M.Sc C.S - I
SEM I
E-Journal

Roll No.	027
Name	OJHA ABHISHEK DEVMANI
Subject	DESIGN ANALYSIS AND ALGORITHM

Thakur Educational Trust's (Regd.)
Thakur College of Science & Commerce
UGC Recognised • Affiliated to University Of Mumbai
(NAAC Accredited with Grade "A" [3rd Cycle] & ISO 9001:2015 Certified)

CERTIFICATE

This is here to certify that Mr. **OJHA ABHISHEK DEVMANI**, Seat Number **027** of M.Sc. I Computer Science, has satisfactorily completed the required number of experiments prescribed by the UNIVERSITY OF MUMBAI during the academic year 2021 – 2022.

Date:

Place: Mumbai

Teacher In-Charge Head of Department

External Examiner

INDEX

Sr. No.	Practical Name	Date
1	Write a program to implement insertion sort and find the running time of the algorithm.	27th Aug 2021
2	**2.A** Write a program to implement merge sot algorithm. Compare the time and memory complexity. **2.B** Difference between Merge Sort and Insertion Sort.	27th Aug 2021
3	Write a program on Strassen's algorithm for matrix multiplication and analyze its complexity.	14th Sep 2021
4	Implement hiring problem and analyze its complexity.	22th Sep 2021
5	Write a program to implement Longest Common Subsequence (LCS) algorithm	24th Sep 2021
6	Write a program to implement Huffman's code algorithm	7th Oct 2021
7	Write a program to implement Kruskal's algorithm.	23rd Oct 2021
8	Write a program to implement Dijkstra's algorithm	23rd Oct 2021
9	Write a program to implement multi threaded computation concepts in the generation of Fibonacci numbers	12th Oct 2021
10	Write a program to implement multi threaded computation concepts in the generation of Fibonacci numbers	30th Nov 2021

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 1

027_Abhishek_Ojha

Experiment No -1 **Date of Experiment : 27 August 2021**

Program : - Write a program to implement Insertion sort and find the running time of the algorithm

Example :-

Input :- A[4, 6, 8, 1, 3, 32, 12]

Algorithm :-

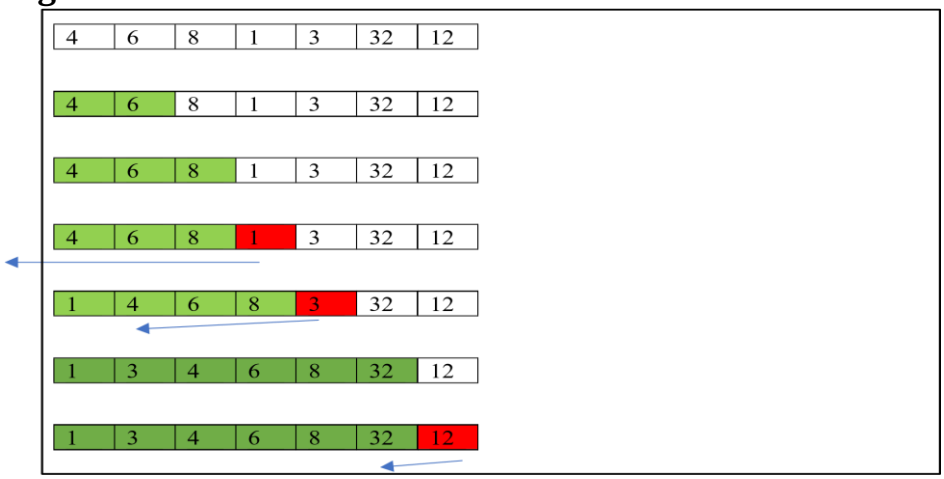
Experiment No:-1 Date of Experiment:-
27th August 2021

Problem:- Write a program to implement Insertion Sort and find the running time of the algorithm

Example:-
A[4, 6, 8, 1, 3, 32, 12]

Algorithm:- For j=2 to A length
key = A[j]
i = j-1
while i > 0 and A[i] > key
A[i+1] = A[i]
i = i-1
A[i+1] = key

Fig :



Program:

```
program :-

import java.io.*;
public class InsertionSort
{
   public static void Sort (int a[])
   {
     int n= a.length, i, j, p, temp;
     for (i=1; i<n; i++)
     {
      for (j=i-1; j>=0 && a[j+1]<a[j]; j--)
      {
       temp=a[j+1];
       a[j+1]= a[j];
        a[j] = temp;
       }
     }
}

public static void printarray(int a[])
{
 for (int i=0; i<a.length; i++)
 {
   System.out.print(a[i]+ " ");
 }
}
public static void main (String [] args) throws
                                        IOException
{
  int n, i;
```

```
BufferedReader b=new BufferedReader(new InputStream
        Reader(System.in));

System.out.println(" Enter Number : ");
 n= Integer.parseInt(b.readline());
 int a[] = new int[n];
System.out.println(" enter "int" elements ");
for (i=0; i<n; i++)

 a[i] = Integer.parseInt(b.readLine());

 System.out.println(" Elements in array ");
 printarray(a);
  Sort(a);
 System.out.println("\n Elements after Sorting ");
 printarray(a);
}
}
```

Conclusion :- Running time of Insertion sort $\Theta(n^2)$

Practical Implementation of Insertion Sort :-

InserionSort.java

C: > Users > aayus > Desktop > DAA Practicals > 027_Abhishek_Ojha > Experiment #01 > InserionSort.java

```java
import java.io.*;
public class InsertionSort
{

public static void Sort(int a[])
{
int n=a.length,i,j,p,temp;
for (i = 1;i < n; i++)
{

for (j=i-1; j >=0 && a[j+1]<a[j]; j--)
{
temp=a[j+1];
a[j+1]=a[j];
a[j]=temp;

}

}
}
public static void printarray(int a[])
{
for(int i=0; i < a.length; i++)
{

System.out.print(a[i]+" ");
}

}
public static void main(String[] args) throws IOException
{

int n,i;
BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
```

C: > Users > aayus > Desktop > DAA Practicals > 027_Abhishek_Ojha > Experiment #01 > InserionSort.java

```java
System.out.println("enter Number: ");
n=Integer.parseInt(b.readLine());
int a[] = new int[n];
System.out.println("enter "+n+" elements ");
for(i= 0; i< n; i++)

a[i] = Integer.parseInt(b.readLine());

System.out.println("elements in array ");
printarray(a);
Sort(a);
System.out.println("\nelements after sorting");
printarray(a);

}
}
```

Output:

```
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #01> javac InsertionSort.java
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #01> java InsertionSort
enter Number:
7
enter 7 elements
4
6
8
1
3
32
12
elements in array
4 6 8 1 3 32 12
elements after sorting
1 3 4 6 8 12 32
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #01>
```

Conclusion: Running time of Insertion Sort $\theta(n \hat{} 2)$

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 2 & 2B

027_Abhishek_Ojha

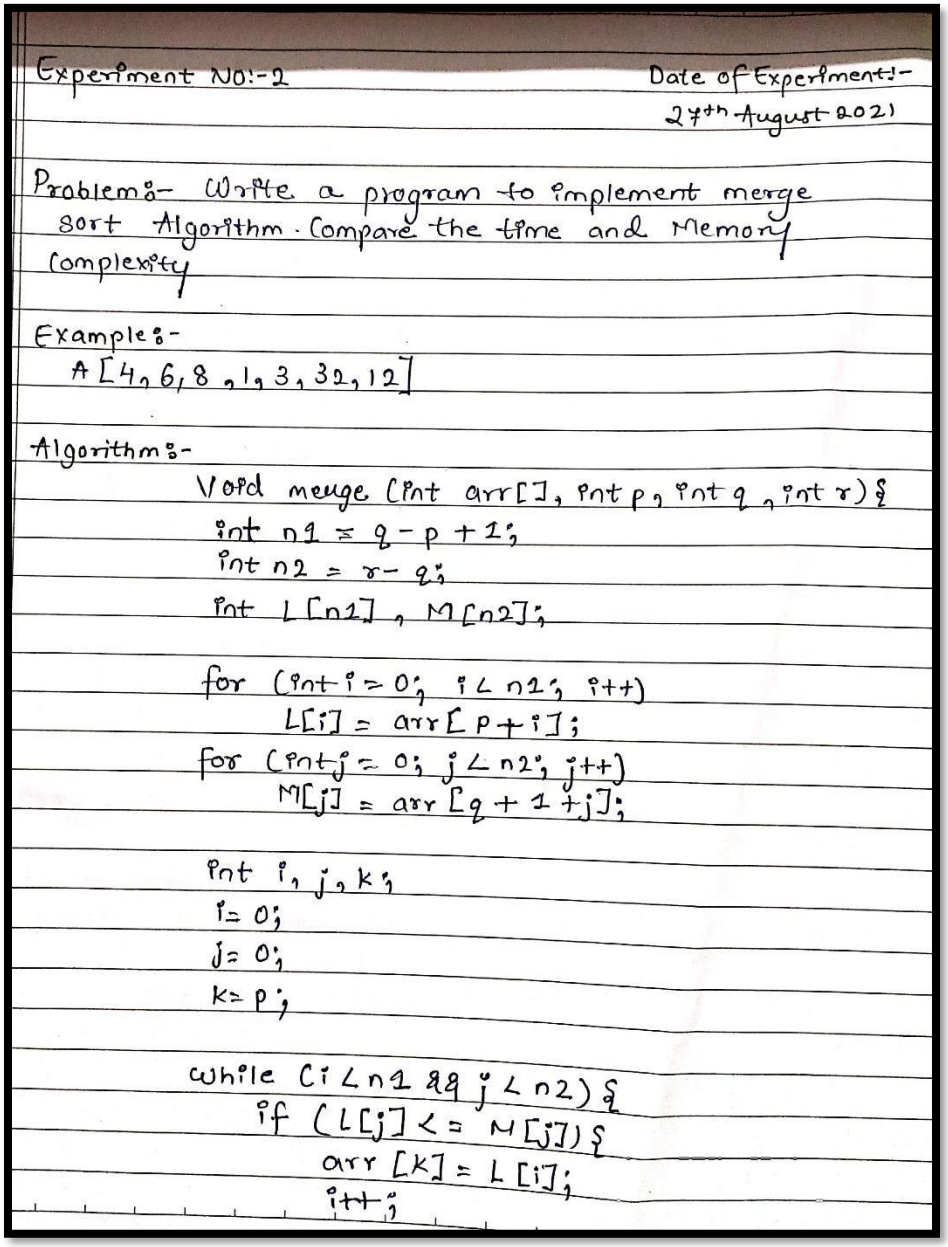
Experiment No -2 **Date of Experiment : 27 August 2021**

