**4.QuickSort**

import java.util.Scanner;

class quickSort {

static void quicksort(int arr[],int low,int high){

    if(low<high){

        int pi=partition(arr,low,high);

        quicksort(arr,low,pi-1);

        quicksort(arr,pi+1,high);

    }

}

static void swap(int[] arr, int i, int j)

{

    int temp = arr[i];

    arr[i] = arr[j];

    arr[j] = temp;

}

static int partition(int[] arr, int low, int high)

{

    int pivot = arr[high];

    int i = (low - 1);

    for(int j = low; j <= high - 1; j++)

    {

        // If current element is smaller

        // than the pivot

        if (arr[j] < pivot)

        {

            // Increment index of

            // smaller element

            i++;

            swap(arr, i, j);

        }

    }

    swap(arr, i + 1, high);

    return (i + 1);

}

    static void printArray(int arr[]){

        for(int i:arr){

            System.out.println(i);

        }

    }

    public static void main(String args[]){

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the number of element:");

        int n=sc.nextInt();

        int arr[] = new int[n];

        System.out.println("Enter the element:");

        for(int i=0;i<arr.length;i++){

            arr[i]=sc.nextInt();

        }

        System.out.println("Array before sorting");

        printArray(arr);

        int high=n-1;

        int low=0;

        quicksort(arr,low,high);

        System.out.println("Array after sorting");

        printArray(arr);

    }

}

5.Strassion Matrix

import java.util.Scanner;

class strasion {

    // Function to multiply matrices

    public int[][] multiply(int[][] A, int[][] B)

    {

        // Order of matrix

        int n = A.length;

        // Creating a 2D square matrix with size n

        // n is input from the user

        int[][] R = new int[n][n];

        // Base case

        // If there is only single element

        if (n == 1)

            // Returning the simple multiplication of

            // two elements in matrices

            R[0][0] = A[0][0] \* B[0][0];

        // Matrix

        else {

            //  Dividing Matrix into parts

            // by storing sub-parts to variables

            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);

            // M1:=(A1+A3)×(B1+B2)

            int[][] M1

                = multiply(add(A11, A22), add(B11, B22));

            // M2:=(A2+A4)×(B3+B4)

            int[][] M2 = multiply(add(A21, A22), B11);

            // M3:=(A1−A4)×(B1+A4)

            int[][] M3 = multiply(A11, sub(B12, B22));

            // M4:=A1×(B2−B4)

            int[][] M4 = multiply(A22, sub(B21, B11));

            // M5:=(A3+A4)×(B1)

            int[][] M5 = multiply(add(A11, A12), B22);

            // M6:=(A1+A2)×(B4)

            int[][] M6

                = multiply(sub(A21, A11), add(B11, B12));

            // M7:=A4×(B3−B1)

            int[][] M7

                = multiply(sub(A12, A22), add(B21, B22));

            // P:=M2+M3−M6−M7

            int[][] C11 = add(sub(add(M1, M4), M5), M7);

            // Q:=M4+M6

            int[][] C12 = add(M3, M5);

            // R:=M5+M7

            int[][] C21 = add(M2, M4);

            // S:=M1−M3−M4−M5

            int[][] C22 = add(sub(add(M1, M3), M2), M6);

            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 R;

    }

    public int[][] sub(int[][] A, int[][] B)

    {

        //

        int n = A.length;

        //

        int[][] C = new int[n][n];

        // Iterating over elements of 2D matrix

        // using nested for loops

        // Outer loop for rows

        for (int i = 0; i < n; i++)

            // Inner loop for columns

            for (int j = 0; j < n; j++)

                // Subtracting corresponding elements

                // from matrices

                C[i][j] = A[i][j] - B[i][j];

        // Returning the resultant matrix

        return C;

    }

    // Function to add two matrices

    public int[][] add(int[][] A, int[][] B)

    {

        //

        int n = A.length;

        // Creating a 2D square matrix

        int[][] C = new int[n][n];

        // Iterating over elements of 2D matrix

        // using nested for loops

        // Outer loop for rows

        for (int i = 0; i < n; i++)

            // Inner loop for columns

            for (int j = 0; j < n; j++)

                // Adding corresponding elements

                // of matrices

                C[i][j] = A[i][j] + B[i][j];

        // Returning the resultant matrix

        return C;

    }

    // Function to split parent matrix

    // into child matrices

    public void split(int[][] P, int[][] C, int iB, int jB)

    {

        // Iterating over elements of 2D matrix

        // using nested for loops

        // Outer loop for rows

        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)

            // Inner loop for columns

            for (int j1 = 0, j2 = jB; j1 < C.length;

                j1++, j2++)

                C[i1][j1] = P[i2][j2];

    }

    // Function to join child matrices

    // into (to) parent matrix

    public void join(int[][] C, int[][] P, int iB, int jB)

    {

        // Iterating over elements of 2D matrix

        // using nested for loops

        // Outer loop for rows

        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)

            // Inner loop for columns

            for (int j1 = 0, j2 = jB; j1 < C.length;

                j1++, j2++)

                P[i2][j2] = C[i1][j1];

    }

    // Main driver method

    public static void main(String[] args)

    {

        // Display message

        System.out.println(

            "Strassen Multiplication Algorithm Implementation For Matrix Multiplication :\n");

        // Create an object of Strassen class

        // in he main function

        strasion s = new strasion();

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the size of matrix:");

        // Size of matrix

        // Considering size as 4 in order to illustrate

        int N =sc.nextInt();

        // Matrix A

        // Custom input to matrix

        System.out.println("Enter the element in matrix A:");

        int[][] A = new int[N][N];

        for(int i=0;i<N;i++){

            for(int j=0;j<N;j++){

                A[i][j]=sc.nextInt();

            }

        }

        System.out.println("Enter the element in matrix B:");

        int[][] B = new int[N][N];

        for(int i=0;i<N;i++){

            for(int j=0;j<N;j++){

                B[i][j]=sc.nextInt();

            }

        }

        // Matrix C computations

        // Matrix C calling method to get Result

        int[][] C = s.multiply(A, B);

        // Display message

        System.out.println(

            "\nProduct of matrices A and B : ");

        for (int i = 0; i < N; i++) {

            // Inner loop for columns

            for (int j = 0; j < N; j++)

// Printing elements of resultant matrix with whitespaces in between

                System.out.print(C[i][j] + " ");

// New line once the all elements

            // are printed for specific row

            System.out.println();

        }

    }

}