamic programming is used.

If we record binomial coefficients n and k values ranging from 0 to n and 0 to k then

	0	1	2	3	4	5	...	(k – 1)	k
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5									
⋮									
k	1								1
⋮									
(n – 1)	1							C(n – 1, k – 1)	C(n – 1, k)
n	1								C(n,k)

This basically is a structure known as Pascal's triangle.



To compute C(n,k) we fill up the above given table row by row starting with C(n,0) = 1 and ending with C(n, n) = 1. The cell at current row is calculated by two adjacent cells of previous row. The algorithm for computing binomial coefficient is as given below,

Algorithm

```
Algorithm Binomial (n,k)
//Problem Description : This algorithm is for
//computing C(n,k) i.e., Binomial coefficient
//Input : A pair of non negative integers n and k.
//Output : The value of C(n,k)
        for i ← 0 to n do
        {
```

```
        for j←0 to k do
//k is computed as min(i,k)
            if ((j=0 or (i=j)) then
            C[i,j] ← 1
            else //start filling the table
                C[i,j] ← C[i–1,j–1] + C[i–1,j]

    }
    return C[n,k] //Computed value of C(n,k)
```

Analysis

The basic operation is addition i.e.,

$$C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$$

Let A(n,k) denotes total additions made in computing C(n,k).

$$A(n,k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1$$

$$= \sum_{i=1}^{k} [(i-1)-1+1] + \sum_{i=k+1}^{n} (k-1+1)$$

$$\because \sum_{i=l}^{n} 1 = (n-l+1)$$

$$= \underbrace{\sum_{i=1}^{k} (i-1)}_{\downarrow} + \sum_{i=k+1}^{n} k$$

$$= [1+2+3+\ldots(k-1)] + k \sum_{i=k+1}^{n} 1$$

$$= \frac{(k-1)\,k}{2} + k \sum_{i=k+1}^{n} 1$$

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

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

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

$$= \frac{k^2}{2} - \frac{k}{2} + nk - k^2$$

$$\approx nk$$

Hence time complexity of binomial coefficient is Θ (nk).

}

Output

Enter the value of n and k = 4,2

The Binomial Coefficient? C(4,2) = 6

4.4 Making Change Problem

GTU : June-11, May-12, Summer-13, Winter-15, Marks 7

Suppose there are some coins available. The values of these coins are as follows.

1 dollar = 100 cents

1 quarter = 25 cents

1 dimes = 10 cents

1 nickels = 5 cents

1 pennies = 1 cent

Now the making change problem is to pay the desired amount of money by using as few coins as possible.

For example : If a customer wants to pay 3.37 $ then he should pay 3 dollars (= 300 cents) + 1 quarter (= 25 cents) + 1 dime (= 10 cents) + 2 pennies (= 2 cents) = 3.37 $.

This method uses **Greedy** approach because at every step it chooses the largest coin it can. Furthermore, once the coin is selected then that is the final. This approach does not allow to change the decision. Unfortunately although **Greedy** approach is very efficient, it works only for limited number of instances. For example if there exists coins of 1, 4 and 6 units and we have to make change for 9 units. Then there are two ways :

1. **Greedy Method :** Select one coin of 6 unit and 3 coins of 1 unit = **4 coins.**
2. **Better Method :** Select two coins of 4 units and 1 coin of 1 unit = **3 coins.**

This better method is devised by dynamic programming approach.

For solving this problem using dynamic programming approach we need to build a table, in which rows correspond to denomination and column corresponds to 0 to N units.

		0	1	2	3	4	5	6	7	8
i = 1	$d_1 = 1$	0								
i = 2	$d_2 = 4$	0								
i = 3	$d_3 = 6$	0								

We can perform computations row-by-row from left to right, or column-by-column from top to bottom or diagonally.

Here we will perform computation **column-by-column** and fill up the table accordingly. We will use following formula for computations.

1. If $i = 1$ then $c[i, j] = 1 + c[1, j - d_1]$
2. If $j < d_i$ then $c[i, j] = c[i-1, j]$
3. Otherwise $c[i][j] = \min(c[i-1, j], 1 + c[i, j - d_i])$

c [1, 1] with i = 1, j = 1, $d_1 = 1$.

As i = 1

Formula used : $1 + c[1, j - d_1]$

$= 1 + c[1, 1-1]$

$= 1 + c[1, 0]$

$= 1 + 0$

$= 1$

$\therefore$ **c[1, 1] = 1**

c [2, 1] with i = 2, j = 1, $d_2 = 4$.

As $j < d_2$ i.e. $1 < 4$

Formula used : $c[i-1, j]$

$= c[1, 1]$

$= 1$

$\therefore$ **c[2, 1] = 1**

c [3, 1] with i = 3, j = 1, $d_3 = 6$

As $j < d_3$ i.e. $1 < 6$

Formula used : $c[i-1,j]$

$= c[3-1,1]$

$= c[2,1]$

$= 1$

$\therefore$ $c[3,1] = 1$

$c[1,2]$ with $i=1,\ j=2,\ d_1 = 1$

As $i = 1$

Formula used : $1 + c[1,j-d_1]$

$= 1 + c[1,2-1]$

$= 1 + c[1,1]$

$= 1 + 1$

$= 2$

$\therefore$ $c[1,2] = 2$

$c[2,2]$ with $i=2,\ j=2,\ d_2 = 4$

As $j < d_2$ i.e. $2 < 4$

Formula used : $c[i-1,j]$

$= c[1,2]$

$= 2$

$\therefore$ $c[2,2] = 2$

$c[3,2]$ with $i=3,\ j=2,\ d_3 = 6$

As $j < d_3$ i.e. $2 < 6$

Formula used : $c[i-1,j]$

$= c[2,2]$

$= 2$

$\therefore$ $c[2,2] = 2$

$c[1,3]$ with $i=1,\ j=3,\ d_1 = 1$

As $i = 1$

Formula used : $1 + c[1,j-d_1]$

$= 1 + c[1, 3-1]$

$= 1 + c[1, 2]$

$= 1 + 2$

$= 3$

$\therefore \quad c[1, 3] = 3$

$c[2, 3]$ with $i=2$, $j=3$, $d_2 = 4$

As $j < d_3$ i.e. $3 < 4$

Formula used : $c[i-1, j]$

$= c[1, 3]$

$= 3$

$\therefore \quad c[2, 3] = 3$

$c[3, 3]$ with $i=3$, $j=3$, $d_3 = 6$

As $j < d_3$ i.e. $3 < 6$

Formula used : $c[i-1, j]$

$= c[2, 3]$

$= 3$

$\therefore \quad c[3, 3] = 3$

$c[1, 4]$ with $i=1$, $j=4$, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j-d_1]$

$= 1 + c[1, 4-1]$

$= 1 + c[1, 3]$

$= 1 + 3$

$= 4$

$\therefore \quad c[1, 4] = 4$

$c[2, 4]$ with $i=2$, $j=4$, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min(c[i-1,j], 1+c[i,j-d[i]])$

$= \min(c[2-1,4], 1+c[2,4-4])$

$= \min(c[1,4]\ 1+c[2,0])$

$= \min(4, 1+0)$

$= 1$

$\therefore$ $c[2,4] = 1$

$c[3,4]$ with $i=3,\ j=4,\ d_3 = 6$

As $j < d_3$ i.e. $4 < 6$

Formula used : $c[i-1,j]$

$= c[2,4]$

$= 1$

$\therefore$ $c[3,4] = 1$

$c[1,5]$ with $i=1,\ j=5,\ d_1 = 1$

As $i = 1$

Formula used : $1 + c[1,j-d_1]$

$= 1 + c[1,5-1]$

$= 1 + c[1,4]$

$= 1 + 4$

$= 5$

$\therefore$ $c[1,5] = 5$

$c[2,5]$ with $i=2,\ j=5,\ d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min(c[i-1,j], 1+c[i,j-d_2])$

$= \min(c[1,5], 1+c[2,5-4])$

$= \min(c[1,5], 1+c[2,1])$

$= \min(5, 1+1)$

$= 2$

$\therefore$ $c[2,5] = 2$

$c[3,5]$ with $i=3,\ j=5,\ d_3 = 6$

As $j < d_3$ i.e. $5 < 6$

Formula used : $c[i-1,j]$

$= c[3-1,5]$

$= c[2,5]$

$\therefore$ **c[3, 5] = 2**

c [1, 6] with i = 1, j = 6, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j-d_1]$

$= 1 + c[1, 6-1]$

$= 1 + c[1,5]$

$= 6$

$\therefore$ **c[1, 6] = 6**

c [2, 6] with i = 2, j = 6, $d_2 = 4$

As $i \neq 2$ and $j > d_2$

Formula used : $\min(c[i-1,j], 1+c[i, j-d_2])$

$= \min(c[1,6], 1+c[2,2])$

$= \min(6, 1+2)$

$= 3$

$\therefore$ **c[2, 6] = 3**

c [3, 6] with i = 3, j = 6, $d_3 = 6$

As $i \neq 3$ and $j > d_3$

Formula used : $\min(c[i-1,j], 1+c[i, j-d_3])$

$= \min(c[2,6], 1+c[3,0])$

$= \min(3, 1+0)$

$= 1$

$\therefore$ **c[3, 6] = 1**

c [1, 7] with i = 1, j = 7, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j-d_1]$

$= 1 + c[1,6]$

$= 1 + 6$

$c[1,7] = 7$

$\therefore$

$c[2,7]$ with $i=2,\ j=7,\ d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min(c[i-1,j], 1+c[i,j-d_2])$

$= \min(c[1,7], 1+c[2,3])$

$= \min(7, 1+3)$

$= 4$

$\therefore\ c[2,7] = 4$

$c[3,7]$ with $i=3,\ j=7,\ d_3 = 6$

As $i \neq 1$ and $j > d_3$

Formula used : $\min(c[i-1,j], 1+c[i,j-d_3])$

$= \min(c[2,7], 1+c[3,1])$

$= \min(4, 1+1)$

$= 2$

$\therefore\ c[3,7] = 2$

$c[1,8]$ with $i=1,\ j=8,\ d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j-d_1]$

$= 1 + c[1,7]$

$= 1 + 7$

$= 8$

$\therefore\ c[1,8] = 8$

$c[2,8]$ with $i=2,\ j=8,\ d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min(c[i-1,j], 1+c[i,j-d_2])$

$= \min(c[1,8], 1+c[2,4])$

$= \min(8, 1+1)$

$= 2$

$\therefore \quad c[2,8] = 2$

$c[3,8]$ with $i=3,\ j=8,\ d_3 = 6$

As $i \neq 1$ and $j > d_3$

Formula used : $\min(c[i-1,j], 1+c[i,j-d_3])$

$= \min(c[2,8], 1+c[3,2])$

$= \min(2, 1+2)$

$= 2$

$\therefore \quad c[3,8] = 2$

Thus we can represent the table filled up by above computations will be

		0	1	2	3	4	5	6	7	8
i = 1	$d_1 = 1$	0	1	2	3	4	5	6	7	8
i = 2	$d_2 = 4$	0	1	2	3	1	2	3	4	2
i = 3	$d_3 = 6$	0	1	2	3	1	2	1	2	2

↑ Total number of coins

The value c [n, N] i.e. c [3, 8] = 2 represents the minimum number of coins required to get the sum for 8 units. Hence we require 2 coins for getting 8 units. The coins are :

4 units = 1coin
+ 4 units = 1coin
8 units = 2coins

Example 4.4.2 *Given coins of denominations 2, 4 and 5 with amount to be pay is 7. Find optimal number of coins and sequence of coins used to pay given amount using dynamic method.*

Initially C [i, 0] = 0

C[1, 0] = C [2, 0] = C [3, 0] = 0

∴

		0	1	2	3	4	5	6	7	8
i = 1	$d_1 = 2$	0								
i = 2	$d_2 = 4$	0								
i = 3	$d_3 = 5$	0								

Computations can be performed column by column; and fill up the table accordingly.

C [1, 1] with i = 1, j = 1, $d_1 = 2$

As i = 1

C [1,1] = 1+C[1,j-d1]

C [1,1] = 1+C[1,-1]

C [1, 1] = ∞

C [2, 1] with i = 2, j = 1, $d_2 = 4$

C [2, 1] = C [i – 1, j]

C [2, 1] = C [1, 1]

C [2, 1] = ∞

C [3, 1] with i = 3, j = 1, $d_3 = 5$

As $j < d_3$

C [i, j] = C [i – 1, j]

C [3, 1] = C [2, 1]

∴ **C [3, 1] = ∞**

C [1, 2] with i = 1, j = 2, $d_1 = 2$

As i = 1

C [i, j] = 1+ C [1, j – d_1]

C [1, 2] = 1 + C [1, 0] = 1 + 0

∴ **C [1, 2] = 1**

C [2, 2] with i = 2, j = 2, $d_2 = 4$

As $j < d_2$

C [i, j] = C [i – 1, j]

C [2, 2] = C [1, 2]

