1. Ques

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickS(int arr[], int low, int high) {

if (low < high) {

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

quickS(arr, low, pi - 1);

quickS(arr, pi + 1, high);

}

}

void printArr(int arr[], int size) {

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

int main() {

int n;

printf(" number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

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

scanf("%d", &arr[i]);

printf("Unsorted array: ");

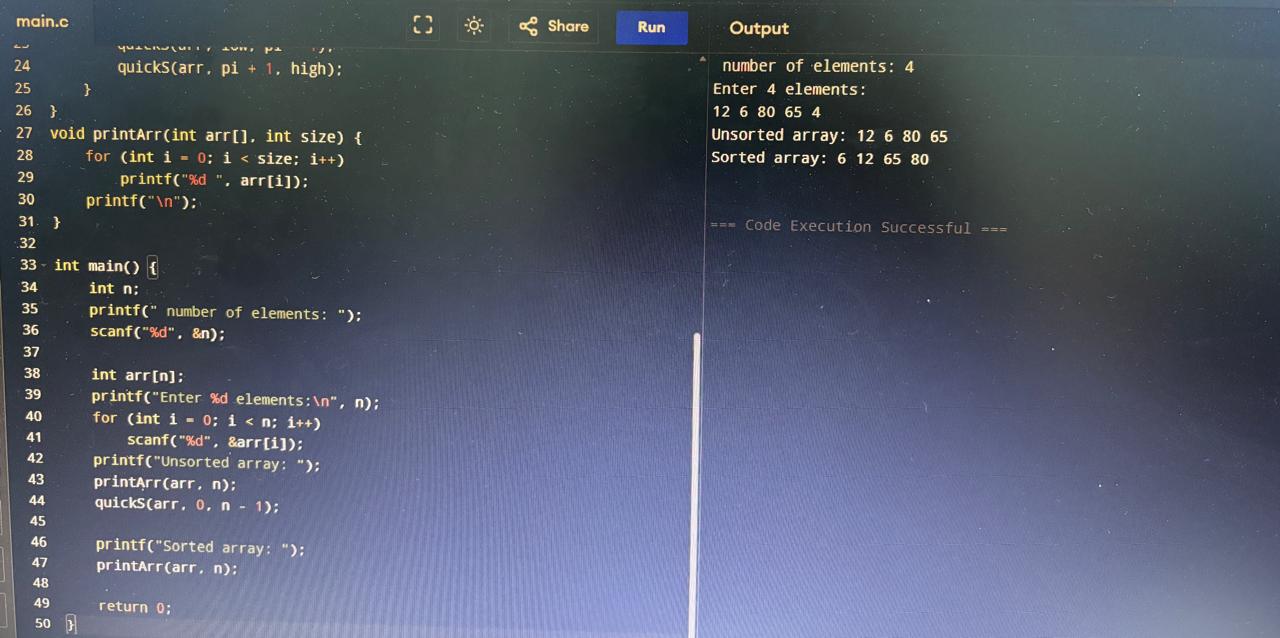
printArr(arr, n);

quickS(arr, 0, n - 1);

printf("Sorted array: ");

printArr(arr, n);

return 0;

}

1. Ques

#include <stdio.h>

void add(int A[4][4], int B[4][4], int C[4][4], int n) {

for (int i=0;i<n;i++) for (int j=0;j<n;j++) C[i][j] = A[i][j] + B[i][j];

}

void sub(int A[4][4], int B[4][4], int C[4][4], int n) {

for (int i=0;i<n;i++) for (int j=0;j<n;j++) C[i][j] = A[i][j] - B[i][j];

}

void strassen2(int A[2][2], int B[2][2], int C[2][2]) {

int M1 = (A[0][0]+A[1][1])\*(B[0][0]+B[1][1]);

int M2 = (A[1][0]+A[1][1])\*B[0][0];

int M3 = A[0][0]\*(B[0][1]-B[1][1]);

int M4 = A[1][1]\*(B[1][0]-B[0][0]);

int M5 = (A[0][0]+A[0][1])\*B[1][1];

int M6 = (A[1][0]-A[0][0])\*(B[0][0]+B[0][1]);

int M7 = (A[0][1]-A[1][1])\*(B[1][0]+B[1][1]);

C[0][0] = M1+M4-M5+M7;

C[0][1] = M3+M5;

C[1][0] = M2+M4;

C[1][1] = M1-M2+M3+M6;