Program : - Write a program to implement Merge Sort Algorithm. Compare the time and Memory Complexity

Example :-

Input :- A[4, 6, 8, 1, 3, 32, 12]

Algorithm :-



Experiment No:-2 Date of Experiment:- 27th August 2021

Problem:- Write a program to implement merge sort Algorithm. Compare the time and Memory complexity

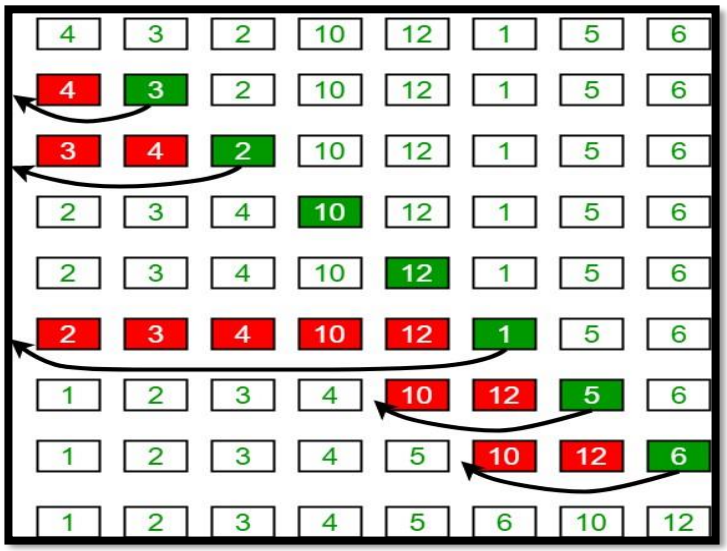
Example:-
A[4, 6, 8, 1, 3, 32, 12]

Algorithm:-

```
Void merge (int arr[], int p, int q, int r) {
 int n1 = q - p + 1;
 int n2 = r - q;
 int L[n1], M[n2];

 for (int i = 0; i < n1; i++)
     L[i] = arr[p + i];
 for (int j = 0; j < n2; j++)
     M[j] = arr[q + 1 + j];

 int i, j, k;
 i = 0;
 j = 0;
 k = p;

 while (i < n1 && j < n2) {
     if (L[j] <= M[j]) {
         arr[k] = L[i];
         i++;
```

```
        }
            else {
                arr[k] = M[j];
                j++;
            }
            k++;
    }
        while (i<n1) {
            arr[k] = L[i];
            i++;
            k++;
    }
        while (j<n2){
            arr[k] = M[j];
            j++;
            k++;
    }
}
```

Fig :



Program:-

```
program :-

import java.io.*;
pubic class Mergesort
{
  public static void merge(int a[], int l, int m, int h)
  {
     int i, j, c=1;
     int b[]=new int [h+1];

  for (i= 1, j= m+1; i<=m && j<=h; c++)
       {
            if (a[i] <= a[j])
            b[c] = a[i++];
            else
            b[c] = a[j++];
       }
   while (i <= m)
            b[c++] = a[i++];
   while (j <= h)
          b[c++] = a[j++];

  for (i=1 ; i<=h; i++)
                a[i] = b[i];

   public static void printarray(int a[])
{
      for (int i=0; i< a.length; i++)
       {
```

```
        System.out.print(a[i]+" ");
      }
}
public static void main(String[] args) throws IOException
{
    int n, i;
    BufferReader b= new BufferedReader (new InputStream
       Reader (System.in));

    System.out.println("Enter Number : ");
    n= Integer.parseInt (b.readLine());
    int a[]= new int[n];
     System.out.println("Enter "+n+" elements ");
    for (i=0; i<n; i++)

     a[i] = Integer.parseInt (b.readLine ());

      System.out.println("Elements in array ");
          printarray(a);
          Sort (a, 0, n-1);
          System.out.println("\n elements after
              sorting");
          printarray(a);
   }
}
```

Merge Sort Complexity

Time Complexity

Best	$O(n * \log n)$
Worst	$O(n * \log n)$
Average	$O(n * \log n)$
Space complexity	$O(n)$

Conclusion :- Merge sort time complexity in Best $O(n * \log n)$, worst $O(n * \log n)$, Average $O(n * \log n)$ and space complexity is $O(n)$

Practical Implementation of Insertion Sort :-

MergeSort.java 1

C: > Users > aayus > Desktop > DAA Practicals > 027_Abhishek_Ojha > Experiment #02 > MergeSort.java

```java
import java.io.*;    MergeSort.java is a non-project file, only syntax erro
public class MergeSort
{

    public static void merge(int a[],int l,int m,int h)
    {
        int i, j,c=l;
        int b[]=new int[h+1];

    for(i = l,j = m+1; i<=m && j<=h; c++)
                {

                    if(a[i] <= a[j])
                    b[c] = a[i++];
                  else
                    b[c] = a[j++];
            }
    while(i <= m )
                    b[c++] = a[i++];

                while(j<=h)
                    b[c++] = a[j++];

      for(i = l ; i <= h; i++)
                            a[i] = b[i];
    }

    public static void Sort(int a[],int l,int h)
    {
          if(l<h)
          {
             int m=(l+h)/2;
             Sort(a,l,m);
             Sort(a,m+1,h);
             merge(a,l,m,h);

          }
```

MergeSort.java 1

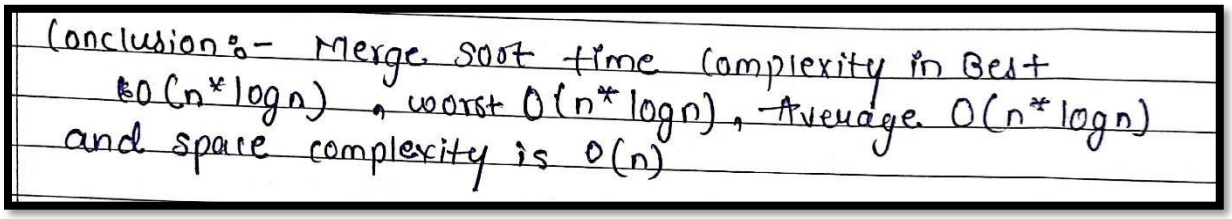
C: > Users > aayus > Desktop > DAA Practicals > 027_Abhishek_Ojha > Experiment #02 > MergeSort.java

```
41
42      public static void printarray(int a[])
43      {
44              for(int i=0; i < a.length; i++)
45               {
46
47               System.out.print(a[i]+"  ");
48               }
49
50      }
        Run | Debug
51      public static void main(String[] args) throws IOException
52      {
53
54          int n,i;
55           BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
56
57
58         System.out.println("enter N: ");
59       n=Integer.parseInt(b.readLine());
60      int a[] = new int[n];
61        System.out.println("enter "+n+" elements ");
62       for(i= 0; i< n; i++)
63
64        a[i] = Integer.parseInt(b.readLine());
65
66        System.out.println("elements in array ");
67              printarray(a);
68              Sort(a,0,n-1);
69               System.out.println("\nelements after sorting");
70               printarray(a);
71
72       }
73      }
```

Output:

```
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #02> javac MergeSort.java
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #02> java  MergeSort
Enter Number:
7
Enter 7 elements
4
6
8
1
3
32
12
Elements in array
4  6  8  1  3  32  12
Elements after sorting
1  3  4  6  8  12  32
PS C:\Users\aayus\Desktop\DAA Practicals\027_Abhishek_Ojha\Experiment #02>
```

Conclusion :


Conclusion :- Merge sort time complexity in Best
O(n*logn), worst O(n*logn), Average O(n*logn)
and space complexity is O(n)

Experiment No -2B **Date of Experiment : 27 August 2021**

Program : Difference between Merge Sort and Insertion Sort

Experiment No:- 2b Date of Experiment 27th August 2021

problem :- Difference between Merge Sort and Insertion Sort.

Merge Sort :- It is external algorithm based on divide and conquer

a. Elements are split into two sub-arrays (n/2) until one element left

b. It uses additional storage to sort auxiliary array.

c. It uses three arrays where two are used to store each half and third external used to store final sorted list

d. All sub array merged to make 'n' element size of array.

Insertion sort:- It is algorithm in which elements are taken from an unsorted item

a. It consist of two loops: an outer loop to pick items and an inner loop to iterate through the array

b. It works on the principle of the sorting playing cards in our hands.

Difference between Merge sort and Insertion sort.

Time Complexity:-
In Merge sort the worst case: $O(N*\log N)$,
Average Case : $O(N*\log N)$,
Best Case: $O(N*\log N)$,

In Insertion sort the worst case: $O(N^2)$,
Average Case: $O(N^2)$,
Best Case: $O(N)$

Space Complexity:-
Merge sort is recursive and takes auxiliary space complexity of $O(N)$

Insertion Sort only takes $O(1)$ auxiliary space complexity.