C [2, 2] = 1

C[3, 2] with $i = 3, j = 2, d_3 = 5$

As $j < d_3$

$$C[i, j] = C[i-1, j]$$

$$\therefore \quad C[3, 2] = C[2, 2]$$

$$\therefore \quad C[3, 2] = 1$$

$C[1, 3]$ with $i = 1, j = 3\ d_1 = 2$

As $i = 1$

$$C[i, j] = 1 + C[1, j - d_1]$$

$$= 1 + C[1, 3-2] = 1 + C[1, 1]$$

$$C[1, 3] = \infty$$

$C[2, 3]$ with $i = 2, j = 3, d_2 = 4$

As $j < d_2$

$$C[i, j] = C[i-1, j]$$

$$C[2, 3] = C[1, 3]$$

$$\therefore \quad C[2, 3] = \infty$$

$C[3, 3]$ with $i = 3, j = 3, d_3 = 5$

As $j < d_3$

$$C[i, j] = C[i-1, j]$$

$$\therefore \quad C[3, 3] = C[2, 3]$$

$$\therefore \quad C[3, 3] = \infty$$

$C[1, 4]$ with $i = 1, j = 4, d_1 = 2$

As $i = 1$

$$C[i, j] = 1 + C[1, j - d_1]$$

$$C[1, 4] = 1 + C[1, 2] = 1 + 1$$

$$\therefore \quad C[1, 4] = 2$$

$C[2, 4]$ with $i = 2, j = 4, d_2 = 4$

As $d_2 = j$ and $i \neq 1$

$$C[i, j] = \min(C[i-1, j], 1 + C[i, j - d[i]])$$

$$= \min(C[1,4], 1 + C[2, 4-4])$$

$= \min (C[1,4], 1+0)$

$C[2, 4] = \min (2, 1)$

$C[3, 4]$ with $i = 3, j = 4$ and $d_3 = 5$

As $j < d_3$

$C[i, j] = C[i - 1, j]$

$C[3, 4] = C[2, 4]$

$\therefore C[3, 4] = 1$

$C[1, 5]$ with $i = 1, j = 5$ and $d_1 = 2$

As $i = 1$

$C[i, j] = 1 + C[i, j - d_1]$

$= 1 + C[1, 3]$

$C[1, 5] = \infty$

$C[2, 5]$ with $i = 2, j = 5$ and $d_2 = 4$

$C[i, j] = \min (C[i-1, j], 1+C[i, j-d[i]])$

$= \min (C[1,5], 1+C[2,1])$

$C[2, 5] = \min (\infty, 1+\infty)$

$C[2, 5] = \infty$

$C[3, 5]$ with $i = 3, j = 5$ and $d_3 = 5$

$C[i, j] = \min (C[i-1, j], 1+C[i, j-d[i]])$

$= \min (C[2,5], 1+C[3,0])$

$C[3, 5] = \min (\infty, 1+0)$

$C[3, 5] = 1$

$C[1, 6]$ with $i = 1, j = 6, d_1 = 2$

As $i = 1$

$C[i, j] = 1 + C[i, j - d_1]$

$C[1, 6] = 1 + C[1, 4] = 1 + 2$

$C[1, 6] = 3$

$C[2, 6]$ with $i = 2, j = 6, d_2 = 4$

$C[i, j] = \min (C[i-1, j], 1+C[i, j-d[i]])$

$= \min (C[1,6],\ 1+C[2,2])$

$C[2, 6] = \min (3,\ 1+1)$

$\mathbf{C[2, 6] = 2}$

$C[3, 6]$ with $i = 3, j = 6, d_3 = 5$

$C[i, j] = \min (C[i-1, j],\ 1+C[i, j-d[i]])$

$= \min (C[2,6],\ 1+C[3,1])$

$= \min (2,\ 1+\infty)$

$\mathbf{C[3, 6] = 2}$

$C[1, 7]$ with $i = 1, j = 7$ and $d_1 = 2$

As $i = 1$

$C[i, j] = 1+ C[i, j - d_1]$

$= 1 + C[1, 5] = 1+\infty$

$\mathbf{C[1, 7] = \infty}$

$C[2,7]$ with $i = 2, j = 7$ and $d_2 = 4$

$C[i, j] = \min (C[i-1, j],\ 1+C[i, j-d[i]])$

$= \min (C[1,7],\ 1+C[2,3])$

$= \min (\infty, 1+\infty)$

$\mathbf{C[2, 7] = \infty}$

$C[3, 7]$ with $i = 3, j = 7$ and $d_3 = 5$

$C[i, j] = \min (C[i-1, j],\ 1+C[i, j-d[i]])$

$= \min (C[2,7],\ 1+C[3,2])$

$= \min (\infty,\ 1+1)$

$\mathbf{C[3, 7] = 2}$

The table can be

That means 2 coins are required for sum as 7. The coins are,

$d_1 = 2 \rightarrow 1$ coin

$d_3 = 5 \rightarrow 1$ coin

Total 7 with 2 coins

Example 4.4.3 *Solve making change problem using Dynamic Programming. (Denominations : d1 = 1, d2 = 4, d3 = 6). Give your answer for making change of Rs. 9.*

GTU : Winter-15, Marks 7

Solution : $d_1 = 1$, $d_2 = 4$, d3 = 6, N = 9 ₹, n = 3. Hence we will create a table with rows ranging from 1 to 3 and columns ranging from 0 to 9.

Initially C[i, 0] = 0 ∴ C[1, 0] = C [2, 0] = C [3, 0] = 0

j →

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0									
$d_2 = 4$	0									
$d_3 = 6$	0									

We will complete the table **column-by-column :** Refer example 4.4.1.

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0	1	2	3	4	5	6	7	8	
$d_2 = 4$	0	1	2	3	1	2	3	4	2	
$d_3 = 6$	0	1	2	3	1	2	1	2	2	

d[1, 9] with i = 1, j = 9, $d_1 = 1$

As i = 1

Formula used : $1 + c[1, j - d_1]$

$= 1 + c[1, 8]$

$= 1 + 8$

$= 9$

∴ **c[1, 9] = 9**

c[2, 9] with i = 2, j = 9, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min (c[i-1, j], 1 + c[i, j - d_2])$

$= \min (c[1, 9], 1 + c[2, 5])$

$= \min (9, 1 + 2)$

$= 3$

$\therefore \quad c[2, 9] = 3$

$c[3, 9]$ with $i = 3$, $j = 9$, $d_3 = 6$

As $i \neq 1$ and $j > d_3$

Formula used : $\min (c[i-1, j], 1 + c[i, j - d_3])$

$= \min (c[2, 9], 1 + c[3, 3])$

$= \min (3, 1 + 3)$

$= 3$

$\therefore \quad c[3, 9] = 3$

Thus the complete table filled up by all the computations will be

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0	1	2	3	4	5	6	7	8	9
$d_2 = 4$	0	1	2	3	1	2	3	4	2	3
$d_3 = 6$	0	1	2	3	1	2	1	2	2	3

Total number of coins

The value c[n, N] i.e. c [3, 9] = 3 represents the minimum number of coins required to get the sum for 9 units. Hence coins will be

$d_2 = 4$	1 coin
$d_2 = +4$	1 coin
$d_1 = +1$	1 coin
₹ 9	= 3 coins

Algorithm Number_of_Coins (N)

{

// **Problem Description** : This algorithm computes

```
//minimum number of coins required to make
// change for N units.
    for ( i ← 1 to n) do
    initialize array d[i] with the values of coins.
for ( i ← 1 to n ) do
        c [i, 0] ← 0
For ( i ← 1 to n ) do
{
    For ( i ← 1 to N ) do
    {

        if (i = 1 & & j < d [ i] ) then
            c [ i ] [ j ] = infinity ;
        else if ( i = 1 ) then
            c [ i ] [ j ] = 1+c[1, j-d[1]];
        else if ( j <d [ i ] ) then
            c [ i ] [ j ] = c[j-1, j];
        else
            c [ i ] [ j ] = min ( c[i-1,j], 1+c [i,j-d[i]]) ;
        }
}
return c [ n, N]
```

Initialising first column by 0

Using this nested for loops the table values are computed

Total number of coins in the change

Analysis - As the basic operation in above algorithms is to fill up the table, which is performed within nested for loops. Hence the time complexity of this algorithm is $\theta(nN)$.

4.5 Assembly Line Scheduling

GTU : June-11, Winter-14, Summer-15, Marks 7

This example of manufacturing problem is based on dynamic programming. The problem of assembly line scheduling can be described as follows -

In an automobile factory, the automobiles are produced using assembly lines. The chassis enters each assembly line and along this line path there are various stations at which the parts are added. Then a finished auto exits at the end of the line. The problem is to determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time through the factory for one auto. The structure of assembly lines along with the stations is as shown in Fig. 4.5.1.

4.6 Knapsack Problem

GTU : Dec.-10, May-12, Summer-13,14,15, Winter-14, Marks 8

As we know that dynamic programming is a technique for solving problems with overlapping subproblems. These subproblems are typically based on recurrence relation. In this section we will discuss the method of solving **knapsack problem** using dynamic programming approach. The knapsack problem can be defined as follows : If there are n items with the weights $w_1, w_2, \ldots w_n$ and values (profit associated with each item) $v_1, v_2, \ldots v_n$ and capacity of knapsack to be W, then find the most valuable subset of the items that fit into the knapsack.

To solve this problem using dynamic programming we will write the recurrence relation as :

$$\text{table}[i, j] = \text{maximum}\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\} \text{ if } j \geq w_i$$

or

$$\text{table}[i-1, j] \text{ if } j < w_i$$

That means, a table is constructed using above given formula. Initially, table [0, j] = 0 as well as table [i, 0] = 0 when $j \geq 0$ and $i \geq 0$.

The initial stage of the table can be

	0	1		$j-w_i$		j		W
0	0	0	...	0	...	0	...	0
	:			table $[i-1, j-w_i]$				
i – 1	0					table $[i-1, j]$		
i	0					table $[i, j]$		
	:							
n	0							table [n, W]

Goal i.e., maximum value of items

The table [n, W] is a goal i.e., it gives the total items sum of all the selected items for the knapsack.

From this goal value the selected items can be traced out. Let us solve the knapsack problem using the above mentioned formula -

Example 4.6.1 *For the given instance of problem obtain the optimal solution for the knapsack problem.*

Item	Weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

The capacity of knapsack is W = 5.

Solution : Initially, table [0, j] = 0 and table [i, 0] = 0. There are 0 to n rows and 0 to W columns in the table.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Now we will fill up the table either row by row or column by column. Let us start filling the table **row by row** using following formula :

$$\text{table}[i, j] = \begin{cases} \text{maximum}\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i] \text{ when } j \geq w_i\} \\ \text{or} \\ \text{table}[i-1, j] \text{ if } j < w_i \end{cases}$$

table [1, 1] With i = 1, j = 1, $w_i = 2$ and $v_i = 3$.

As $j < w_i$ we will obtain table [1, 1] as

table [1, 1] = table [i – 1, j]

= table [0, 1]

$\therefore$ table [1, 1] = 0

table [1, 2] With i = 1, j = 2, $w_i = 2$ and $v_i = 3$

As $j \geq w_i$ we will obtain table [1, 2] as

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0					
3	0					
4	0					

Now let us fill up next row of the table.

table [2, 1] With $i = 2$, $j = 1$, $w_i = 3$ and $v_i = 4$

As $j < w_i$, we will obtain table [2, 1] as

table [2, 1] = table [i − 1, j]

= table [1, 1]

∴ table [2, 1] = 0

table [2, 2] With $i = 2$, $j = 2$, $w_i = 3$ and $v_i = 4$

As $j < w_i$, we will obtain table [2, 2] as

table [2, 2] = table [i − 1, j]

= table [1, 2]

∴ table [2, 2] = 3

table [2, 3] With $i = 2$, $j = 3$, $w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 3] as

table [2, 3] = maximum $\{table[i-1, j], v_i + table[i-1, j-w_i]\}$

= maximum {table [1, 3], 4 + table [1, 0]}

= maximum {3, 4 + 0}

∴ table [2, 3] = 4

table [2, 4] With $i = 2$, $j = 4$, $w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 4] as

table [2, 4] = maximum $\{table[i-1, j], v_i + table[i-1, j-w_i]\}$

= maximum {table [1, 4], 4 + table [1, 1]}

= maximum {3, 4 + 0}

∴ table [2, 4] = 4

table [2, 5] With i = 2, j = 5, $w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 5] as

table [2, 5] = maximum {table[i−1, j], v_i + table[i−1, j− w_i]}

= maximum {table [1, 5], 4 + table [1, 2]}

= maximum {3, 4 + 3}

∴ table [2, 5] = 7

The table with these computed values will be

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

table [3, 1] With i = 3, j = 1, $w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 1] as

table [3, 1] = table [i − 1, j]

= table [2, 1]

∴ table [3, 1] = 0

table [3, 2] With i = 3, j = 2, $w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 2] as

table [3, 2] = table [i − 1, j]

= table [2, 2]

∴ table [3, 2] = 3

Table [3, 3] with $i = 3$, $j = 3$, $w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 3] as

table [3, 3] = table [i − 1, j]

= table [2, 3]

∴ table [3, 3] = 4

Table [3, 4] With $i = 3$, $j = 4$, $w_i = 4$ and $v_i = 5$

As $j \geq w_i$, we will obtain table [3, 4] as

table [3, 4] = maximum $\{table[i-1, j], v_i + table[i-1, j-w_i]\}$

= maximum {table [2, 4], 5 + table [2, 0]}

= maximum {4, 5 + 0}

∴ table [3, 4] = 5

Table [3, 5] With $i = 3$, $j = 5$, $w_i = 4$ and $v_i = 5$

As $j \geq w_i$, we will obtain table [3, 5] as

table [3, 5] = maximum $\{table[i-1, j], v_i + table[i-1, j-w_i]\}$

= maximum {table [2, 5], 5 + table [2, 1]}

= maximum {7, 5 + 0}

∴ table [3, 5] = 7

The table with these values can be

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0					

table [4, 1] With $i = 4$, $j = 1$, $w_i = 5$, $v_i = 6$

As $j < w_i$, we will obtain table [4, 1] as

$$\text{table [4, 1]} = \text{table [i - 1, j]}$$
$$= \text{table [3, 1]}$$

$\therefore$ **table [4, 1] = 0**

table [4, 2] With $i = 4$, $j = 2$, $w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 2] as

$$\text{table [4, 2]} = \text{table [i - 1, j]}$$
$$= \text{table [3, 2]}$$

$\therefore$ **table [4, 2] = 3**

table [4, 3] With $i = 4$, $j = 3$, $w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 3] as

$$\text{table [4, 3]} = \text{table [i - 1, j]}$$
$$= \text{table [3, 3]}$$

$\therefore$ **table [4, 3] = 4**

table [4, 4] with $i = 4$, $j = 4$, $w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 4] as

$$\text{table [4, 4]} = \text{table [i - 1, j]}$$
$$= \text{table [3, 4]}$$

$\therefore$ **table [4, 4] = 5**

table [4, 5] With $i = 4$, $j = 5$, $w_i = 5$ and $v_i = 6$

As $j \geq w_i$, we will obtain table [4, 5] as

$$\text{table [4, 5]} = \text{maximum } \{\text{table [i-1, j]}, v_i + \text{table}[i-1, j-w_i]\}$$
$$= \text{maximum } \{\text{table [3, 5]} , 6 + \text{table [3, 0]}\}$$
$$= \text{maximum } \{7, 6 + 0\}$$

$\therefore$ **table [4, 5] = 7**

Thus the table can be finally as given below

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

→ This is the total value of selected items

How to find actual knapsack items ?