}

int main() {

int A[4][4]={{0}}, B[4][4]={{0}}, C[4][4]={{0}};

int n=3;

printf("Enter 3x3 matrix A:\n");

for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&A[i][j]);

printf("Enter 3x3 matrix B:\n");

for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&B[i][j]);

int A11[2][2],A12[2][2],A21[2][2],A22[2][2];

int B11[2][2],B12[2][2],B21[2][2],B22[2][2];

int C11[2][2],C12[2][2],C21[2][2],C22[2][2];

for(int i=0;i<2;i++) for(int j=0;j<2;j++){

A11[i][j]=A[i][j]; B11[i][j]=B[i][j];

A12[i][j]=A[i][j+2]; B12[i][j]=B[i][j+2];

A21[i][j]=A[i+2][j]; B21[i][j]=B[i+2][j];

A22[i][j]=A[i+2][j+2]; B22[i][j]=B[i+2][j+2];

}

strassen2(A11,B11,C11);

strassen2(A12,B12,C12);

strassen2(A21,B21,C21);

strassen2(A22,B22,C22);

for(int i=0;i<2;i++) for(int j=0;j<2;j++){

C[i][j]=C11[i][j]; C[i][j+2]=C12[i][j];

C[i+2][j]=C21[i][j]; C[i+2][j+2]=C22[i][j];

}

printf("\nResultant 3x3 Matrix:\n");

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

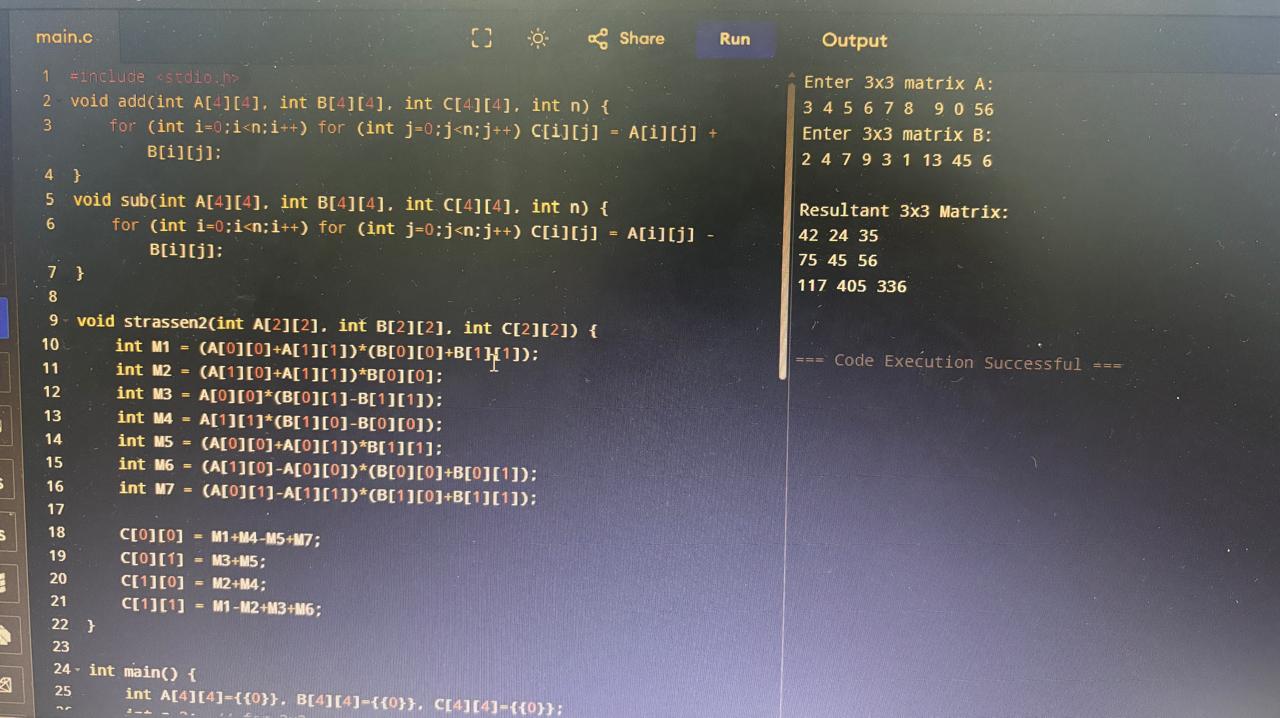
for(int j=0;j<n;j++) printf("%d ",C[i][j]);

printf("\n");

}

return 0;

}



Normal Matrix Multiplication takes O(n3) time (because we compute n2 entries, each at n multiplications).

Strassen's Algorithm reduces the number of multiplications from 8 → 7.

It uses divide and conquer strategy .

Splits each matrix into 4 submatrices of each size n/2 \*n/2 .

This is better for and more faster for larger matrices like if n>=128 .Strassen is efficient in scientific computing, machine learning, and image processing.