Datasets:-

Merge sort is prefered for huge data sets
Insertion sort is preferred for fewer elements.

Efficiency:-

Merge sort is efficient in terms of time.
Insertion sort is efficient in terms of space.

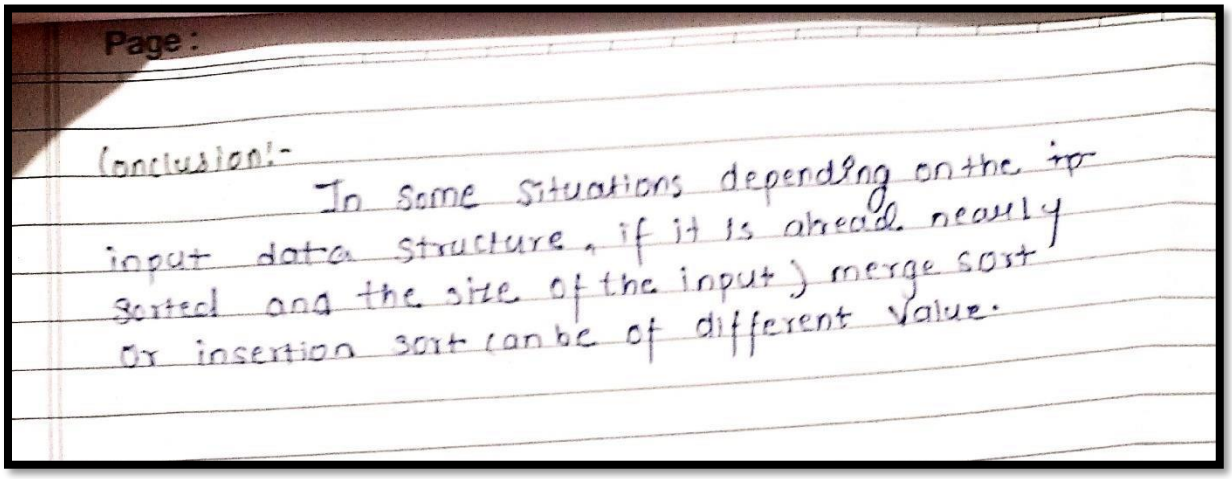
Sorting Method:-

Merge sorting uses external sorting in which data is sorted cannot be accomodate into memory and needed auxiliary memory for sorting

Insertion sorting uses idea that one element from the input elements is consumed in each iteration to finds its correct position.

parameters	Merge Sort	Insertion Sort.
Worst case	O(N * log N)	O(N²)
Average case	O(N * log N)	O(N²)
Best Case Complexity	O(N * log N)	O(N)
Auxiliary Space Complexity	O(N)	O(1)

Balaji


Page :

Conclusion:-

In some situations depending on the ~~ip~~ input data structure, if it is already nearly sorted and the size of the input) merge sort or insertion sort can be of different value.

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 3

027_Abhishek_Ojha

Experiment No – 3 **Date of Experiment : 14th September 2021**

Program : - Write a program on Strassen's algorithm for matrix multiplication and analyze its complexity

Example :-

Matrix Multiplication 3x3

Input :-

Matrix -I			Matrix -II		
2	2	3	1	4	6
1	5	7	2	9	7
8	4	3	5	8	3

Algorithm :-

```
for i = 1 to p do
for j = 1 to r do
        Z[i,j] := 0
          for k = 1 to q do
    Z[i,j] := Z[i,j] + X[i,k] × Y[k,j]
```

Fig :-

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} X \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

$$p1 = a(f - h) \qquad p2 = (a + b)h$$
$$p3 = (c + d)e \qquad p4 = d(g - e)$$
$$p5 = (a + d)(e + h) \qquad p6 = (b - d)(g + h)$$
$$p7 = (a - c)(e + f)$$

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} X \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}$$

A B C

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Program : -

```
/**
 ** Java Program to Implement Strassen Algorithm
 **/

import java.util.Scanner;

/** Class Strassen **/
public class Strassen
{
   /** Function to multiply matrices **/
   public int[][] multiply(int[][] A, int[][] B)
   {
      int n = A.length;
      int[][] R = new int[n][n];
      /** base case **/
      if (n == 1)
         R[0][0] = A[0][0] * B[0][0];
      else
      {
         int[][] A11 = new int[n/2][n/2];
         int[][] A12 = new int[n/2][n/2];
         int[][] A21 = new int[n/2][n/2];
         int[][] A22 = new int[n/2][n/2];
         int[][] B11 = new int[n/2][n/2];
         int[][] B12 = new int[n/2][n/2];
         int[][] B21 = new int[n/2][n/2];
         int[][] B22 = new int[n/2][n/2];

         /** Dividing matrix A into 4 halves **/
         split(A, A11, 0 , 0);
         split(A, A12, 0 , n/2);
         split(A, A21, n/2, 0);
         split(A, A22, n/2, n/2);
         /** Dividing matrix B into 4 halves **/
         split(B, B11, 0 , 0);
         split(B, B12, 0 , n/2);
         split(B, B21, n/2, 0);
```

```
        split(B, B22, n/2, n/2);

        /**
         M1 = (A11 + A22)(B11 + B22)
         M2 = (A21 + A22) B11
         M3 = A11 (B12 - B22)
         M4 = A22 (B21 - B11)
         M5 = (A11 + A12) B22
         M6 = (A21 - A11) (B11 + B12)
         M7 = (A12 - A22) (B21 + B22)
        **/

        int [][] M1 = multiply(add(A11, A22), add(B11, B22));
        int [][] M2 = multiply(add(A21, A22), B11);
        int [][] M3 = multiply(A11, sub(B12, B22));
        int [][] M4 = multiply(A22, sub(B21, B11));
        int [][] M5 = multiply(add(A11, A12), B22);
        int [][] M6 = multiply(sub(A21, A11), add(B11, B12));
        int [][] M7 = multiply(sub(A12, A22), add(B21, B22));

        /**
         C11 = M1 + M4 - M5 + M7
         C12 = M3 + M5
         C21 = M2 + M4
         C22 = M1 - M2 + M3 + M6
        **/
        int [][] C11 = add(sub(add(M1, M4), M5), M7);
        int [][] C12 = add(M3, M5);
        int [][] C21 = add(M2, M4);
        int [][] C22 = add(sub(add(M1, M3), M2), M6);

        /** join 4 halves into one result matrix **/
        join(C11, R, 0 , 0);
        join(C12, R, 0 , n/2);
        join(C21, R, n/2, 0);
        join(C22, R, n/2, n/2);
    }
    /** return result **/
    return R;
}
/** Funtion to sub two matrices **/
```

```
public int[][] sub(int[][] A, int[][] B)
{
    int n = A.length;
    int[][] C = new int[n][n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}
/** Funtion to add two matrices **/
public int[][] add(int[][] A, int[][] B)
{
    int n = A.length;
    int[][] C = new int[n][n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}
/** Funtion to split parent matrix into child matrices **/
public void split(int[][] P, int[][] C, int iB, int jB)
{
    for(int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
        for(int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
            C[i1][j1] = P[i2][j2];
}
/** Funtion to join child matrices intp parent matrix **/
public void join(int[][] C, int[][] P, int iB, int jB)
{
    for(int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
        for(int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
            P[i2][j2] = C[i1][j1];
}
/** Main function **/
public static void main (String[] args)
{
    Scanner scan = new Scanner(System.in);
    System.out.println("Strassen Multiplication Algorithm Test\n");
    /** Make an object of Strassen class **/
    Strassen s = new Strassen();
```

```
        System.out.println("Enter order n :");
        int N = scan.nextInt();
        /** Accept two 2d matrices **/
        System.out.println("Enter N order matrix 1\n");
        int[][] A = new int[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                A[i][j] = scan.nextInt();

        System.out.println("Enter N order matrix 2\n");
        int[][] B = new int[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                B[i][j] = scan.nextInt();

        int[][] C = s.multiply(A, B);

        System.out.println("\nProduct of matrices A and  B : ");
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(C[i][j] +" ");
            System.out.println();
        }

    }
}
```

Practical Implementation of Strassen's Algorithm :-

Strassens.java 1

C: > Users > aayus > Desktop > DAA > Experiment no #3 > 027_Ojha_Abhishek > Strassens.java > Strassens > multiply(int[][], int[][])

```java
import java.util.Scanner;     strassens.java is a non-project file, only syntax errors are reported

public class Strassens{


  private static Scanner scan = new Scanner(System.in);

  public int[][] multiply(int[][] a, int[][] b) {

    int n = a.length;


    int[][] c = new int[n][n];

    if (n == 1)
      c[0][0] = a[0][0] * b[0][0];
    else {
      int[][] A11 = new int[n / 2][n / 2];
      int[][] A12 = new int[n / 2][n / 2];
      int[][] A21 = new int[n / 2][n / 2];
      int[][] A22 = new int[n / 2][n / 2];
      int[][] B11 = new int[n / 2][n / 2];
      int[][] B12 = new int[n / 2][n / 2];
      int[][] B21 = new int[n / 2][n / 2];
      int[][] B22 = new int[n / 2][n / 2];


      split(a, A11, 0, 0);
      split(a, A12, 0, n / 2);
      split(a, A21, n / 2, 0);
      split(a, A22, n / 2, n / 2);

      split(b, B11, 0, 0);
      split(b, B12, 0, n / 2);
      split(b, B21, n / 2, 0);
      split(b, B22, n / 2, n / 2);