Now, as we know that table [n, W] is the total value of selected items, that can be placed in the knapsack. Following steps are used repeatedly to select actual knapsack item

```
Let, i = n and k = W then
while (i>0 and k>0)
{
    if(table [i,k] ≠ table[i – 1,k])then
    mark i^th item as in knapsack
    i = i–1 and k=k – w_i          //selection of i^th item
  else
    i = i–1  //do not select i^th item
}
```

Let us apply these steps to the above given problem. As we have obtained the final table as.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

→ Start from here

i = 4 and k = 5

i.e., table [4, 5] = table [3, 5]

∴ Do not select i^{th} i.e., 4^{th} item.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Now set $i = i - 1$

$i = 3$

Capacity →

items ↓	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

As table [i, k] = table [i – 1, k]

i.e., table [3, 5] = table [2, 5]

∴ Do not select i^{th} item i.e., 3^{rd} item.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
✓2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Now set $i = i - 1 = 2$

As table [i, k] ≠ table [i – 1, k]

i.e. table [2, 5] ≠ table [1, 5]

Select i^{th} item.

That is, select 2^{nd} item.

Set $i = i - 1$ and $k = k - w_i$

i.e. $i = 1$ and $k = 5 - 3 = 2$

	0	1	2	3	4	5
0	0	0	0	0	0	0
✓1	0	0	3	3	3	3
✓2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

As table [i, k] ≠ table [i – 1, k]

i.e. table [1, 2] ≠ table [0, 2]

Select i^{th} item.

That is select 1^{st} item.

Set $i = i - 1$ and $k = k - w_i$

i.e. $i = 0$ and $k = 2 - 2 = 0$

Thus we have selected **item 1** and **item 2** for the knapsack. This solution can also be represented by solution vector **(1, 1, 0, 0).**

Example 4.6.2 *Solve the following 0/1 Knapsack problem using dynamic programming. There are five items whose weights and values are given in following arrays.*

Weight w[] = { 1, 2, 5, 6, 7 }

Value v [] = { 1, 6, 18, 22, 28 }

Show your equation and find out the optimal Knapsack items for weight capacity of 11 units.

GTU : Dec.-10, Marks 8

Solution :

Let,

Item	Weight	Value
1	1	1
2	2	6

0	0	0	0	0	0	0	0	0	0	0	0	0
1	0											
2	0											
3	0											
4	0											
5	0											

Now we will fill up table row by row using following formula -

$$\text{table}[i, j] = \begin{cases} \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i] \text{ when } j \geq w_i\} \\ \text{or} \\ \text{table}[i-1, j] \text{ if } j < w_i \end{cases}$$

$i = 1, \quad j = 1, \quad w_i = 1, \quad v_i = 1$

$$\text{table}[i, j] = \max\begin{Bmatrix} \text{table}[i-1, j], \\ v_i + \text{table}[i-1, j-w_i] \end{Bmatrix} \quad \because j \geq w_i$$

$$= \max\{0, 1+0\}$$

$$\text{table}[1, 1] = 1$$

$i = 1, \quad j = 2, \quad w_i = 1, \quad v_i = 1$

$$\text{table}[i, j] = \max\begin{Bmatrix} \text{table}[i-1, j], \\ v_i + \text{table}[i-1, j-w_i] \end{Bmatrix} \quad \because j \geq w_i$$

$$= \max\{0, 1+0\}$$

$$\text{table}[1, 2] = 1$$

$i = 1, \quad j = 3, \quad w_i = 1, \quad v_i = 1$

$$table[i, j] = \max\begin{Bmatrix} table[i-1, j], \\ v_i + table[i-1, j-w_i] \end{Bmatrix} \quad \because j \geq w_i$$

$$= \max\{0, 1+0\}$$

$table[1, 3] = 1$

$i = 1, \quad j = 4, \quad w_i = 1, \quad v_i = 1$

$$table[i, j] = \max\begin{Bmatrix} table[i-1, j], \\ v_i + table[i-1, j-w_i] \end{Bmatrix} \quad \because j \geq w_i$$

$$= \max\{0, 1+0\}$$

$table[1, 4] = 1$

Continuing in this fashion we can compute all the values of table.The table will be -

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

This is the total value of selected items

0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1
0	1	6	7	7	7	7	7	7	7	7	7
0	1	6	7	7	18	19	24	25	25	25	25
0	1	6	7	7	18	22	24	28	29	29	40
0	1	6	7	7	18	22	28	29	34	35	40

4, 11] = table [5, 11]

select i^{th} i.e. 5^{th} item.

= i – 1

= 4

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1
0	1	6	7	7	7	7	7	7	7	7	7
0	1	6	7	7	18	19	24	25	25	25	25
0	1	6	7	7	18	22	24	28	29	29	40
0	1	6	7	7	18	22	28	29	34	35	40

$i = i-1, \quad k = k - w_i$

$i = 4-1, \quad k = 11-6$

$i = 3, \quad k = 5$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
(3)	0	1	6	7	7	18	19	24	25	25	25	25
(4)	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

table [3, 5] ≠ table [2, 5]

∴ Select 3rd item. (i.e. i^{th} item.)

$i = i-1, \quad k = k - w_i$

$= 5 - 5$

i.e. $i = 2, \quad k = 0$

table [2, 0] = 0.

Thus we have selected **item 3** and **item 4,** for the Knapsack.

The total profit $= 18 + 22 = 40.$

Example 4.6.3 *Solve the following problem using dynamic method. Write the equation for solving above problem.* **GTU : May-12, Marks 8**

n = 5 , W = 100					
Object →	1	2	3	4	5
Weight (w) →	10	20	30	40	50
Value (v)→	20	30	66	40	60

Solution : We have to build a table for 0 to n rows and with 0 to W columns. i.e. For 0 to 5 rows and for 0 to 100 columns. But practically designing a table of 0 to 100 column is complicated.

Hence we will divide each weight by 10 and capacity (W) by 10. Hence new data that can be considered for processing will be

w[i]	v[i]
1	20
2	30
3	66
4	40
5	60

Now we will fill up the table row by row, using following formula :

$$\text{table } [i, j] = \begin{cases} \text{maximum } \{\text{table}[i-1,j],\ v_i + \text{table}[i-1, j-w_i] \text{ when } j \geq w_i\} \\ \text{or} \\ \text{table}[i-1,j] \text{ if } j < w_i \end{cases}$$

The table can be computed as follows -

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	20	20	20	20	20	20	20	20	20	20
2	0	20	30	50	50	50	50	50	50	50	50
3	0	20	30	66	86	96	116	116	116	116	116
4	0	20	30	66	86	96	116	116	126	136	156
5	0	20	30	66	86	96	116	116	126	146	156

The items that can be selected as item 1, item 2, item 3 and item 4. Hence solution for this knapsack problem is (1, 1, 1, 0) with profit **156.**

Max items allowed	Max weight					
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

The maximum profit is table [4, 5] = $ 37

As table [4, 5] ≠ table [3, 5]. Item 4 is included.

Total capacity 5–2 = 3 .

Now table [3, 3] = table [2, 3]. Item 3 is not included.

Now table [2, 3] ≠ table [1, 3]. Item 2 is included.

Remaining weight 3 – 1 = 2

This specifies to select item 1.

∴ Solution = {item 1, item 2, item 4}

Ans. :

Let, W = 4

w_i	v_i
1	3
3	4
4	5

Initially table [0, j] = 0 and table [i, 0] = 0.

There are 0 to n rows and 0 to W columns in the table.

	0	1	2	3	4
0	0	0	0	0	0
1	0				
2	0				
3	0				

We will fill up the table row by row by using following formula.

$$\text{table}[i, j] = \begin{cases} \text{maximum}\{\text{table}[i-1, j], v_i + \text{table}[i-i, j-w_i]\} & \text{when } j \geq w_i \\ \text{or} & \\ \text{table}[i-1, j] \text{ if} & j < w_i \end{cases}$$

table [1, 1] with i = 1, j = 1, w_i = 1, v_i = 3

As $j \geq w_i$

$$\text{table}[1, 1] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[0, 1], 3 + \text{table}[0, 0]\}$$

table [1, 1] = 3

table [1, 2] with i = 1, j = 2, w_i = 1, v_i = 3

As $j \geq w_i$

$$\text{table}[1, 2] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[0, 2], 3 + \text{table}[0, 1]\}$$
$$= \max\{0, 3+0\}$$

table [1, 2] = 3

table [1, 3] with i = 1, j = 3, $w_i = 1$, $v_i = 3$

$j > w_i$

As table [1, 3] = max { table [i − 1, j], v_i + table [i − 1, j − w_i]}

= max { table [0, 3], 3 + table [0, 2]}

table [1, 3] = 3

table [1, 4] with i = 1, j = 4, $w_i = 1$, $v_i = 3$

$j > w_i$

As table [1, 4] = max { table [i − 1, j], v_i + table [i − 1, j − w_i]}

= max { table [0, 4], 3 + table [0, 3]}

= max {0, 3 + 0}

table [1, 4] = 3

The table can be represented as

	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
2	0				
3	0				

table [2, 1] with i = 2, j = 1, $w_i = 3$, $v_i = 4$

As $j < w_i$

table [2, 1] = table [i − 1, j]

= table [1, 1]

table [2, 1] = 0

table [2, 2] with i = 2, j = 2, $w_i = 3$, $v_i = 4$

As $j < w_i$

table [2, 2] = table [i−1, j]

= table [1, 2]

table [2, 2] = 3

table [2, 3] with i = 2, j = 3, $w_i = 3$, $v_i = 4$

As $j = w_i$

$$table[2, 3] = \max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max\{[1, 3], 4 + table[1, 0]\}$$
$$= \max\{3, 4+0\}$$

table [2, 3] = 4

table [2, 4] with i = 2, j = 4, $w_i = 3$, $v_i = 4$

As $j > w_i$

$$table[2, 4] = \max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max\{table[1, 4], 4 + table[1, 1]\}$$
$$= \max\{3, 4+3\}$$

table [2, 4] = 7

The table now is -

	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
2	0	0	3	4	7
3	0				

table [3, 1] with i = 3, j = 1, $w_i = 4$, $v_i = 5$

As $j < w_i$

$$table[3, 1] = table[i-1, j]$$
$$= table[2, 1]$$

table [3, 1] = 0

table [3, 2] with i = 3, j = 2, $w_i = 4$, $v_i = 5$

As $j < w_i$

$$table[3, 2] = table[i-1, j]$$
$$= table[2, 2]$$

table [3, 2] = 3

table [3, 3] with i = 3, j = 3, $w_i = 4$, $v_i = 5$

As $j < w_i$

$$table[3, 3] = table[i-1, j]$$
$$= table[2, 3]$$

...ble [3, 3] = 4

...ble [3, 4] with i = 3, j = 4, $w_i = 4$, $v_i = 5$

$j = w_i$

$$table[3, 4] = \max \{table[i-1, j], v_i + table[i-1, j-W_i]\}$$
$$= \max \{ table[2, 4], 5 + table[2, 0] \}$$
$$= \max \{7, 5 + 0\}$$

...ble [3, 4] = 7

...ow the complete table is -

	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
2	0	0	3	4	7
3	0	0	3	4	7

...ow we will find the actual items of Knapsack

Step 1 :

table [2, 4] = table [3, 4]

∴ Do not select $i^{th} = 3^{rd}$ item

	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
2	0	0	3	4	7
3	0	0	3	4	7

Step 2 :

Set $i = i - 1$

$\therefore$ $i = 2$

table [2, 4] ≠ table [1, 4]

$\therefore$ Select i^{th} i.e. 2^{nd} item

	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
(2)	0	0	3	4	7
3	0	0	3	4	7

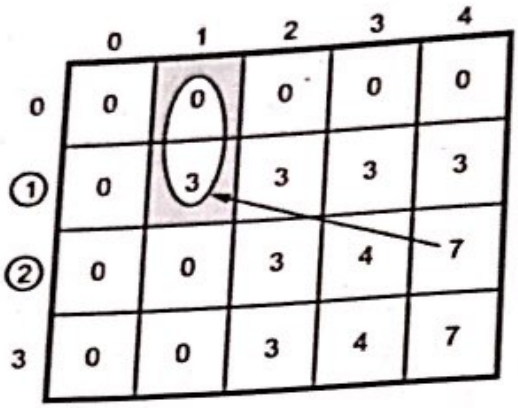
Step 3 :

$i = i - 1 = 1$

$j = j - w_i = 4 - 3 = 1$

table [1, 1] ≠ table [0, 1]

∴ Select i^{th} i.e. 1^{st} item



	0	1	2	3	4
0	0	0	0	0	0
1	0	3	3	3	3
2	0	0	3	4	7
3	0	0	3	4	7

Step 4 :

$$i = i-1=0$$

$$j = j-w_i=1-1=0$$

Thus we have selected item 1 and item 2

Total maximum profit = 3 + 4 = 7

Dynamic programming uses both top down and bottom up approaches. The top down method uses memorization technique while bottom up method uses tabulation technique.

Example 4.6.6 *Solve the following knapsack problem with the given capacity W = 5 using dynamic programming.* **GTU : Winter-14, Marks 7**

Item	Weight	Value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

Now we will fill up the table row by row. Let us start filling the table row by row using following formula :

$$table[i, j] = \begin{cases} \text{maximum } \{table[i-1, j], v_i + table[i-1, j-w_i]\} & \text{when } j \geq w_i \\ \text{or} & \\ table[i-1, j] \text{ if } j < w_i & \end{cases}$$

table [1, 1] with $i = 1, j = 1$, and $w_i = 2, v_i = 12$

As $j < w_i$

$$table[1, 1] = table[i-1, j]$$
$$= table[0, 1]$$

$$table[1, 1] = 0$$

table [1, 2] with $i = 1, j = 2, w_i = 2, v_i = 12$

As $j \geq w_i$

$$table[1, 2] = \max \{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max \{0, 12+0\}$$

$\therefore$ table [1, 2] = 12

table [1, 3] with $i = 1, j = 3, w_i = 2, v_i = 12$

As $j \geq w_i$

$$table[1, 3] = \max \{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max \{0, 12 + 0\}$$

$\therefore$ table [1, 3] = 12

table [1, 4] with $i = 1, j = 4, w_i = 2$ and $v_i = 12$

As $j \geq w_i$

$$table[1, 4] = \max \{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max \{0, 12+0\}$$

$\therefore$ **table [1, 4] = 12**

table [1, 5] with $i = 1$, $j = 5$, $w_i = 2$ and $v_i = 12$

As $j \geq w_i$

$$\text{table}[1,5] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$

$$= \max\{0, 12+0\}$$

$\therefore$ **table [1, 5] = 12**

The table with these values will be.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

table [2, 1] with $i = 2$, $j = 1$, $w_i = 1$, $v_i = 10$

As $j \geq w_i$

$$\text{table}[2,1] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$

$$= \max\{0, 10+0\}$$

$\therefore$ **table [2, 1] = 10**

table [2, 2] with $i = 2$, $j = 2$, $w_i = 1$, $v_i = 10$

As $j \geq w_i$

$$\text{table}[2,2] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$

$$= \max\{\text{table}[1,2]\ 10 + \text{table}[1,1]\}$$

$$= \max\{12, 10+0\}$$

table [2, 2] = 12

table [2, 3] with $i = 2$, $j = 3$, $w_i = 1$, $v_i = 10$

As $j \geq w_i$

$$\text{table}[2,3] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$

$$= \max\{\text{table}[1,3]\ 10 + \text{table}[1,2]\}$$

$$= \max\{12, 10+12\}$$

$\therefore$ **table [2, 3] = 22**

table [2, 4] with $i = 2, j = 4, w_i = 1, v_i = 10$

As $j \geq w_i$

$$table[2, 4] = \max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max\{table[1, 4], 10 + table[1, 3]\}$$

$$table[2, 4] = 22$$

table [2, 5] with $i = 2, j = 5, w_i = 1, v_i = 10$

As $j \geq w_i$

$$table[2, 5] = \max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max\{table[1, 5], 10 + table[1, 4]\}$$

$table[2, 5] = 22$. The partial table will be -

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

table [3, 1] with $i = 3, j = 1, w_i = 3, v_i = 20$

As $j \geq w_i$

$$table[3, 1] = table[i-1, j]$$
$$= table[2, 1]$$

$\therefore$ $table[3, 1] = 10$

table [3, 2] with $i = 3, j = 2, w_i = 3, v_i = 20$

As $j < w_i$

$$table[3, 2] = table[i-1, j]$$
$$= table[2, 2]$$

$\therefore$ $table[3, 2] = 12$

table [3, 3] with $i = 3, j = 3, w_i = 3, v_i = 20$

As $j \geq w_i$

$$table[3, 3] = \max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$$
$$= \max\{table[2, 3], 20 + table[2, 0]\}$$

$= \max \{22, 20+0\}$

$\therefore$ **table [3, 3] = 22**

table [3, 4] with $i = 3, j = 4, w_i = 3, v_i = 20$

As $j \geq w_i$

table [3, 4] = $\max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table } [2, 4] \; 20 + \text{table } [2, 1]\}$

$= \max \{22, 20+10\}$

$\therefore$ **table [3, 4] = 30**

table [3, 5] with $i = 3, j = 5, w_i = 3, v_i = 20$

As $j \geq w_i$

table [3, 5] = $\max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table } [2, 5] \; 20 + \text{table } [2, 2]\}$

$= \max \{22, 20+12\}$

$\therefore$ **table [3, 5] = 32**

The partially filled up table is as follows -

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

table [4, 1] with $i = 4, j = 1, w_i = 2, v_i = 15$

As $j < w_i$

table [4, 1] = table $[i-1, j]$

= table [3, 1]

$\therefore$ **table [4, 1] = 10**

table [4, 2] with $i = 4, j = 2, w_i = 2, v_i = 15$

As $j \geq w_i$

table [4, 2] = $\max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table } [3, 2] \; 15 + \text{table } [3, 0]\}$

$= \max \{12, 15+0\}$

$\therefore$ table [4, 2] = 15

table [4, 3] with $i = 4, j = 3, w_i = 2, v_i = 15$

As $j \geq w_i$

table [4, 3] $= \max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table}[3, 3], 15 + \text{table}[3, 1]\}$

$= \max \{22, 15+10\}$

$\therefore$ table [4, 3] = 25

table [4, 4] with $i = 4, j = 4, w_i = 2, v_i = 15$

As $j \geq w_i$

table [4, 4] $= \max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table}[3, 4], 15 + \text{table}[3, 2]\}$

$= \max \{30, 15+12\}$

$\therefore$ table [4, 4] = 30

table [4, 5] with $i = 4, j = 5, w_i = 2, v_i = 15$

As $j \geq w_i$

table [4, 5] $= \max \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$

$= \max \{\text{table}[3, 5], 15 + \text{table}[3, 3]\}$

$= \max \{32, 15+22\}$

$\therefore$ table [4, 5] = 37

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	(32)
4	0	10	15	25	30	(37)

Now we will trace for the solution using above table -

$i = 4, k = 4$

As table [i, j] $\neq$ table [i−1, k]

ie. tabl[ ] [ ]

Select i^{th} i.e. 4^{th} **item.**

Now $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 4 - 1 = 3$ and $k = 5 - 2 = 3$

As table [i, k] = table [i−1, k]

i.e. table [3, 3] = table [2, 3]

$\therefore$ Do not select i^{th} i.e. 3^{rd} item.

Now set $i = i - 1$

$\therefore$ $i = 2$ and $k = 3$.

As table [2, 3] $\neq$ table [1, 3]

$\therefore$ Select i^{th} i.e 2^{nd} **item.**

Set $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 1$ and $k = 3 - 1 = 2$.

As table [1, 2] $\neq$ table [0, 2]

Select i^{th} i.e. 1^{st} **item.**

Now set $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 0$ and $k = 0$

Thus solution to this Knapsack problem is **(item 1, item 2, item 4)** with total **profit = 37.**

Examples for Practice

Example 4.6.7 *Consider 0/1 knapsack problem N = 3, w = (4, 6, 8), p = (10, 12, 15) using dynamic programming devise the recurrence relations for the problem and solve the same. Determine the optimal profit for the knapsack of capacity 10.*

GTU : Dec.-11, 12

[Ans. : table [1, 1]=0, table [1, 2]= 0, table [1, 4]= 10, table [2, 1]= 0, table [2, 6]= 12, table [3, 10]= 22]

Example 4.6.8 *Using dynamic programming solve the following Knapsack instance :*

$n = 3, [w_1, w_2, w_3] = [1,2,2]$ and $[P_1, P_2, P_3] = [18,16,6]$ and $M = 4$

Let us now discuss the algorithm for the knapsack problem using dynamic programming.

```
Algorithm Dynamic_Knapsack(n,W,w[ ],v[ ])
//Problem Description : This algorithm is for obtaining knapsack
//solution using dynamic programming
//Input: n is total number of items, W is the capacity of
//knapsack, w[ ] stores weights of each item and v[ ] stores
//the values of each item.
//Output: Returns the total value of selected items for the
//knapsack.
for (i ← 0 to n) do
{
for (j ← 0 to W) do
{
  table[i,0]=0   // table initialization
  table[0,j]=0
}
}
for (i ← 0 to n) do
{
for (j ← 0 to W) do
{
 if(j<w[i]) then
   table[i,j]← table[i – 1, j]
 else if(j>=w[i]) then
   table[i,j]← max (table[i-1 ,j],(v[i]+table[i – 1,j – w[i]]) )
}
```

Thus $c(n) \approx W_n$

The time complexity of this algorithm is $\Theta(nW)$.

4.6.1 Memory Functions

While solving recurrence relation using dynamic programming approach common subproblems may be solved more than once and this makes inefficient solving of the problem. Hence memory function is a method that solves only necessary subproblems. Hence we can define the goal of memory function as : solve only subproblems that are **necessary** and solve these subproblems **only once.**

Memorization is a way to deal with overlapping subproblems in dynamic programming. During memorization -

1. After computing the solution to a subproblem, store it in a table.
2. Make use of recursive calls.

The 0-1 Knapsack Memory Function Algorithm is as given below -

Globally some values are to be set before the actual memory function algorithm

```
for (i←1 to n) do
 for (j←1 to W) do
   table[i,j]← –1
 for (j←0 to W) do
   table[0,j]← 0   //making 0th row 0
for i←1 to n do
    table[i,0]←0   //making 0th column 0
 Algorithm Mem_Fun_knapsack(i,j)
//Problem Description : Implementation of memory function method
//for the knapsack problem
//Input: i is the number of items and j denotes the knapsack's capacity
//Output : optimal solution subset
if (table[i,j] < 0) then
{
 if (j < w[i]) then
    value ← Mem_Fun_knapsack(i – 1, j)
 else
  value ← max(Mem_Fun_knapsack(i – 1, j),
    v[i] + Mem_Fun_knapsack(i – 1, j – w[i]))
  table[i,j]← value
}
return table[i, j]
```

Review Question

1. *Discuss and derive an equation for solving the 0/1 Knapsack problem using dynamic programming method. Design and analyze the algorithm for the same.* **GTU : Summer-15, Marks 7**

wt[3][3] 0

Computing All pair shortest path ...

D(0) =

0	8	5
2	0	999
999	1	0

D(1) =

0	8	5
2	0	7
999	1	0

D(2) =

0	8	5
2	0	7
3	1	0

D(3) =

0	6	5
2	0	7
3	1	0

Examples for Practice

Example 4.7.4 *Find the shortest path between all pairs of nodes in the following graph. (Refer Fig. 4.7.2)*

Example 4.7.5 *Find the shortest path using Floyd Warshall algorithm. (Refer Fig. 4.7.3)*

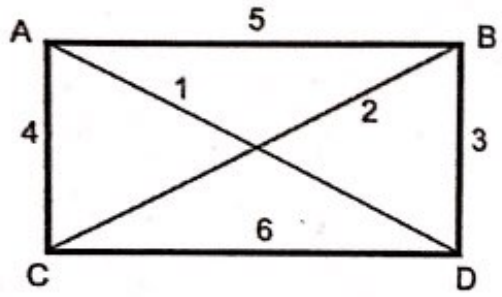


Fig. 4.7.2

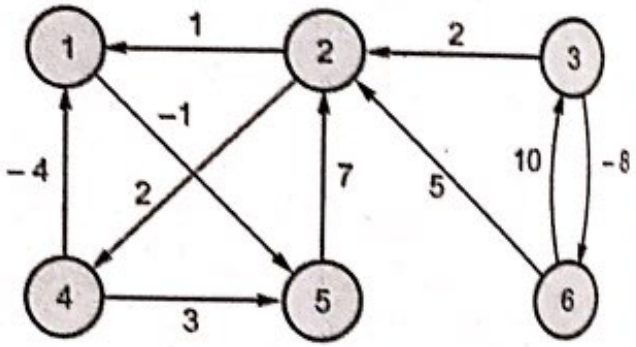


Fig. 4.7.3

4.8 Matrix Chain Multiplication

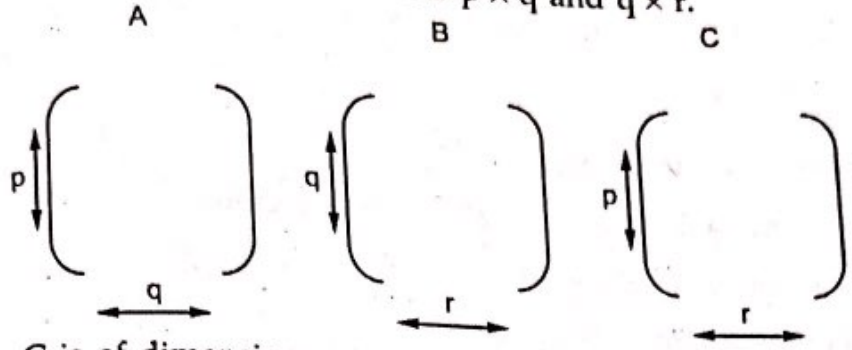
GTU : June-11,12, Dec.-11, Summer-15, Winter-14,15, Marks 8

- **Input :** n matrices $A_1, A_2, ..., A_n$ of dimensions $P_1 \times P_2$, $P_2 \times P_3$, ... , $P_n \times P_{n+1}$ respectively.
- **Goal :** To compute the matrix product $A_1A_2...A_n$

- **Problem :** In what order should $A_1A_2...A_n$ be multiplied so that it would take the minimum number of computations to derive the product.

For performing Matrix multiplication the cost is :

Let A and B be two matrices of dimensions $p \times q$ and $q \times r$.



Then $C = AB \cdot C$ is of dimensions $p \times r$.

Thus C_{ij} takes n scalar multiplications and $(n - 1)$ scalar additions.

Consider an example of the best way of multiplying 3 matrices:

Let A_1 of dimensions 5×4, A_2 of dimensions 4×6, and A_3 of dimensions 6×2.

$(A_1A_2)A_3$ takes $(5 \times 4 \times 6) + (5 \times 6 \times 2) = 180$

$A_1(A_2A_3)$ takes $(5 \times 4 \times 2) + (4 \times 6 \times 2) = 88$

Thus, $A_1(A_2A_3)$ is much cheaper to compute than $(A_1A_2)A_3$, although both lead to the same final answer. Hence optimal cost is 88.

To solve this problem using dynamic programming method we will perform following steps.

Step 1 : Decide the structure of optimal parenthsization

In this step we have to find the optimal substructure. This substructure is used to construct an optimal solution to the problem. Suppose that there is a production sequence of matrix $A_i\ A_{i+1} \ldots A_j$ such that $i < j$. Then in optimal parenthsization we have to split this sequence into $A_1 \ldots A_k$ and $A_{k+1} \ldots A_j$. The cost of parenthsization is obtained so that total computing cost can be decided. Here k is some integer value ranging from $i \leq k < j$. Hence **optimal substructure problem** can be described as follow : Consider that there is sequence $A_i\ A_{i+1} \ldots A_j$ which can be split into two products A_k and A_{k+1}. Then the parenthsization should produce subchain $A_i\ A_{i+1} \ldots A_k$ which is 1 of optimal cost. Thus optimal substructure is important for producing optimal solution. This suggests **bottom up** approach for computing optimal cost.