```

Strassens.java 1

C: > Users > aayus > Desktop > DAA > Experiment no #3 > 027_Ojha_Abhishek > Strassens.java > Strassens > multip

```
38
39          int[][] p1 = multiply(add(A11, A22), add(B11, B22));
40          int[][] p2 = multiply(add(A21, A22), B11);
41          int[][] p3 = multiply(A11, sub(B12, B22));
42          int[][] p4 = multiply(A22, sub(B21, B11));
43          int[][] p5 = multiply(add(A11, A12), B22);
44          int[][] p6 = multiply(sub(A21, A11), add(B11, B12));
45          int[][] p7 = multiply(sub(A12, A22), add(B21, B22));
46
47          int[][] C11 = add(sub(add(p1, p4), p5), p7);
48          int[][] C12 = add(p3, p5);
49          int[][] C21 = add(p2, p4);
50          int[][] C22 = add(sub(add(p1, p3), p2), p6);
51
52          join(C11, c, 0, 0);
53          join(C12, c, 0, n / 2);
54          join(C21, c, n / 2, 0);
55          join(C22, c, n / 2, n / 2);
56        }
57
58        return c;
59      }
60      public int[][] add(int[][] a, int[][] b) {
61        int n = a.length;
62        int[][] c = new int[n][n];
63        for (int i = 0; i < n; i++)
64          for (int j = 0; j < n; j++)
65            c[i][j] = a[i][j] + b[i][j];
66        return c;
67      }
68
69      public int[][] sub(int[][] a, int[][] b) {
70        int n = a.length;
71        int[][] c = new int[n][n];
72        for (int i = 0; i < n; i++)
73          for (int j = 0; j < n; j++)
74            c[i][j] = a[i][j] - b[i][j];
```

Strassens.java 1

C: > Users > aayus > Desktop > DAA > Experiment no #3 > 027_Ojha_Abhishek > Strassens.java > Strassens > multiply(int[][], int[][])

```
75      return c;
76    }
77
78    public void split(int[][] parentMatrix, int[][] childMatrix,
79                      int fromIndex, int toIndex) {
80      for (int i1=0, i2=fromIndex; i1 < childMatrix.length; i1++, i2++)
81        for (int j1=0, j2=toIndex; j1 < childMatrix.length; j1++, j2++)
82          childMatrix[i1][j1] = parentMatrix[i2][j2];
83    }
84
85    public void join(int[][] childMatrix, int[][] parentMatrix,
86                     int fromIndex, int toIndex) {
87      for (int i1=0, i2=fromIndex; i1 < childMatrix.length; i1++, i2++)
88        for (int j1=0, j2=toIndex; j1 < childMatrix.length; j1++, j2++)
89          parentMatrix[i2][j2] = childMatrix[i1][j1];
90    }
91
92    public int[][] readMatrix(int[][] temp) {
93      for (int i = 0; i < temp.length; i++) {
94        for (int j = 0; j < temp[0].length; j++) {
95          // read matrix elements
96          temp[i][j] = scan.nextInt();
97        }
98      }
99      return temp;
100   }
101
      Run | Debug
102   public static void main(String[] args) {
103
104     System.out.println("Strassen's Matrix "+
105                        "Multiplication\n");
106
107
108     Strassens mtx = new Strassens();
109
110
```

Strassens.java 1

C: > Users > aayus > Desktop > DAA > Experiment no #3 > 027_Ojha_Abhishek > Strassens.java > Strassens > multiply(int[][], int[][])

```
107
108         Strassens mtx = new Strassens();
109
110
111         int size = 0;
112         int a[][] = null; // first matrix
113         int b[][] = null; // second matrix
114         int c[][] = null; // resultant matrix
115
116         System.out.print("Enter Matrix Order: ");
117         size = scan.nextInt();
118
119         a = new int[size][size];
120         b = new int[size][size];
121         c = new int[size][size];
122
123
124         System.out.println("Enter Matrix A: ");
125         a = mtx.readMatrix(a);
126         System.out.println("Enter Matrix B: ");
127         b = mtx.readMatrix(b);
128
129
130         c = mtx.multiply(a, b);
131
132         System.out.println("Resultant Matrix: ");
133         for(int i=0; i<c.length; i++) {
134           for(int j=0; j<c[0].length; j++) {
135             System.out.print(c[i][j]+" ");
136           }
137           System.out.println();
138         }
139       }
140   }
```

Output

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\aayus\Desktop\DAA\Experiment no #3\027_Ojha_Abhishek> javac Strassens.jav
PS C:\Users\aayus\Desktop\DAA\Experiment no #3\027_Ojha_Abhishek> java Strassens
Strassen's Matrix Multiplication

Enter Matrix Order: 3
Enter Matrix A:
2 2 3
1 5 7
8 4 3
Enter Matrix B:
1 4 6
2 9 7
5 8 3
Resultant Matrix:
6 26 0
11 49 0
0 0 0
PS C:\Users\aayus\Desktop\DAA\Experiment no #3\027_Ojha_Abhishek>
```

Analysis :

$$T(n) = \begin{cases} c & if\ n = 1 \\ 7\ x\ T(\frac{n}{2}) + d\ x\ n^2 & otherwise \end{cases}$$ where c and d are constants

Using this recurrence relation, we get $T(n) = O(n^{log7})$

Hence, the complexity of Strassen's matrix multiplication algorithm is $O(n^{log7})$

Conclusion :

integer operations take *O(1)* time. There are three for loops in this algorithm and one is nested in other. Hence, the algorithm takes $O(n^3)$ time to execute.

DATA ANALYSIS AND ALGORITHM

027_ABHISHEK_OJHA

Experiment No – 4 **Date of Experiment : 22th September 2021**

Program : - Program to implementing hiring problem and analyze its complexity.

Input :-

deadlines and

profits JobID

Deadline Profit

a 4

20 b

1 10 c

1 40

d 1

30

Output: Following is

maximum profit

sequence of jobs

c, a

Algorithm :-

Begin

2. Sort all the jobs based on profit Pi so
3. P1 > P2 > P3 ……..>=Pn
4. d = maximum deadline of job in A
5. Create array S[1,……,d]
6. For i=1 to n do
7. Find the largest job x
8. For j=i to 1

9. If ((S[j] = 0) and (x deadline<= d))
10. Then
11. S[x] = i;
12. Break;
13. End if
14. End for
15. End for
16. End

Fig :-

Practical Implementation hiring Problem :-

```
def printJobScheduling(arr, t):

    n = len(arr)

    for i in range(n):         for j in
range(n - 1 - i):              if
arr[j][2] < arr[j + 1][2]:
arr[j], arr[j + 1] = arr[j + 1],
arr[j]

    result = [False] * t

    job = ['-1'] * t

    for i in range(len(arr)):          for j
in range(min(t - 1, arr[i][1] - 1), -1, -
1):
            if result[j]
is False:
result[j] = True
job[j] = arr[i][0]
break     print(job)
arr = [['a', 2, 100],
       ['b', 1, 19],
       ['c', 2, 27],
       ['d', 1, 25],
       ['e', 3, 15]]

print("Following is maximum profit sequence of jobs")

printJobScheduling(arr, 3)
```

Output:

```
PS C:\Users\aayus\Desktop\DAA\Experiment no #4\027_Ojha_Abhishek> & C:/
aayus/Desktop/DAA/Experiment no #4/027_Ojha_Abhishek/hiringproblem.py"
Following is maximum profit sequence of jobs
['c', 'a', 'e']
PS C:\Users\aayus\Desktop\DAA\Experiment no #4\027_Ojha_Abhishek>
```

Time Complexity of the above solution is $O(n^2)$. It can be optimized using Priority Queue(max heap).

Time complexity : O(nlog(n))
Space complexity : O(n)

Analysis :

The optimal sample size and Probability of success for different values of n are : Optimal Sample size k = n / e Probability of success is given by :

$$P(x) = x \int_x^1 \frac{1}{t}\, dt = -x \ln(x)\ .$$

Conclusion:

The Optimal Strategy doesn't always find the best candidate but selects the almost best candidates most of the times

DATA ANALYSIS AND ALGORITHM

PRACTICAL NO 5

027_Ojha Abhishek

Experiment No – 5 **Date of Experiment : 24th September 2021**

Program : - Write a Program to implement Longest Common Subsequence.

Input :-

S1 : ACADB

S2 : CBDA

LCS: CB

Algorithm :-

X and Y be two given sequences

Initialize a table LCS of dimension X.length * Y.length

X.label = X

Y.label = Y

LCS[0][] = 0

LCS[][0] = 0

Start from LCS[1][1]

Compare X[i] and Y[j]

If X[i] = Y[j]

LCS[i][j] = 1 + LCS[i-1, j-1]

Point an arrow to LCS[i][j]

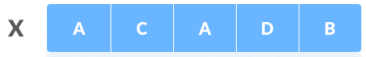
Else

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

Point an arrow to max(LCS[i-1][j], LCS[i][j1])

Fig :-

The First Sequence



The Second Sequence

The following steps are followed for finding the longest common subsequence.

Create a table of dimension `n+1*m+1` where n and m are the lengths of `X` and `Y` respectively.

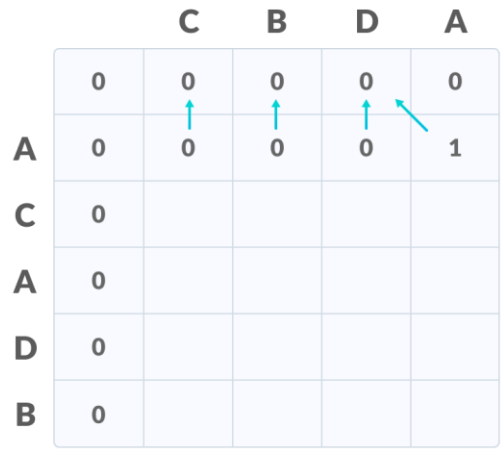
The first row and the first column are filled with zeros.

		C	B	D	A
	0	0	0	0	0
A	0				
C	0				
A	0				
D	0				
B	0				

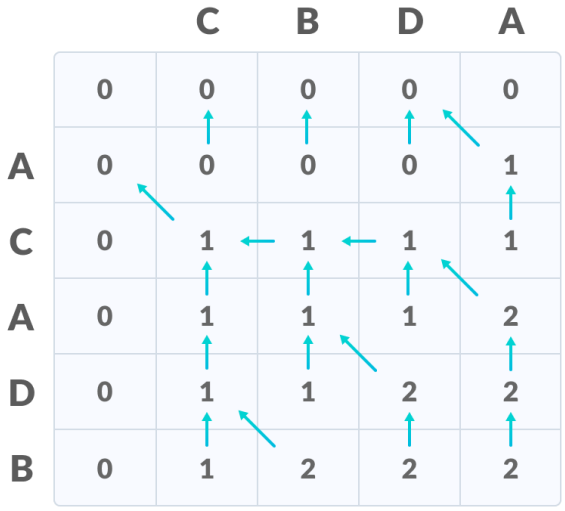
Fill each cell of the table using the following logic.

If the character correspoding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell.

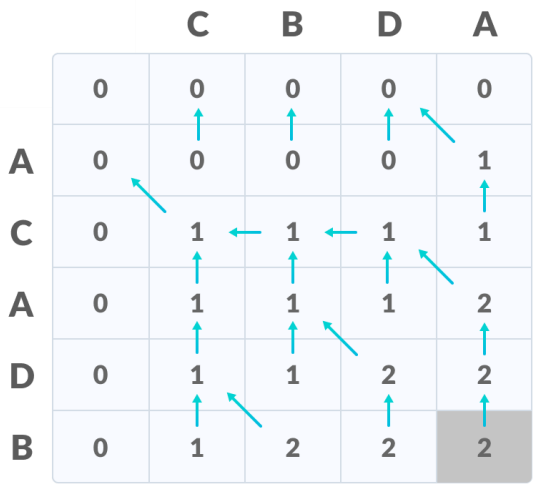
Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them.



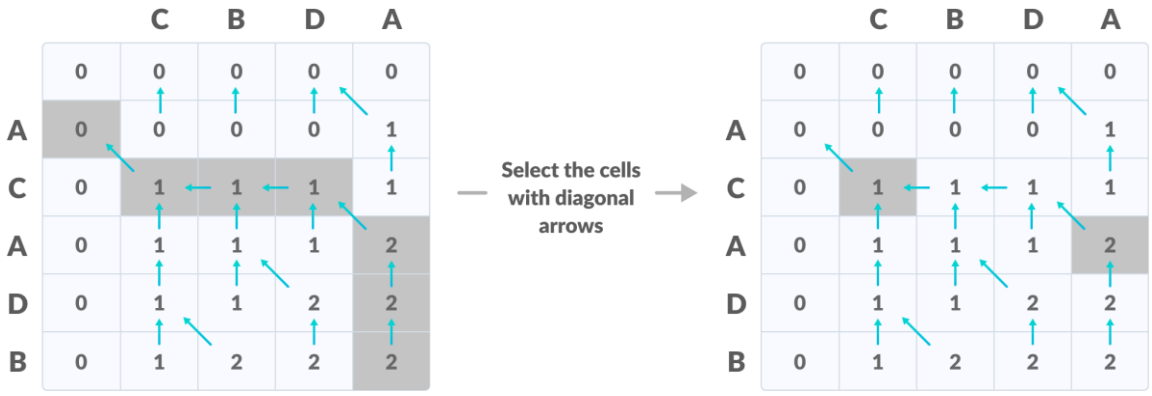
Step 2 **is repeated until the table is filled.**



The value in the last row and the last column is the length of the longest common
subsequence.



In order to find the longest common subsequence, start from the last element and follow the direction of the arrow. The elements corresponding to () symbol form the longest common subsequence.



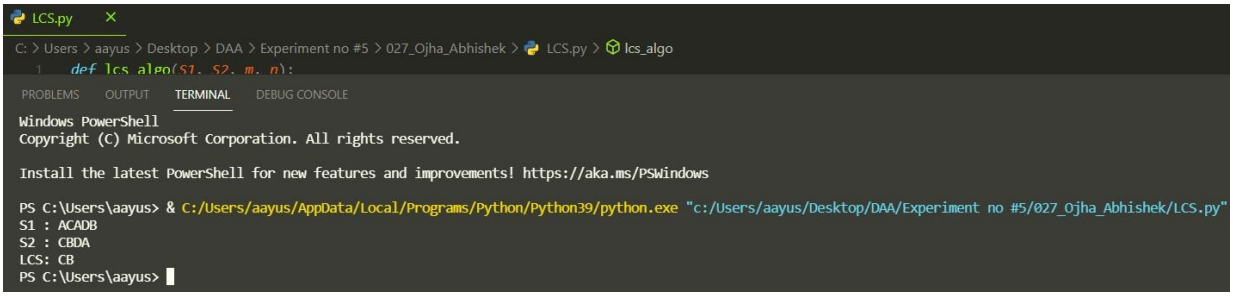
Thus, the longest common subsequence is *CA* .

C	A

Practical Implementation of Longest Common Subsequence :-

```
def lcs_algo(S1, S2, m, n):
   L = [[0 for x in range(n+1)] for x in range(m+1)]
    for i in range(m+1):        for j in
range(n+1):            if i == 0 or j == 0:
L[i][j] = 0            elif S1[i-1] == S2[j-1]:
L[i][j] = L[i-1][j-1] + 1            else:
            L[i][j] = max(L[i-1][j], L[i][j-1])     index =
L[m][n]    lcs_algo = [""] * (index+1)
lcs_algo[index] = ""     i = m     j = n     while i > 0 and
j > 0:
      if S1[i-1] == S2[j-1]:
         lcs_algo[index-1] = S1[i-1]
i -= 1           j -= 1           index -= 1
elif L[i-1][j] > L[i][j-1]:
i       -= 1        else:
j       -= 1
          print("S1 : " + S1 + "\nS2 : " +
S2)    print("LCS: " + "".join(lcs_algo))
S1 = "ACADB" S2 =
"CBDA" m = len(S1) n =
len(S2) lcs_algo(S1,
S2, m, n)
```

Output:



Time complexity of the above naive recursive approach is O(2^n) in worst case and worst case happens when all characters of X and Y mismatch i.e., length of LCS is 0.

Conclusion:

Time Complexity of the above implementation is O(mn) which is much better than the worst-case time complexity.

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 6

027_Abhishek_Ojha

Experiment No - 6 **Date of Experiment :- 7 October 2021**

Program :- Write a program to Implement Huffman's code algorithm.

Input:

A string with different characters, say "ACCEBFFFFAAXXBLKE"

Output:

Code for different characters:

Data: K, Frequency: 1, Code: 0000

Data: L, Frequency: 1, Code: 0001

Data: E, Frequency: 2, Code: 001

Data: F, Frequency: 4, Code: 01

Data: B, Frequency: 2, Code: 100

Data: C, Frequency: 2, Code: 101

Data: X, Frequency: 2, Code: 110 Data: A,

Frequency: 3, Code: 111

Algorithm

huffmanCoding(string)

Input: A string with different characters.

Output: The codes for each individual characters.

Begin

define a node with character, frequency, left and right child of the node for Huffman tree.

create a list 'freq' to store frequency of each character, initially, all are 0

for each character c in the string do

```
            increase the frequency for character ch in freq list.
    done

    for all type of character ch do
        if the frequency of ch is non zero then
```

027_Abhishek_Ojha