Step 2 : A recursive solution

For each i , which is within a range of 1 to n and for each j which is within a range of 1 to n the optimal cost m [i, j] is computed using following formula :

For all i, $1 \le i \le n$, $m[i, i] = 0$

and for all i and j such that

$1 \le i < j \le n$

$$m[i, j] = \min \{ m[i, k] + m[k + 1, j] + P_{i-1} P_k P_j \}$$

where $i \le k \le j - 1$

Step 3 : Computing optimal costs

We will construct m [i, j] table using following formula -

i) For i = 1 to n set m [i, i] = 0

ii) For l ← 2 to n compute m [i, j] using

$$m[i, j] = \min \{ m[i,k] + m[k+1, j] + P_{i-1} P_k P_j \}$$

with $i \le k \le j - 1$

Let us understand the procedure of computing optimal cost with some example -

Example 4.8.1 *Consider*

Matrix	Dimension
A_1	5×4
A_2	4×6
A_3	6×2
A_4	2×7

Compute matrix chain order. **GTU : Winter-2015, Marks 7**

Solution : Here $P_0 = 5$, $P_1 = 4$, $P_2 = 6$, $P_3 = 2$, $P_4 = 7$

For all i, $1 \le i \le n$

$$m[i, i] = 0$$

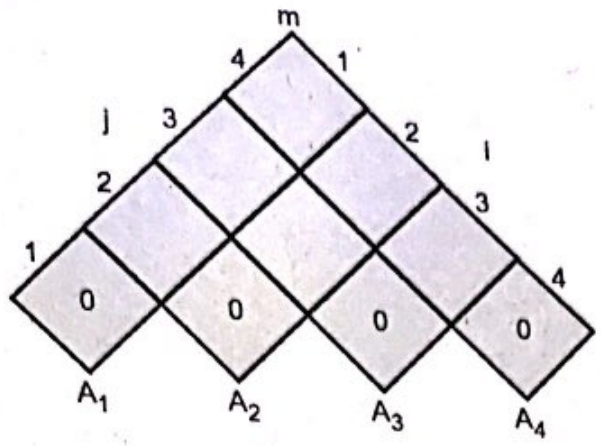
Hence $m[1, 1] = 0$

$m[2, 2] = 0$

$m[3, 3] = 0$

$m[4, 4] = 0$

Hence



Now we will fill up the table horizontally from left to right, assuming $i \le k \le j - 1$

Let $i = 1, j = 2, k = 1$

$$m[1,2] = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$$
$$= m[1,1] + m[2,2] + P_0 P_1 P_2$$
$$= 0 + 0 + 5 \times 4 \times 6$$

$$m[1,2] = 120$$

Let, $i = 2, j = 3, k = 2$.

$$m[2,3] = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$$
$$= m[2,2] + m[3,3] P_1 P_2 P_3$$
$$= 0 + 0 + 4 \times 6 \times 2$$

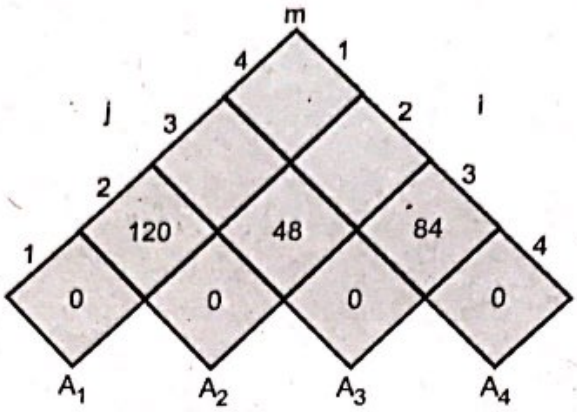
$$m[2,3] = 48$$

Let, $i = 3, \quad j = 4, \quad k = 3$

$$m[3,4] = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$$
$$= m[3,3] + m[4,4] + P_2 P_3 P_4$$
$$= 0 + 0 + 6 \times 2 \times 7$$

$$m[3,4] = 84$$

The table will be partially



Let $i = 1, j = 3$ $k = 1$ or $k = 2$

$$m[1,3] = \min \begin{cases} (m[1,1]+m[2,3]+P_0\ P_1\ P_3) \rightarrow k=1 \\ (m[1,2]+m[3,3]+P_0\ P_2\ P_3) \rightarrow k=2 \end{cases}$$

$$= \min \begin{cases} (0+48+5\times4\times2) \\ (120+0+5\times6\times2) \end{cases}$$

$$= \min \begin{cases} 48+40=88 & \text{when } k=1 \\ 120+60=180 \end{cases}$$

$$m[1,3] = 88$$

Let $i = 2, j = 4$, $k = 2$, or $k = 3$

$$m[2,4] = \min \begin{cases} (m[2,2]+m[3,4]+P_1\ P_2\ P_4 \\ (m[2,3]+m[4,4]+P_1\ P_3\ P_4 \end{cases} \begin{matrix} \rightarrow k=2 \\ \rightarrow k=3 \end{matrix}$$

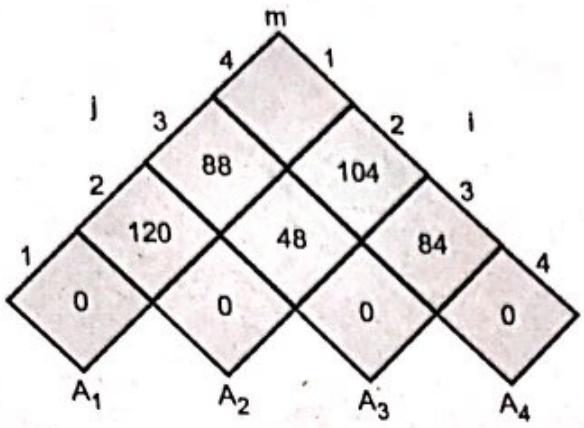
$$= \min \begin{cases} 0+84+4\times6\times7 \\ 48+0+4\times2\times7 \end{cases}$$

$$\min[2,4] = \min \begin{cases} 84+168 \\ 48+56 \end{cases}$$

$$= \min \begin{cases} 252 \\ 104 \rightarrow \text{when } k=3 \end{cases}$$

$$\min[2,4] = 104$$

table can be partially shown as :



Let $i = 1, j = 4$ then $k = 1$ or 2 or 3

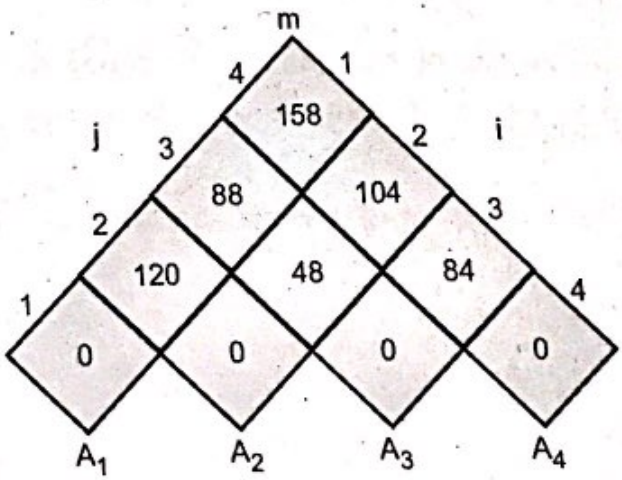
Now,

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 & \rightarrow k=1 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 & \rightarrow k=2 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 & \rightarrow k=3 \end{cases}$$

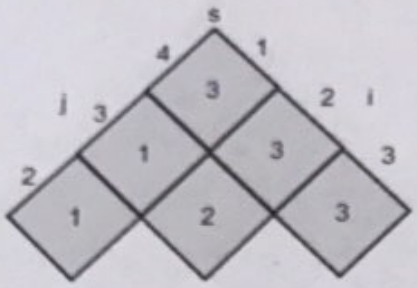
$$= \min \begin{cases} 0+104+5\times4\times7 \\ 120+84+5\times6\times7 \\ 88+0+5\times2\times7 \end{cases}$$

$$= \min \begin{cases} 244 \\ 414 \\ 158 \end{cases} \rightarrow \text{when } k=3$$

Hence the matrix table will be -



We will build the s table using s [i, j] = k



m [1, 4] determines the optimal cost i.e. **158**

Step 4 : Printing optimal parenthesization

The sequence of optimal parenthesization which gives the optimum cost is obtained using following algorithm.

```
Algorithm display_matrix_order ( s, i, j )
// Problem Description : This algorithm prints
// the optimal parenthsization of matrix
if ( i = j ) then
        print ( " A " i )
else
{ print " ( "
        display_matrix_order ( s, i, s [ i, j ] )
        display_matrix_order ( s, s [ i, j ] + 1 , j )
        print " ) "
} // end of else
} // end of algorithm
```

Using above algorithm we get the matrix chain order as (A_1 (A_2 A_3)) A_4

4.8.1 Algorithm

The algorithm for building the matrix chain table m[i,j] and s[i,j] is as follows -

```
Algorithm Matrix_chain(p[0...n])
{
//Problem Description: This algorithm is to build the table m[i,j] and s[i,j]
for (i ← 1 to n) do
   m[i,i] ← 0
for(len ← 2 to n) do
{
for (i←2 to (n – len+1) ) do
{
  j ← i+len – 1
```

```
m[i,j]=infinity
for (k ← i to j – 1) do
{
q ← m [i,k] +m[k+1,j]+p[i–1]*p[k]*p[j]
if (q<m[i,j]) then
{
  m[i,j]← q
  s[i,j] ← k
}//end of if
}//end of k's for loop
}//end of i's for loop
}//end of len's for loop
return m[1,n]
}//end of algorithm
```

Computing **m[i, j] and s[i,j]**

Analysis

The basic operation in this algorithm is computation of m[i,j] and s[i,j] which is within three nested for loops. Hence the time complexity of the above algorithm is $\theta(n^3)$.

Example 4.8.2 *Consider the chain of matrices A1, A2, ... A6 with the dimensions given below. Give the optimal parenthsization to get the product A2 ... A5.* **GTU : June-11, Marks 7**

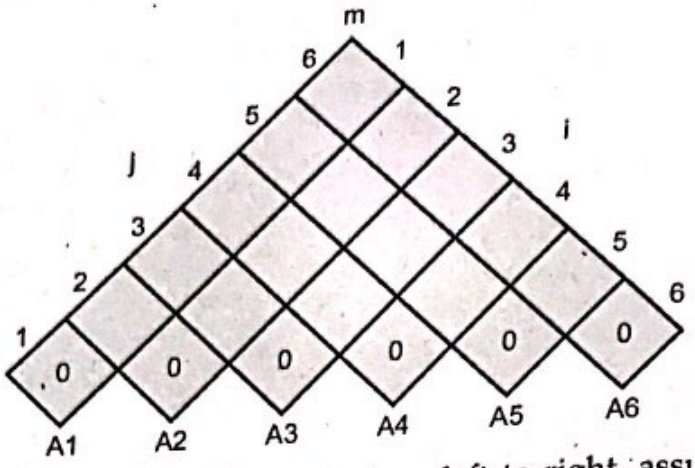
Matrix	Dimension
A1	30 × 35
A2	35 × 15
A3	15 × 5
A4	5 × 10
A5	10 × 20
A6	20 × 25

Solution : We will compute matrix chain order.

Here $P_0 = 30$, $P_1 = 35$, $P_2 = 15$, $P_3 = 5$, $P_4 = 10$, $P_5 = 20$, $P_6 = 25$

For all $1 \le i \le n$

$$m[i, i] = 0 \quad \therefore m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = m[5, 5] = m[6, 6] = 0$$



Now, we will fill up the table horizontally from left to right, assuming $i \le k \le j-1$

Let i = 1, j = 2, k = 1

$\therefore$ $m[1, 2] = m[i, k] + m[k+1, j] + P_{i-1} P_k P_j$

$= m[1, 1] + m[2,2] + P_0 P_1 P_2$

$= 0 + 0 + 30 * 35 * 15$

$= 15750.$ **with k = 2**

Similarly

$m[2, 3] = m[2, 2] + m[3, 3] + P_1 P_2 P_3$

$= 0 + 0 + 35 * 15 * 5$

$m[2, 3] = 2625$ with k = 2

$m[3, 4] = m[3, 3] + m[4, 4] + P_2 P_3 P_4$

$= 0 + 15 * 5 * 10$

$m[3, 4] = 750$ with k = 3

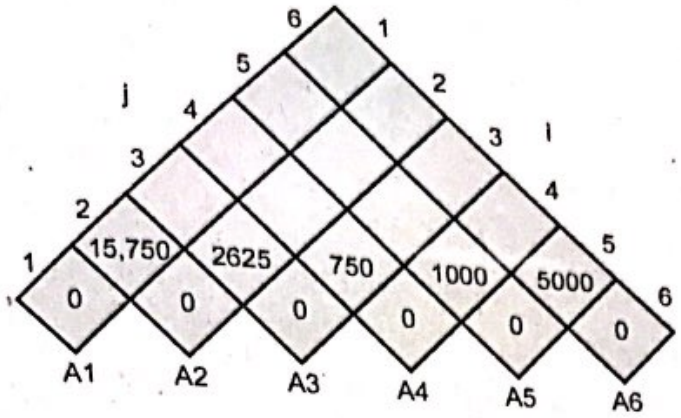
$m[4, 5] = m[4, 4] + m[5, 5] + P_3 P_4 P_5$

$m[4, 5] = 1000$ with k = 4

$m[5, 6] = m[5, 5] + m[6, 6] + P_4 P_5 P_6$

$m[5, 6] = 5000$ with k = 5

The table will be.



Now

$i = 1,\ j = 3\ \ k = 1$ or 2.

$$m[1, 3] = \min \begin{cases} m[1, 1] + m[2, 3] + P_0 P_1 P_2 = 0 + 2625 + (30 \cdot 35 \cdot 5) = 7875 \\ m[1, 2] + m[3, 3] + P_0 P_2 P_3 = 15750 + 0 + (30 \cdot 15 \cdot 5) = 18000 \end{cases}$$

$m[1, 3] = 7875$ **with k = 1**

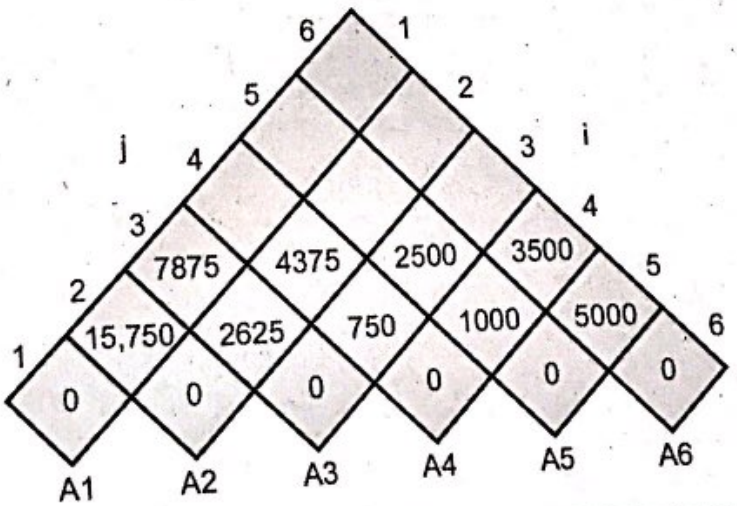
$$m[2, 4] = \min \begin{cases} m[2, 2] + m[3, 4] + P_1 P_2 P_4 = 0 + 750 + (35 \cdot 15 \cdot 10) = 6000 \\ m[2, 3] + m[4, 4] + P_1 P_3 P_4 = 2625 + 0 + (35 \cdot 5 \cdot 10) = 4375 \end{cases}$$

$m[2, 4] = 4375$ **with k = 3**

$$m[3, 5] = \min \begin{cases} m[3, 3] + m[4, 5] + P_2 P_3 P_5 = 0 + 1000 + (15 \cdot 5 \cdot 20) = 2500 \\ m[3, 4] + m[5, 5] + P_2 P_4 P_5 = 750 + 0 + (15 \cdot 10 \cdot 20) = 3750 \end{cases} \quad \begin{matrix} \textbf{min} = 2500 \\ \textbf{k} = 3 \end{matrix}$$

$$m[4, 6] = \min \begin{cases} m[4, 4] + m[5, 6] + P_3 P_4 P_6 = 0 + 5000 + (5 \cdot 10 \cdot 25) = 6250 \\ m[4, 5] + m[6, 6] + P_3 P_5 P_6 = 1000 + 0 + (5 \cdot 20 \cdot 25) = 3500 \end{cases} \quad \begin{matrix} \textbf{min} = 3500 \\ \textbf{k} = 5 \end{matrix}$$

The table will then be -



$$m[1,4] = \min\begin{cases} m[1,1]+m[2,4]+P_0P_1P_4 = 0+4375+(30\cdot35\cdot10) = 14875 \\ m[1,2]+m[3,4]+P_0P_2P_4 = 15750+750+(30\cdot15\cdot10) = 21000 \\ m[1,3]+m[4,4]+P_0P_3P_4 = 7875+0+(30\cdot5\cdot10) = 9375 \end{cases} \min = 9375 \quad k = 3$$

$$m[2,5] = \min\begin{cases} m[2,2]+m[3,5]+P_1P_2P_5 = 0+2500+(30\cdot15\cdot20) = 13000 \\ m[2,3]+m[4,5]+P_1P_3P_5 = 2625+1000+(35\cdot5\cdot20) = 5125 \\ m[2,4]+m[5,5]+P_1P_4P_5 = 4375+0+(35\cdot10\cdot20) = 11375 \end{cases} \min = 7125 \quad k = 3$$

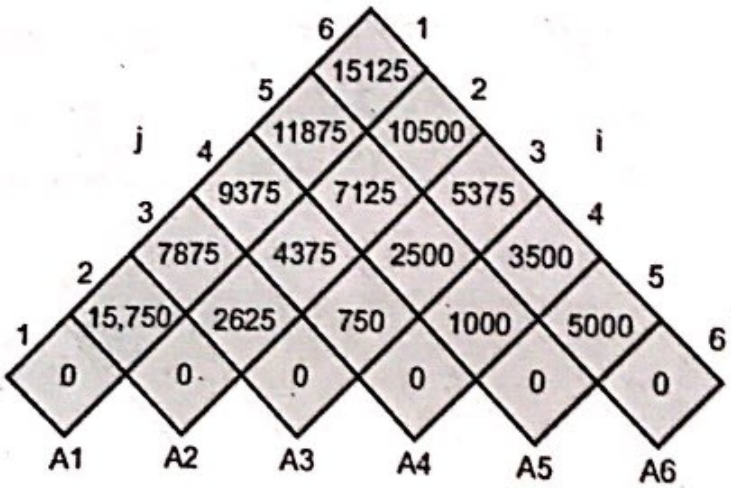
$$m[3,6] = \min\begin{cases} m[3,3]+m[4,6]+P_2P_3P_6 = 0+3500+(15\cdot5\cdot25) = 5375 \\ m[3,4]+m[5,6]+P_2P_4P_6 = 750+5000+(15\cdot10\cdot25) = 9500 \\ m[3,5]+m[6,6]+P_2P_5P_6 = 2500+0+(15\cdot20\cdot25) = 10000 \end{cases} \min = 5375 \quad k = 3$$

$$m[1,5] = \min\begin{cases} m[1,1]+m[2,5]+P_0P_1P_5 = 0+7125+(30\cdot35\cdot20) = 25125 \\ m[1,2]+m[3,5]+P_0P_2P_5 = 15750+2500+(30\cdot15\cdot20) = 27250 \\ m[1,3]+m[4,5]+P_0P_3P_5 = 7875+1000+(30\cdot5\cdot20) = 11875 \\ m[1,4]+m[5,5]+P_0P_4P_5 = 9375+0+(30\cdot10\cdot20) = 15375 \end{cases} \min = 11875 \quad k = 3$$

$$m[2,6] = \min\begin{cases} m[2,2]+m[3,6]+P_1P_2P_6 = 0+9375+(35\cdot15\cdot25) = 22500 \\ m[2,3]+m[4,6]+P_1P_3P_6 = 2625+3500+(35\cdot5\cdot25) = 10500 \\ m[2,4]+m[5,6]+P_1P_4P_6 = 4375+5000+(35\cdot10\cdot25) = 18125 \\ m[2,5]+m[6,6]+P_1P_5P_6 = 7125+0+(35\cdot20\cdot25) = 24625 \end{cases} \min = 10500 \quad k = 3$$

$$m[1,6] = \min\begin{cases} m[1,1]+m[2,6]+P_0P_1P_6 = 0+10500+(30\cdot35\cdot25) = 36750 \\ m[1,2]+m[3,6]+P_0P_2P_6 = 15750+9375+(30\cdot15\cdot25) = 36375 \\ m[1,3]+m[4,6]+P_0P_3P_6 = 7875+3500+(30\cdot5\cdot25) = 15125 \\ m[1,4]+m[5,6]+P_0P_4P_6 = 9375+5000+(30\cdot10\cdot25) = 21875 \\ m[1,5]+m[6,6]+P_0P_5P_6 = 11875+0+(30\cdot20\cdot25) = 26875 \end{cases} \min = 15125 \quad k = 3$$

The table will then be -



Solution : We will compute matrix chain order.

Here $P_0 = 5, P_1 = 10, P_2 = 3, P_3 = 12, P_4 = 5, P_5 = 50, P_6 = 6.$

Our first step is to set $m[i, i] = 0$. for $1 \le i \le 5$

Hence $m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0.$

Now

$$m[1, 2] = P_0 * P_1 * P_2 = 5 * 10 * 3 = 150$$

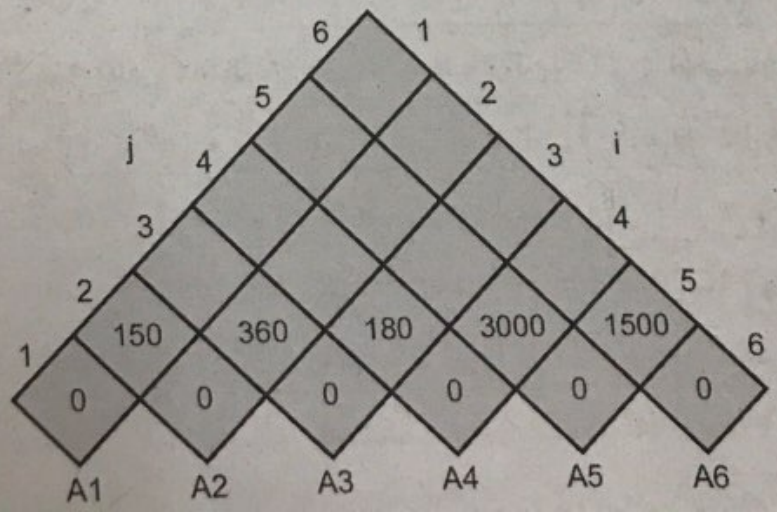
$$m[2, 3] = P_1 * P_2 * P_3 = 10 * 3 * 12 = 360$$

$$m[3, 4] = P_2 * P_3 * P_4 = 3 * 12 * 5 = 180$$

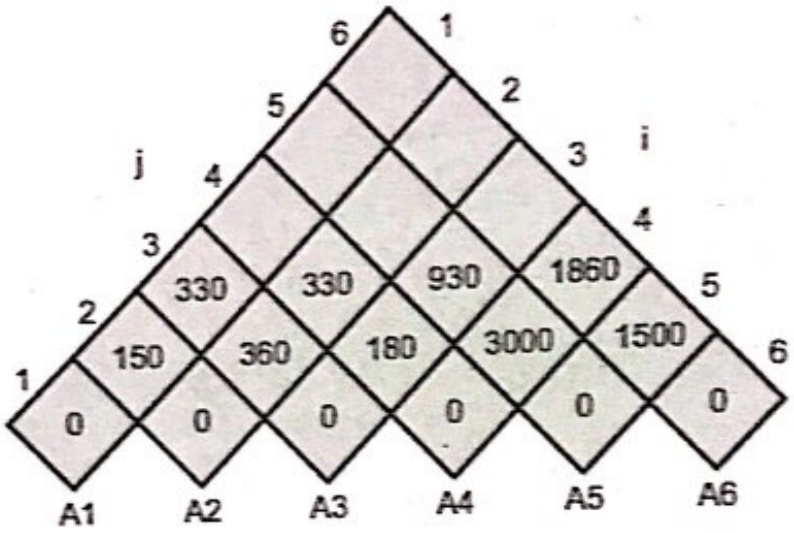
$$m[4, 5] = P_3 * P_4 * P_5 = 12 * 5 * 50 = 3000$$

$$m[5, 6] = P_4 * P_5 * P_6 = 5 * 50 * 6 = 1500$$

The table will then be



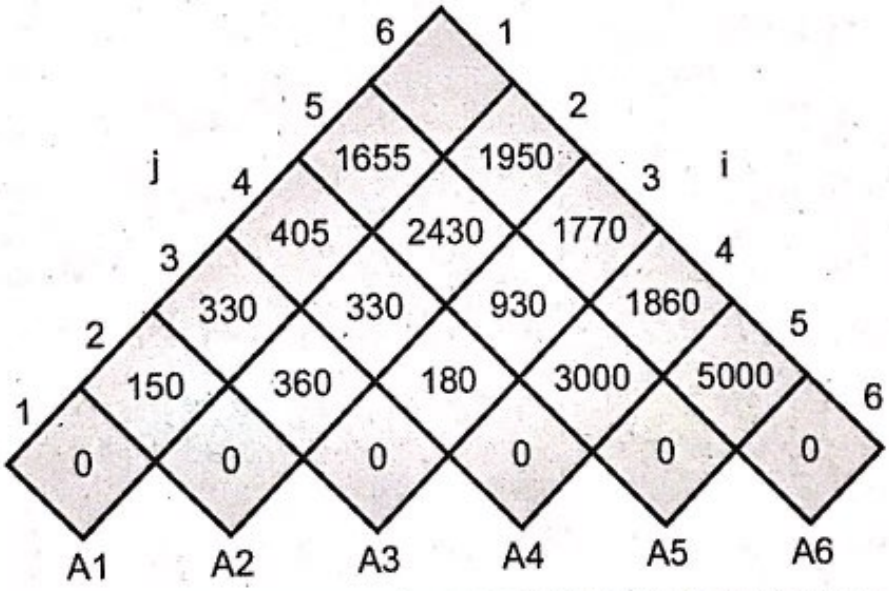
		= 0 + 360 + (5·10·12) = 960	min = 330
m[1, 3] = min	m [1,1] + m [2,3] + $P_0 P_1 P_3$	= 150 + 0 + (5·3·12) = 330	with k = 2
	m [1,2] + m [3,3] + $P_0 P_2 P_3$		
m[2, 4] = min	m [2,2] + m [3,4] + $P_1 P_2 P_4$	= 0 + 180 + (10·3·5) = 330	min = 330 k = 2
	m [2,3] + m [4,4] + $P_1 P_3 P_4$	= 360 + 0 + (10·12·50) = 6360	
m[3,5] = min	m [3,3] + m [4,5] + $P_2 P_3 P_5$	= 0 + 3000 + (3·12·50) = 4800	min = 930 k = 4
	m [3,4] + m [5,5] + $P_2 P_4 P_5$	= 180+ 0+ (3·5·50) = 930	
m[4, 6] = min	m [4,4] + m [5,6] + $P_3 P_4 P_6$	= 0 + 1500 + (12·5·6) = 1860	min = 1860 k = 4
	m [4,5] + m [6,6] + $P_3 P_5 P_6$	= 3000 + 0 + (12·50·6) = 6600	