```
            add ch and its frequency as a node of priority queue Q.
    done

    while Q is not empty do
            remove item from Q and assign it to left child of node
            remove item from Q and assign to the right child of node
        traverse the node to find the assigned code
    done
End
```

Fig :-

Suppose the string below is to be sent over a network.

Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of 8 * 15 = 120 bits are required to send this string.

Using the Huffman Coding technique, we can compress the string to a smaller size.

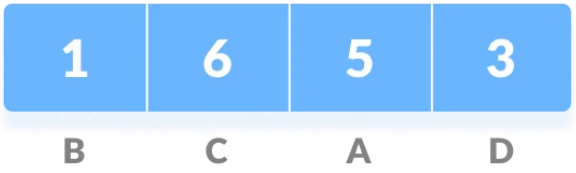
Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

Once the data is encoded, it has to be decoded. Decoding is done using the same tree.

Huffman Coding prevents any ambiguity in the decoding process using the concept of prefix code ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.

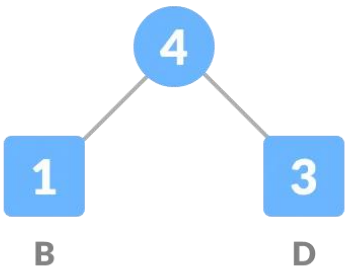
Huffman coding is done with the help of the following steps.

Calculate the frequency of each character in the string.



Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.

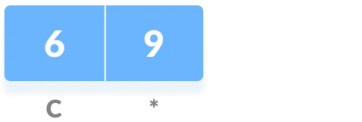
Make each unique character as a leaf node.
Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.

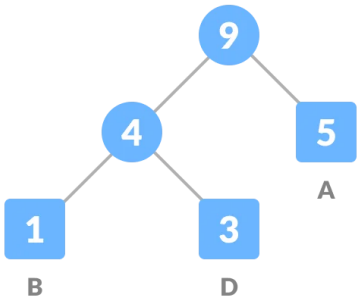


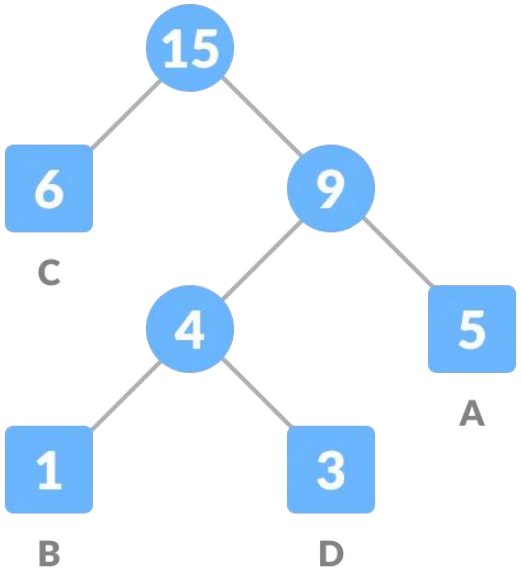
Remove these two minimum frequencies from Q and add the sum into the list of frequencies (* denote the internal nodes in the figure above).

Insert node z into the tree.

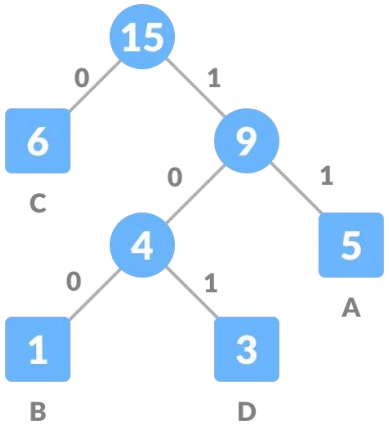
Repeat steps 3 to 5 for all the characters.







For each non-leaf node, assign 0 to the left edge and 1 to the right edge.



For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

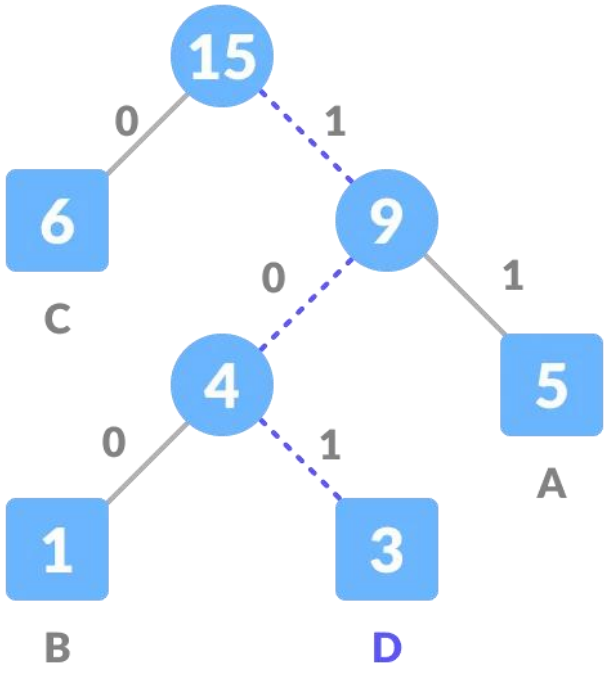
Character	Frequency	Code	Size
A	5	11	5*2 = 10
B	1	100	1*3 = 3
C	6	0	6*1 = 6
D	3	101	3*3 = 9
4 * 8 = 32 bits	15 bits		28 bits

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to 32 + 15 + 28 = 75.

Decoding the code

For decoding the code, we can take the code and traverse through the tree to find the character.

Let 101 is to be decoded, we can traverse from the root as in the figure below.



Practical Implementation of Huffman's code algorithm

```
# A Huffman Tree Node class
node:
        def __init__(self, freq, symbol, left=None, right=None):
            # frequency of symbol
            self.freq = freq

            # symbol name (character) self.symbol =
            symbol

            # node left of current node
            self.left = left

            # node right of current node
            self.right = right

            # tree direction (0/1)
            self.huff = ''

# utility function to print huffman
# codes for all symbols in the newly #
created Huffman tree


def printNodes(node, val=''): # huffman
        code for current node newVal =
        val + str(node.huff)

        # if node is not an edge node # then
        traverse inside it if(node.left):
        printNodes(node.left, newVal)
        if(node.right):
            printNodes(node.right, newVal)

            # if node is edge node then
```

```
            # display its huffman code
        if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")


# characters for huffman tree
chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters
freq = [ 5, 9, 12, 13, 16, 45]

# list containing unused nodes nodes =
[]

# converting characters and frequencies
# into huffman tree nodes for x in
range(len(chars)): nodes.append(node(freq[x],
chars[x]))

while len(nodes) > 1:
        # sort all the nodes in ascending order
        # based on theri frequency nodes =
        sorted(nodes, key=lambda x: x.freq)

        # pick 2 smallest nodes
        left = nodes[0] right =
        nodes[1]

        # assign directional value to these nodes
        left.huff = 0 right.huff = 1

        # combine the 2 smallest nodes to create
        # new node as their parent newNode = node(left.freq+right.freq,
        left.symbol+right.symbol, left,
right)
```

```
        # remove the 2 nodes and add their #
        parent as new node among others
        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

# Huffman Tree is ready! printNodes(nodes[0])
```

Output :

```
f:  0  c:
100    d:
101    a:
1100  b:
1101  e:
111
```

The time complexity for encoding each unique character based on its frequency is O(nlog n).

Extracting minimum frequency from the priority queue takes place 2*(n-1) times and its complexity is O(log n). Thus the overall complexity is O(nlog n).

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 7

027_Abhishek_ Ojha

Experiment No:- 7 **Date of Experiment: - 23 October 2021**

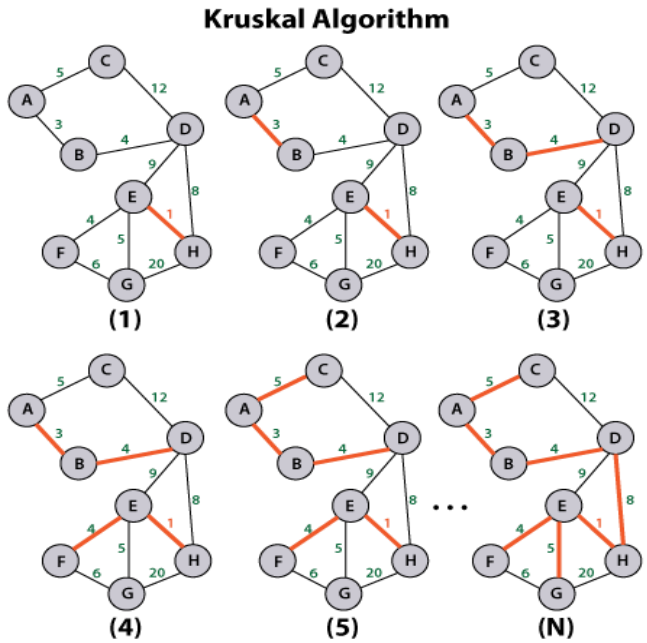
Program: -Write a program to Implement Krushal's Algorithm.

Algorithm

Krushal's Algorithm

1. Begin
2. Create the edge list of given graph, with their weights.
3. Sort the edge list according to their weights in ascending order.
4. Draw all the nodes to create skeleton for spanning tree.
5. Pick up the edge at the top of the edge list (i.e. edge with minimum weight).
6. Remove this edge from the edge list.
7. Connect the vertices in the skeleton with given edge. If by connecting the vertices, a cycle is created in the skeleton, then discard this edge.
8. Repeat steps 5 to 7, until n-1 edges are added or list of edges is over.
9. Return.

Fig:



```
#Practical Implementation Krushal's code algorithm

class Graph:     def __init__(self, vertex):
       self.V = vertex         self.graph = []
     def add_edge(self, u, v, w):
       self.graph.append([u, v, w])
       def search(self, parent, i):
          if parent[i] == i:
          return i
          return self.search(parent, parent[i])
     def apply_union(self, parent, rank, x, y):
       xroot = self.search(parent, x)
       yroot = self.search(parent, y)
        if rank[xroot] < rank[yroot]:
          parent[xroot] = yroot

        elif rank[xroot] > rank[yroot]:
          parent[yroot] = xroot
        else:
          parent[yroot] = xroot
          rank[xroot] += 1
         def kruskal(self):
           result = []
           i, e = 0, 0
           self.graph = sorted(self.graph, key=lambda item: item[2])
            parent = []
            rank = []
            for node in range(self.V):
                parent.append(node)
                rank.append(0)
                while e < self.V - 1:
          u, v, w = self.graph[i]
          i = i + 1
          x = self.search(parent, u)
          y = self.search(parent, v)
          if x != y:
             e = e + 1
             result.append([u, v, w])
             self.apply_union(parent, rank, x, y)
             for u, v, weight in result:
          print("Edge:",u, v,end =" ")
```

```
print("-",weight)
   g = Graph(5)
g.add_edge(0, 1, 8)
g.add_edge(0, 2, 5)
g.add_edge(1, 2, 9)
g.add_edge(1, 3, 11)
g.add_edge(2, 3, 15)
g.add_edge(2, 4, 10)
g.add_edge(3, 4, 7)
g.kruskal()
```

Output:

```
0 - 2: 5
3 - 4: 7
0 - 1: 8
2 - 4: 10
```

The time complexity for Krushal's Algorithm is O(E log V)

Conclusion: Successfully Implemented the Krushal's Algorithm.

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 8

027_Abhishek_Ojha

Experiment No - 8 **Date of Experiment :- 23 October 2021**

Program :- Write a program to Implement Dijkstra's algorithm.

Algorithm

Dijkstra's Algorithm

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going

from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is

infinity.

2. Array visited[ ] is initialized to zero.

for(i=0;i<n;i++)

visited[i]=0;

3. If the vertex 0 is the source vertex then visited[0] is marked as 1.

4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1

from the source vertex 0.

for(i=1;i<n;i++)

distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

- Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark

visited[w] as 1.

- Recalculate the shortest distance of remaining vertices from the source.

- Only, the vertices not marked as 1 in array visited[ ] should be considered for

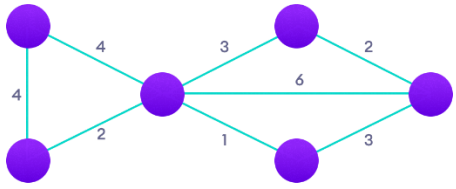
recalculation of distance. i.e. for each vertex v

if(visited[v]==0)

distance[v]=min(distance[v],

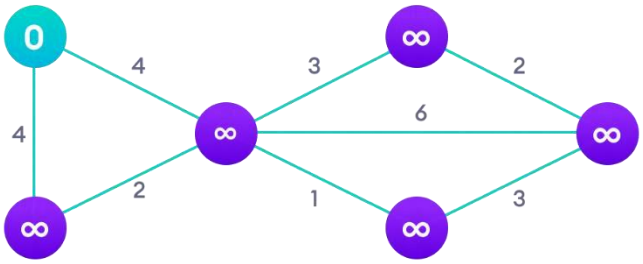
distance[w]+cost[w][v])

Fig:



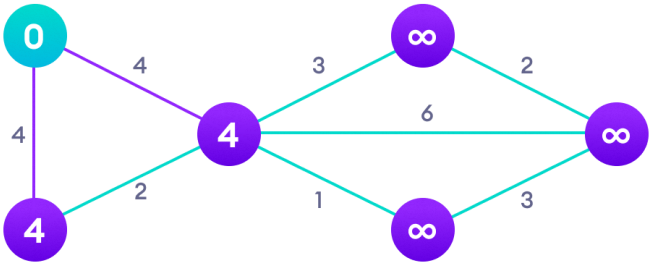
Step: 1

Start with a weighted graph



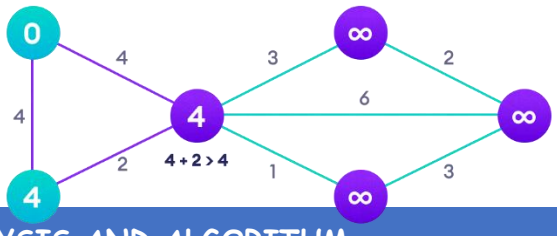
Step: 2

Choose a starting vertex and assign infinity path values to all other devices



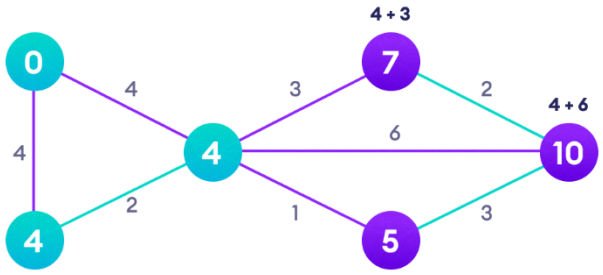
Step: 3

Go to each vertex and update its path length



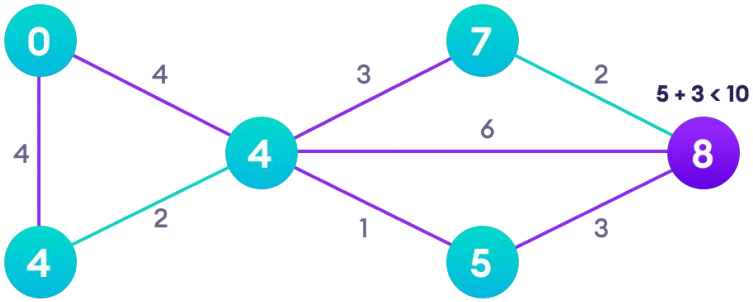
Step: 4

If the path length of the adjacent vertex is lesser than new path length, don't update it



Step: 5

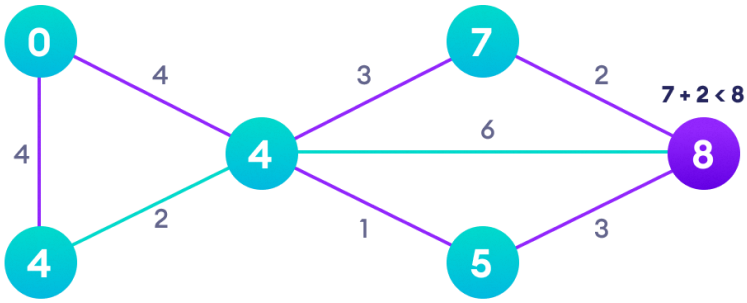
Avoid updating path lengths of already visited vertices



Step: 6

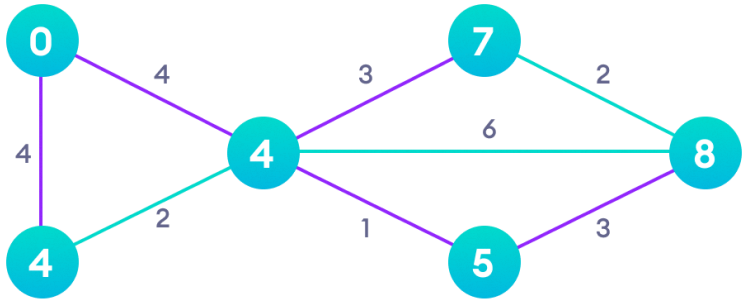
After each iteration, we pick the unvisited vertex with the least path length. So

we choose 5 before 7



Step: 7

Notice how the rightmost vertex has its path length updated twice



Step: 8

Repeat until all the vertices have been visited

Practical Implementation Djikstra's code algorithm

```
import sys

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print "Vertex \tDistance from Source"
        for node in range(self.V):
            print node, "\t", dist[node]


    def minDistance(self, dist, sptSet):

        min = sys.maxint
        for u in range(self.V):
            if dist[u] < min and sptSet[u] == False:
                min = dist[u]
                min_index = u
```

```
        return min_index
    def dijkstra(self, src):

        dist = [sys.maxint] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):

            x = self.minDistance(dist, sptSet)
            sptSet[x] = True
            for y in range(self.V):
                if self.graph[x][y] > 0 and sptSet[y] == False and \
                dist[y] > dist[x] + self.graph[x][y]:
                        dist[y] = dist[x] + self.graph[x][y]

        self.printSolution(dist)

g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
        [4, 0, 8, 0, 0, 0, 0, 11, 0],
        [0, 8, 0, 7, 0, 4, 0, 0, 2],
        [0, 0, 7, 0, 9, 14, 0, 0, 0],
        [0, 0, 0, 9, 0, 10, 0, 0, 0],
        [0, 0, 4, 14, 10, 0, 2, 0, 0],
        [0, 0, 0, 0, 0, 2, 0, 1, 6],
        [8, 11, 0, 0, 0, 0, 1, 0, 7],
        [0, 0, 2, 0, 0, 0, 6, 7, 0]
        ];

g.dijkstra(0);
```

Output:

```
Vertex  Distance from Source
0           0
1           4
2           12
```

```
3       19
4       21
5       11
6       9
7       8
8       14
```

Time Complexity of the implementation is O(V^2). If the input graph is represented using adjacency list, it can be reduced to O(E log V) with the help of a binary heap.

Conclusion: Successfully Implemented the Dijkstra's algorithm.

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 9

027_Abhishek_Ojha

Experiment No - 9 Date of Experiment :- 12 October 2021

Program :- Write a program to implement multi threaded computation concepts in the generation of Fibonacci numbers.

Algorithm

Step 1: Start

Step 2: create class prime implements Runnable

Step 3: create class fib implements Runnable

Step 4: Thread ct=Thread.currentThread()

Step 5: print ct.getName()

Step 6: prime p=new prime()

Step 7: fib f=new fib()

Step 8: Thread fib=new Thread(f,"fibo")

Step 9: Thread prime=new prime(p,"prime")

Step 10: fib.start()

Step 11: print fib.getName()

Step 12: prime.start()

Step 13: print prime.getName()

Step 14: stop

Function public void run() in class prime

Step 1: Start

Step 2: Set i=0

Step 3: Repeat steps 4 to 8 until i<=100 incrementing i by 1

Step 4: Set j=2

Step 5: Repeat steps 5 to 6 until j<=i incrementing j by 1

Step 6: if(i%j==0) go to 6

Step 7: if(i==j) go to 7

Step 8: increment c by 1

Step 9: print the value of i

Step 10: exit

Function public void run() of class fib

Step 1: Start

Step 2: Set n=0,a=0,b=1 and c=0

Step 3: Repeat steps 4 to 7 until n<75 incrementing n by 1

Step 4: print the value of a

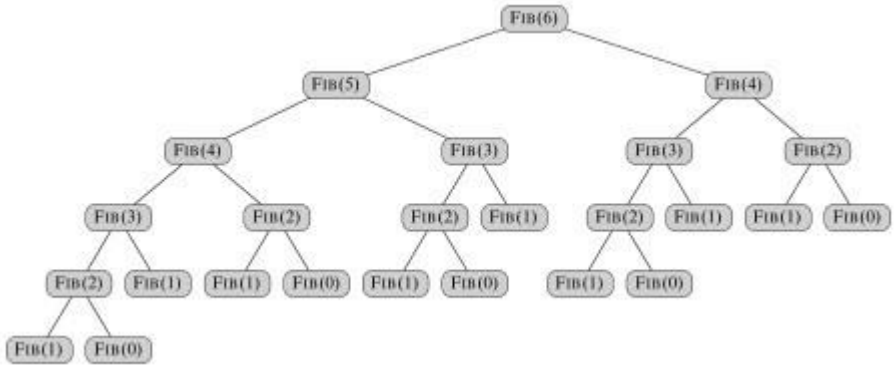
Step 5: Compute c=a+b,assign a=b and b=c

Step 6: if(n==50) go to 7

Step 7: Thread.sleep(500)

Step 8: Exit.

Fig :-



Practical Implementation of Huffman's code algorithm