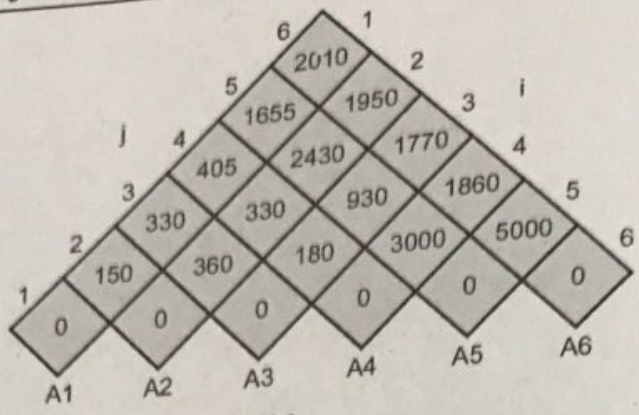
Now we will compute m[1, 4], m[2, 5] and m[3, 6]

m[1, 4] = min	m [1,1] + m [2,4] + $P_0 P_1 P_4$	= 0 + 330 + (5·10·5) = 580	min = 405 k = 2
	m [1,2] + m [3,4] + $P_0 P_2 P_4$	= 150 + 180 + (5·3·5) = 405	
	m[1,3] + m[4,4] + $P_0 P_3 P_4$	= 330 + 0 + (5·12·5) = 630	
m[2, 5] = min	m [2,2] + m [3,5] + $P_1 P_2 P_5$	= 0 + 930 + (10·3·50) = 2430	min = 2430 k = 2
	m [2,3] + m [4,5] + $P_1 P_3 P_5$	= 360 + 3000 + (10·12·50)= 9360	
	m[2,4] + m[5,5] + $P_1 P_4 P_5$	= 330 + 0 + (10·5·50) = 2830	
m[3, 6] = min	m [3,3] + m [4,6] + $P_2 P_3 P_6$	= 0 + 1860 + (3·12·6) = 2076	min = 1770 k = 4
	m [3,4] + m [5,6] + $P_2 P_4 P_6$	= 180 + 1500 + (3·5·6) = 1770	
	m [3,5] + m [6,6] + $P_2 P_5 P_6$	= 930 + 0 + (3·50·6) = 1830	

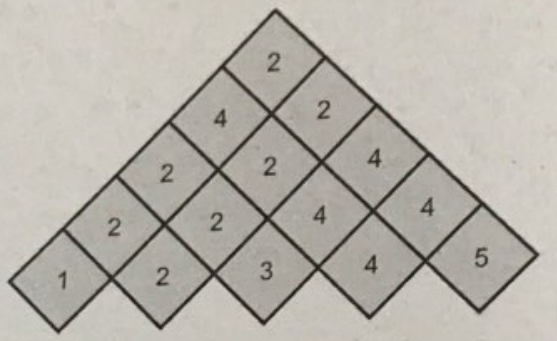
m[1, 5] = min	m[1, 1] + m[2, 5] + $P_0 P_1 P_5$	= 0 + 2430 + (5 · 10 · 50) = 4930	min = 1655 k = 4
	m[1, 2] + m[3, 5] + $P_0 P_2 P_5$	= 150 + 930 + (5 · 3 · 50) = 1830	
	m[1, 3] + m[4, 5] + $P_0 P_3 P_5$	= 330 + 3000 + (5 · 12 · 50) = 6330	
	m[1, 4] + m[5, 5] + $P_0 P_4 P_5$	= 405 + 0 + (5 · 5 · 50) = 1655	
m[2, 6] = min	m[2, 2] + m[3, 6] + $P_1 P_2 P_6$	= 0 + 1770 + (10 · 3 · 6) = 1950	min = 1950 k = 2
	m[2, 3] + m[4, 6] + $P_1 P_3 P_6$	= 360 + 1860 + (10 · 12 · 6) = 2940	
	m[2, 4] + m[5, 6] + $P_1 P_4 P_6$	= 330 + 1500 + (10 · 5 · 6) = 2130	
	m[2, 5] + m[6, 6] + $P_1 P_5 P_6$	= 2430 + 0 + (10 · 50 · 6) = 5430	



m[1,6] = min	m[1, 1] + m[2, 6] + $P_0 P_1 P_6$	= 0 + 1950 + (5 · 10 · 6) = 2250	min = 2010 k = 2
	m[1, 2] + m[3, 6] + $P_0 P_2 P_6$	= 150 + 1770 + (5 · 3 · 6) = 2010	
	m[1, 3] + m[4, 6] + $P_0 P_3 P_6$	= 330 + 1860 + (5 · 12 · 6) = 2550	
	m[1, 4] + m[5, 6] + $P_0 P_4 P_6$	= 405 + 1500 + (5 · 5 · 6) = 2055	
	m[1, 5] + m[6, 6] + $P_0 P_5 P_6$	= 1655 + 0 + (5 · 50 · 6) = 3155	



The s [i, j] table for the values of k will be



The optimal parenthesization will be

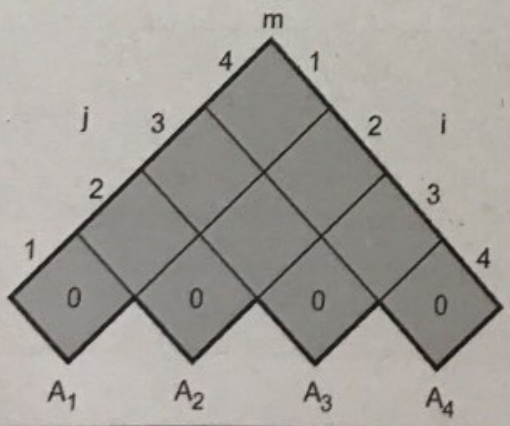
$(A1 * A2) * ((A3 * A4) * (A5 * 6))$

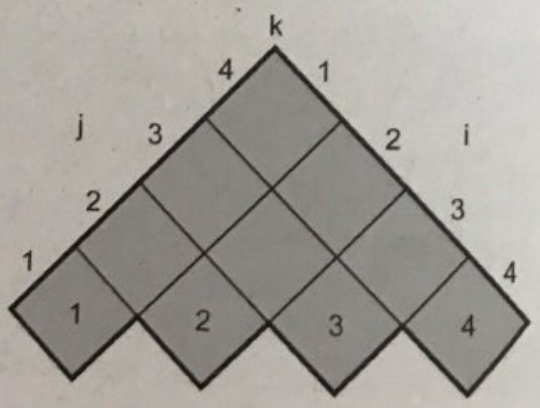
Example 4.8.4 *Using algorithm find an optimal parenthesization of a matrix chain product whose sequence of dimension is (13, 5, 89, 3, 34) (Use dynamic programming) .*

GTU : Winter-14, Marks 7,CSE

Solution : Here $P_0 = 13$, $P_1 = 5$, $P_2 = 89$, $P_3 = 3$, $P_4 = 34$ For all $1 \leq i \leq n$

$m [i, i] = 0 \therefore m [1, 1] = m [2, 2] = m [3, 3] = m [4, 4] = 0$ with value of k = 1, 2, 3 and 4 respectively Hence





m[1,3] Here value of k = 1 or k = 2	$\min\{m[i,k]+m[k+1,j]+P_{i-1}+P_kP_j\}$ where $i \le k \le j-1$	$0+1335+(13*5*3)$ $= 1530$	$\therefore [1, 3] = 1530$ with $k = 1$
	$m[1,1]+m[2,3]+P_0*P_1*P_3$		
	$m[1,2]+m[3,3]+P_0*P_2*P_3$	$5785+0+(13*89*3)$ $= 9256$	
		$\therefore \min\{1530,9256\}$ $= 1530$	
m[2, 4] Here value of k = 2 or k = 3	$m[2,2]+m[3,4]+P_1P_2P_4$	$0+9078+(5*89*34)$ $= 24208$	$\therefore m [2, 4] = 1845$ with $k = 3$
	$m[2,3]+m[4,4]+P_1P_3P_4$	$1335+0+(5*3*34)$ $= 1845$	

Now we will compute m [1, 4]

$i = 1, j = 4$ then $k = 1, 2$ or 3.

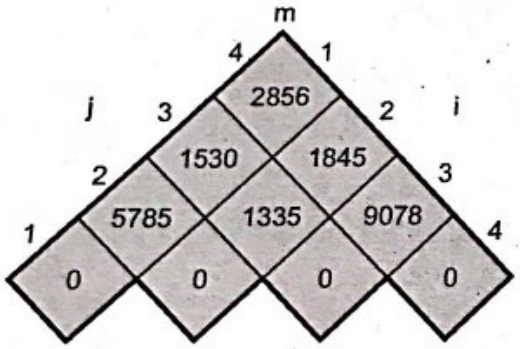
$$\therefore \quad m[1,4] = \min \begin{cases} m[i,k] + m[k+1, j] + P_{i-1} P_k P_j \\ m[1,1] + m[2,4] + P_0 P_1 P_4 \rightarrow k = 1 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \rightarrow k = 2 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \rightarrow k = 3 \end{cases}$$

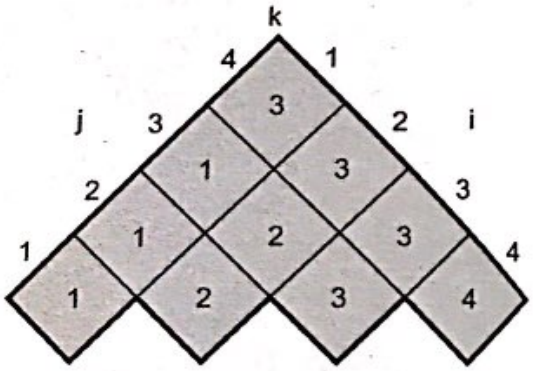
$$= \min \begin{cases} 0 + 1845 + (13 * 5 * 34) \\ 5785 + 9078 + (13 * 89 * 34) \\ 1530 + 0 + (13 * 3 * 34) \end{cases}$$

$$= \min \begin{cases} 4055 \\ 54201 \\ 2856 \end{cases}$$

$$m[1,4] = 2856 \text{ with } k = 3$$

Hence m-table and k table will be -





The matrix chain order will be $((A_1(A_2A_3))A_4)$.

Review Questions

1. *Design and analyze dynamic programming algorithm with and without memorization to solve matrix chain multiplication problem.* **GTU : June-11, Marks 8**
2. *Explain chained matrix multiplication with example.* **GTU : June-12, Marks 7**
3. *Discuss matrix multiplication problem using divide and conquer technique.* **GTU : Summer-15, Marks 7**

Let, $B = < b_1, b_2 \ldots\ldots b_n >$ and $A = < a_1, a_2 \ldots a_m >$ be strings over an alphabet. Then B is a subsequence of A if B can be generated by striking out some elements from A.

For example

$< x, z, y, x >$ is a subsequence of $< x, y, x, z, y, z, y, x, z >$

The **longest Common Subsequence** problem (LCS) is the problem of finding given two sequences $A = < a_1, a_2 \ldots a_m >$ and $B = < b_1, b_2, b_3, \ldots b_n >$ a maximum length common subsequence of A and B. Let us apply the steps of dynamic programming for obtaining Longest Common Subsequence (LCS)

Step 1 : Characterizing the longest common subsequence

If we apply the straightforward approach of obtaining common subsequence then for the sequence $A = < a_1, a_2 \ldots a_m >$ and $B = < b_1, b_2, \ldots b_n >$, all the sequences of A has to be checked against all the subsequences of B. But this approach will not useful for long sequences. Hence to solve the problem of LCS we need to define **optimal substructure** of LCS. Let us discuss one theorem based on optimal substructure of LCS.

Theorem : Let, $A = < a_1, a_2, \ldots a_m >$ and $B = < b_1, b_2, \ldots b_n >$ then,

i) If $a_m = b_n$, then longest common subsequence of A_{m-1} and B_{n-1} can be constructed by appending a_m ($= b_n$)to LCS of A and B.

ii) If $a_m \neq b_n$ then it implies

a) LCS of A_{m-1} and B or

b) LCS of A and B_{n-1}.

Proof : Consider $C = < c_1, c_2, c_3 \ldots c_k >$ be the LCS of A and B. Then if $c_k = a_m$ then we could append $a_m = b_n$ to C for getting the longest common subsequence. If necessary modify the production of C from either A or from B, so that its last element could be A_m or B_n. Then C becomes a common subsequence of string of A_{m-1} and B_{n-1}. Hence C with maximum length becomes the longest common subsequence.

If $c_k \neq a_m$, then for any LCS of C of A and B, generation of C cannot use both a_m and b_n. So C is either a_n LCS of A and B_{n-1} or it can be LCS of A_{m-1} and B.

Step 2 : A recursive definition

If $a_m = b_n$ then we should find LCS of A_{m-1} and B_{n-1} otherwise, compare LCS of A and B_{n-1} and LCS of A_{m-1} and B. Then from these sequences pick up the longer sequence.

```
Algorithm compute_LCS ( A, B )
// Problem Description : This algorithm builds c [ i, j ] and d [ i , j ] table.
m ← length ( A )
n ← length ( B )
for ( i ← 1 to m ) do
        c [ i, 0 ] ← 0 }
for  ( j ← 0 to n ) do
        c [ 0, j ] ← 0                                  } making first column zero
for ( i ← 1 to m ) do
{
        for ( i ← 1 to n ) do                          } making first row zero.
        {
        if ( a_i = b_i ) then
        {
        c [ i, j ] ← c [ i – 1, j – 1 ] + 1

                d [ i , j ] ← " ↖ " // Putting direction
        } else if ( c [ i – 1 , i ] ≥ c [ i, i – 1 ] then
        {
        c [ i, j ] ← c [ i – 1, j ]
        d [ i, j ] ← " ↑ "
        }
        else
        {
        c [ i, j ] ← c [ i, j – 1]
        d [ i, j ] ← " ← "
        }
}// end of inner for loop
}// end of outer for loop.
        return c and d
}
```

From the above algorithm let us solve some problem of computing longest commo subsequence.

Example 4.9.1 *Given two sequences of characters*
A=< X, Y, Z, Y, T, X, Y > and
B=< Y, T, Z, X, Y, X >
obtain longest common subsequence.

GTU : Winter-14, Marks 7

Solution : We have to obtain longest common subsequence. Arrange elements of A and B in the arrays as -

	1	2	3	4	5	6	7
A[i]	X	Y	Z	Y	T	X	Y
B [j]	Y	T	Z	X	Y	X	

Step 1 : We will build c [i, j] and d [i, j]. Initially c [i , 0] ← 0 where i represents row and c [0, j] ← 0 where j represents the column. Note that c table will store values and d table will store directions.

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Then

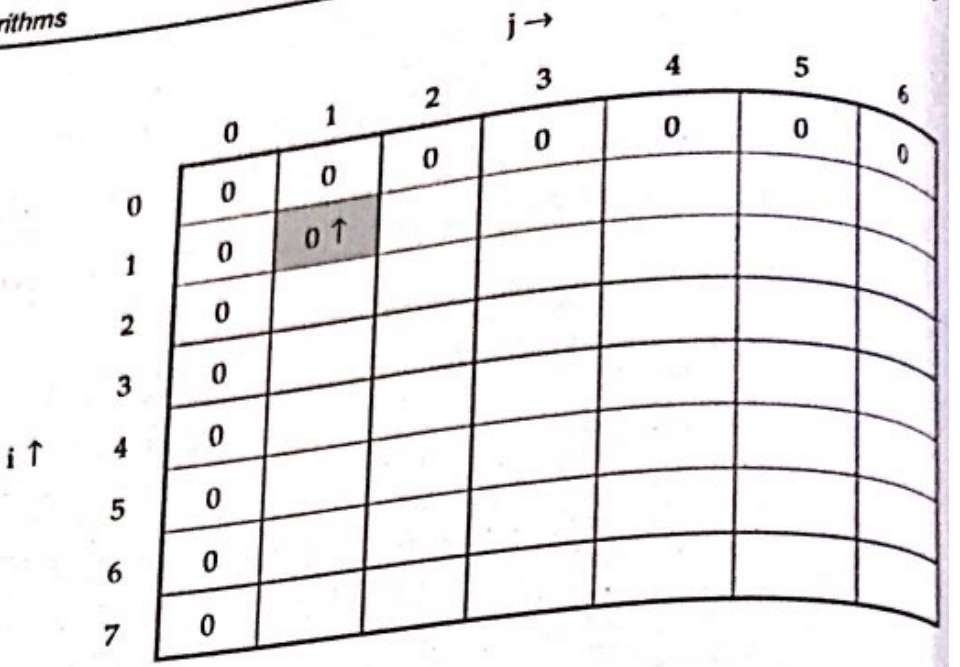
for (i ← 1 to m) and

for (j ← 1 to n)

We go on filling up c [i, j] and d [i, j] horizontally

∴ Let i = 1, j = 1

	1	2	3	4	5	6	7
A[i]	X	Y	Z	Y	T	X	Y
B[j]	Y	T	Z	X	Y	X	



As $A[i] \neq B[j]$

If $(c[0,1] \geq c[1,0])$ is true

$\therefore \quad 0 \geq 0$

$c[1,1] \leftarrow c[i-1, j]$

$\therefore \quad c[1,1] \leftarrow c[0,1]$

$\therefore \quad c[1,1] \leftarrow 0$

and $d[1,1] \leftarrow \uparrow$

Step 2 :

Let $i = 1, j = 2$ then

	1	2	3	4	5	6	7
A [i]	X	Y	Z	Y	T	X	y
B [j]	Y	T	Z	X	Y	X	

As $A[i] \neq B[j]$

then if $(c[i-1, j] \geq c[i, j-1]) \rightarrow$ is true

$\therefore \quad c[0,2] \geq c[1,1]$

i.e. $0 \geq 0$

Hence $c[i, j] \leftarrow c[i-1, j]$

$d[i, j] \leftarrow$ "↑"

We get c [1, 2] ← c [0, 2] i.e. 0

and d [i, j] ← "↑"

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	↑ 0	↑ 0				
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Continuing in this fashion we can fill up the table as follows -

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Now we can construct a longest common subsequence using c [i, j] and d [i , j].

```
{
// Problem Description : This algorithm
// points the longest common subsequence
        if (i = 0 || j = 0) then
                return
if ( d [ i, j ] = " ↖ " then
        {
                display LCS ( d, A, i – 1, j – 1 )
                print ( a_i )
}
        else if ( d [ i, j ] = " ↑ ") then
        {       display_ LCS ( d, A, i – 1, j )
         }
else
        display_LCS ( d, A, i, j – 1 )
}// end of algorithm
```

Let us apply this algorithm for obtaining LCS. Initially a call is given as :

```
display_LCS ( d, A [ ] ,7, 6 )
```

Here 7 is upper bound of i and 6 is the upper bound of j.

As d [7, 6] = ↑ we get recursive call display_ LCS (d, A, 6, 6). Then [d[6 ,6] = ↖] a recursive call display_LCS (d, A, 5, 5) is encountered. Then the algorithm tells us to print value of a_i which is a_6 . Thus if we follow the complete algorithm we get YZYX as LCS.

Example 4.9.2 *Explain how to find out longest common subsequence of two strings using dynamic programming method. Find any one Longest Common Subsequence of given two strings using dynamic programming.*

S1 = abbacdcba

S2 = bcdbbaac

GTU : Dec.-10, Marks 8 Winter-14,15, Marks 7

We will arrange these strings in arrays

	1	2	3	4	5	6	7	8	9
S1 [i]	a	b	b	a	c	d	c	b	a
S2 [j]	b	c	d	b	b	c	a	a	c

Step 1 : We will build c [i, j] and d [i, j]. Initially c [i, 0] ← 0 where i represents columns and c [0, j] ← 0 where j represents the rows.

The c table stores the values and d table stores the directions. The table will be

		i →		a	b	b	a	c	d	c	b	a
			0	1	2	3	4	5	6	7	8	9
		0	0	0	0	0	0	0	0	0	0	0
	b	1	0									
	c	2	0									
	d	3	0									
j ↓	b	4	0									
	b	5	0									
	c	6	0									
	a	7	0									
	a	8	0									
	c	9	0									

for (i ← 1 to m) and

for (j ← 1 to n)

We go on filling up c [i, j] and d [i, j] vertically

∴ Let i = 1, j = 1

	1	2	3	4	5	6	7	8	9
S1 [i]	a	b	b	a	c	d	c	b	a
S2 [j]	b	c	d	b	b	c	a	a	c

S1 [i] ≠ S2 [j]

if (c [i - 1, j] ≥ c [i, j - 1]) true

i.e. if (c [0, 1] ≥ c [1, 0]) true

∴ 0 ≥ 0

c [1, 1] ← c [i - 1, j]

c [1, 1] ← c [0, 1]

c [1, 1] ← 0

d [1, 1] ← ↑

Step 2 :

Let i = 1, j = 2

S1 [i] = a and S2 [j] = c

i.e. S1 [i] ≠ S2 [j]

if (c [i - 1, j] ≥ c [i, j - 1]

i.e. if (c [0,2] ≥ c [1, 1]

0 ≥ 0 → is true

∴ c [1,2] ← c [i-1, j]

c [1,2] ← c [0,2]

∴c [1, 2] ← 0

d [1,2] ← ↑

Step 3 :

Let i = 1, j = 3

S1 [1] = a and S2 [3] = d

i.e. S1 [1] ≠ S2 [3]

if (c [i - 1, j] ≥ c [i, j - 1])

i.e. if (c [0, 3] ≥ c [1,2])

i.e. if (0 ≥ 0) → is true.

∴ c [1, 3] = c [i - 1, j]

= c [0, 3]

c [1, 3] = 0

d [1, 3] = ↑

Step 4 :

Let i = 1, j = 4

S1 [1] = a, S2 [4] = b

i.e. S1 [i] ≠ S2 [j]

if (c [i - 1, j] ≥ c [i, j - 1])

i.e. if (c[0, 4] : ≥ c[1, 3])

i.e. if (0 ≥ 0) → is true.

$$c[1,4] = c[i-1, j]$$

$$= c[0, 4]$$

$$c[1, 4] = 0$$

$$d[1, 4] = \uparrow$$

We will start filling up the table in this manner finally we will get.

		0	1 a	2 b	3 (b)	4 a	5 (c)	6 (d)	7 c	8 (b)	9 (a)
0		0	0	0	0	0	0	0	0	0	0
1	(b)	0	↑0	\1	\1	←1	←1	←1	←1	\1	←1
2	(c)	0	↑0	↑1	↑1	↑1	\2	←2	\2	←2	←2
3	(d)	0	↑0	↑1	↑1	↑1	↑2	\3	←3	←3	←3
4	b	0	↑0	\1	\2	←2	↑2	↑3	↑3	\4	←4
5	(b)	0	↑0	\1	\2	↑2	↑2	↑3	↑3	\4	↑4
6	c	0	↑0	↑1	↑2	↑2	\3	↑3	\4	↑4	↑4
7	a	0	\1	↑1	↑2	\3	↑3	↑3	↑4	↑4	\5
8	(a)	0	\1	↑1	↑2	\3	↑3	↑3	↑4	↑4	\5
9	c	0	↑1	↑1	↑2	↑3	\4	←4	\4	↑4	↑5

Thus the matching longest sub sequence is **bcdba**.

Example 4.9.3 *Using algorithm determine longest sequence of (A, B, C, D, B, A, C, D, F) and (C, B, A, F) (use dynamic programming).* **GTU : Dec.-11, Marks 7**

Solution : Let,

	1	2	3	4	5	6	7	8	9
A[i]	A	B	C	D	B	A	C	D	F
B [j]	C	B	A	F					

We will build c [i, j] and d [i, j]. Initially c [i , 0] ← 0 where i represents row and c [0, j] ← 0 where j represents the column. The c table will store some values and d table will store directions.

Step 1 :

	0	1 C	2 B	3 A	4 F
0		0	0	0	0
A 1	0				
B 2	0				
C 3	0				
D 4	0				
B 5	0				
A 6	0				
C 7	0				
D 8	0				
F 9	0				

Step 2 :

	1	2	3	4	5	6	7	8	9
A[i]	A	B	C	D	B	A	C	D	F
B [j]	C	B	A	F					

As $A[i] \neq B[j]$

If ($c[0,1] \geq c[1,0]$) is true

i.e. $0 \geq 0$

$c[1,1] = c[i-1, j]$

$c[1,1] = c[0,1]$

$c[1,1] = 0$

$d[1, 1] = \uparrow$

Step 3 :

Let $i = 1, j = 2$ then

	1	2	3	4	5	6	7	8	9
A[i]	A	B	C	D	B	A	C	D	F
B[j]	C	B	A	F					

$A[i] \neq B[j]$

As

If $(c[i-1, j] \geq c[i, j-1])$

i.e. $c[0, 2] \geq c[1, 1]$ is true i.e. $0 \geq 0$.

$c[i, j] = c[i-1, j]$

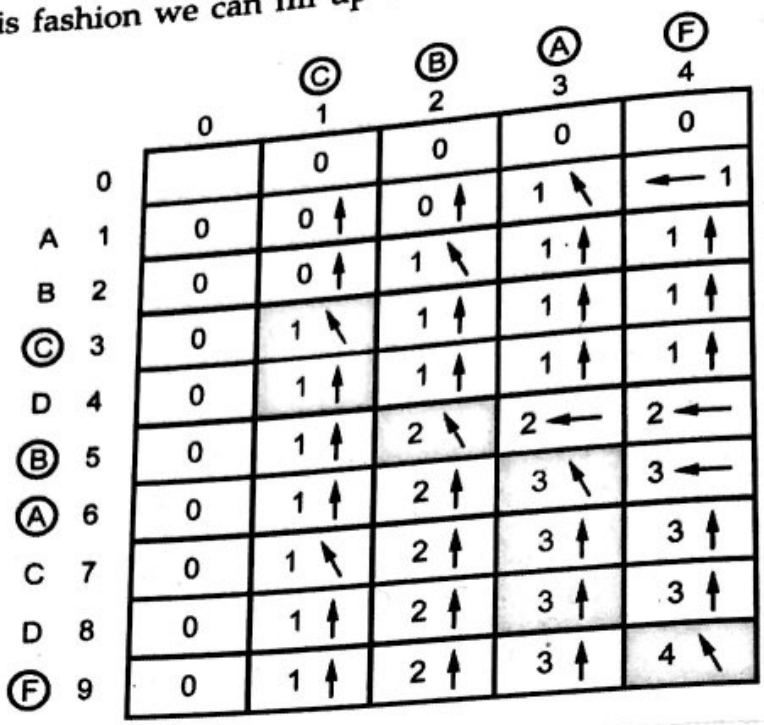
$c[1, 2] = c[0, 2] = 0$

$d[i, j] = \uparrow$

The table will be

	0	1	2	3	4
0		0	0	0	0
1	0	0 ↑	0 ↑		
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				
8	0				
9	0				

Continuing in this fashion we can fill up the table as follows -



			C	B	A	F
		0	1	2	3	4
	0		0	0	0	0
A	1	0	0 ↑	0 ↑	1 ↖	← 1
B	2	0	0 ↑	1 ↖	1 ↑	1 ↑
C	3	0	1 ↖	1 ↑	1 ↑	1 ↑
D	4	0	1 ↑	1 ↑	1 ↑	1 ↑
B	5	0	1 ↑	2 ↖	2 ←	2 ←
A	6	0	1 ↑	2 ↑	3 ↖	3 ←
C	7	0	1 ↖	2 ↑	3 ↑	3 ↑
D	8	0	1 ↑	2 ↑	3 ↑	3 ↑
F	9	0	1 ↑	2 ↑	3 ↑	4 ↖

Example 4.9.4 *Given two sequences of characters, P = <MLNOM> Q = <MNOM>, Obtain the longest common subsequence.* **GTU : Summer-13, 14, Marks 7**

Solution :

Let

	1	2	3	4	5
P[i]	M	L	N	O	M
Q[i]	M	N	O	M	

We will build the c [i,j] and d [i, j] table as follows

			M	N	O	M
		0	1	2	3	4
	0		0	0	0	0
M	1	0	↖1	←1	←1	↖1
L	2	0	↑1	↑1	↑1	↑1
N	3	0	↑1	↖2	←2	←2
O	4	0	↑1	↑2	↖3	←3
M	5	0	↖1	↑2	↑3	↖4