```
class Prime implements Runnable
{
        long j,c;
        Prime()
        {
```

```
                super(); c=0;
        }
         public void run()
        {
                for(long i=0;i<=100;i++)
                { for(j=2;j<=i;j++)
                        {
                                if(i%j==0) break;
                        } if(j==i)
                        {
                                c++;
                                        System.out.println(c+"th" +" Prime no: = "+i);
                        }
                }
        }
}

class Fib implements Runnable
{
        long a,b,c,n;
        Fib()
        {
                a=c=n=0; b=1;
        }
        public void run()
        {
                while(n++<75)
                {
                        System.out.println(n+"th" +" Fib no: = "+a);
                        c=a+b; a=b; b=c; try
                        {
                                if(n==50)
                                {
System.out.println("Thread fibonacci is put into sleep.");
                                        Thread.sleep(500);
```

```
                        }

                    }
                    catch(InterruptedException e)
                    {
                        System.out.println("Error : " + e);
                    }
                }
            }
        }

public class MyPriFib {

        public static void main(String[] args) {
            Thread ct=Thread.currentThread();
            System.out.println("Main thread name : "+ct.getName());
            Prime p=new Prime();
            Fib f=new Fib();
            Thread fib=new Thread(f,"fibonacci"); Thread
            prime=new Thread(p,"prime"); fib.start();
            System.out.println("Thread "+ fib.getName() + " started.");
            prime.start();
            System.out.println("Thread "+ prime.getName() + " started.");

        }
}
```

Output :

Main thread name : main Thread fibonacci started.

1th Fib no: = 0

2th Fib no: = 1 Thread prime

started.

3th Fib no: = 1

4th Fib no: = 2

5th Fib no: = 3

1th Prime no: = 2

6th Fib no: = 5

2th Prime no: = 3

7th Fib no: = 8

3th Prime no: = 5

8th Fib no: = 13

4th Prime no: = 7

9th Fib no: = 21

5th Prime no: = 11

10th Fib no: = 34

6th Prime no: = 13

11th Fib no: = 55

7th Prime no: = 17

12th Fib no: = 89

8th Prime no: = 19

13th Fib no: = 144

9th Prime no: = 23

14th Fib no: = 233

10th Prime no: = 29

15th Fib no: = 377

11th Prime no: = 31

16th Fib no: = 610

12th Prime no: = 37

17th Fib no: = 987

13th Prime no: = 41

18th Fib no: = 1597

14th Prime no: = 43

19th Fib no: = 2584

15th Prime no: = 47

20th Fib no: = 4181

16th Prime no: = 53

21th Fib no: = 6765

17th Prime no: = 59

22th Fib no: = 10946

18th Prime no: = 61

23th Fib no: = 17711

19th Prime no: = 67

24th Fib no: = 28657

20th Prime no: = 71

25th Fib no: = 46368

21th Prime no: = 73

26th Fib no: = 75025

22th Prime no: = 79

27th Fib no: = 121393

23th Prime no: = 83

28th Fib no: = 196418

24th Prime no: = 89

29th Fib no: = 317811

25th Prime no: = 97 30th Fib no: = 514229

31th Fib no: = 832040

32th Fib no: = 1346269

33th Fib no: = 2178309

34th Fib no: = 3524578

35th Fib no: = 5702887

36th Fib no: = 9227465

37th Fib no: = 14930352

38th Fib no: = 24157817

39th Fib no: = 39088169

40th Fib no: = 63245986

41th Fib no: = 102334155

42th Fib no: = 165580141

43th Fib no: = 267914296

44th Fib no: = 433494437

45th Fib no: = 701408733

46th Fib no: = 1134903170

47th Fib no: = 1836311903

48th Fib no: = 2971215073

49th Fib no: = 4807526976

50th Fib no: = 7778742049 Thread fibonacci is

put into sleep.

51th Fib no: = 12586269025

52th Fib no: = 20365011074

53th Fib no: = 32951280099

54th Fib no: = 53316291173

55th Fib no: = 86267571272

56th Fib no: = 139583862445

57th Fib no: = 225851433717

58th Fib no: = 365435296162

59th Fib no: = 591286729879

60th Fib no: = 956722026041

61th Fib no: = 1548008755920

62th Fib no: = 2504730781961

63th Fib no: = 4052739537881

64th Fib no: = 6557470319842

65th Fib no: = 10610209857723

66th Fib no: = 17167680177565

67th Fib no: = 27777890035288

68th Fib no: = 44945570212853

69th Fib no: = 72723460248141

70th Fib no: = 117669030460994

71th Fib no: = 190392490709135

72th Fib no: = 308061521170129

73th Fib no: = 498454011879264

74th Fib no: = 806515533049393

75th Fib no: = 1304969544928657

DESIGN ANALYSIS AND ALGORITHM

PRACTICAL NO 10

027_Abhishek_Ojha

Experiment No – 10 **Date of Experiment:- 06th December 2021**

Program: Implement Chinese reminder theorem to a constraint satisfaction problem. Analyze its complexity.

Algorithm for Chinese Remainder Theorem

- **Step 1:-** Find M=m, * m, * ... * m. This is the

common modulus.

Step 2:- Find My = M/m, M, = M/m2, ...,Mk =

M/m.

- **Step 3:-** Find the multiplicative inverse of M1,

M2, ...,Mk using the corresponding moduli (m,

m2,

m). Call the inverses as:-

M, ?,M,',..., M.: -1

- **Step 4:** The solution to the simultaneous

equations is:-

x = (a, XM, XM, ' + a2 XM, XM, ' + ... +ak * My x MA-") mod M

Input: num[] = {5, 7}, rem[] = {1, 3}

Output: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get remainder 1.

(2) When we divide it by 7, we get remainder 3.

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}

Output: 11

Explanation:

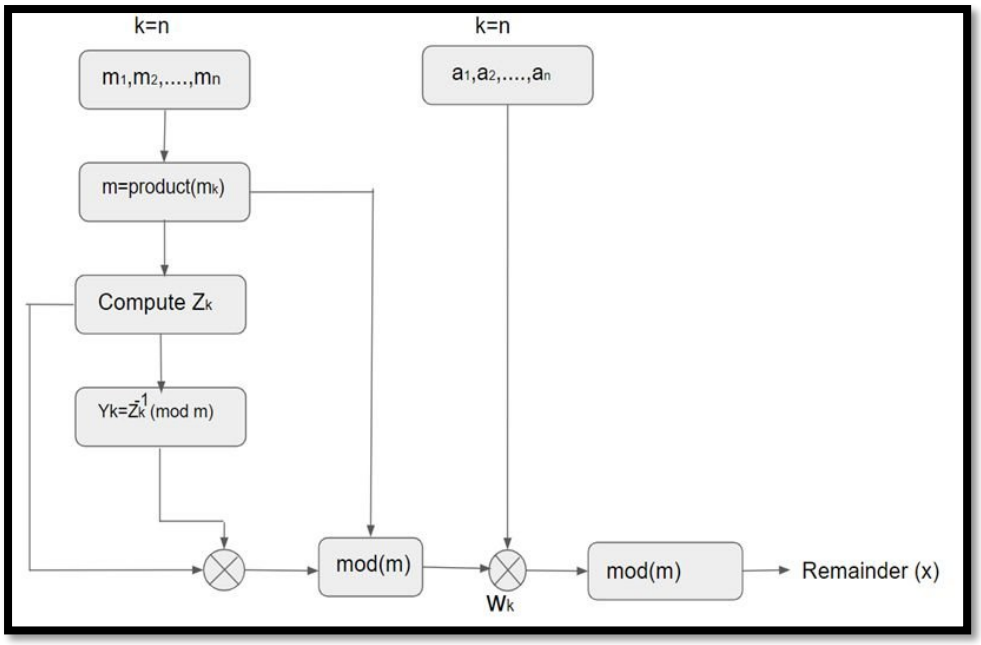
11 is the smallest number such that:

(1) When we divide it by 3, we get remainder 2.

(2) When we divide it by 4, we get remainder 3.

(3) When we divide it by 5, we get remainder 1.

Figure:



Practical Implementation of Chinese Remainder Theorem

```
# Chinese Remainder Theorem
def findMinX(num, rem, k):
    x = 1;
    while(True):

        j = 0;
        while(j < k):
            if (x % num[j] != rem[j]):
                break;
            j += 1;
        if (j == k):
            return x;
        x += 1;
num = [3, 4, 5];
rem = [2, 3, 1];
k = len(num);
print("x is", findMinX(num, rem, k));
```

Output:

```
x is 11
```

Time Complexity: O(M), M is the product of all elements of num[] array.

Auxiliary Space: O(1)

Conclusion: Successfully Implemented the Chinese Remainder Theorem.