**I: Divide and Conquer Strategy Date:**

**Aim:-**Write algorithm and C program to implement the following problems using divide and conquerstrategy

1. Binary search
2. Merge Sort
3. Finding Minimum and maximum element
4. Quick sort
5. Finding Kth smallest element
6. Strassen’s Matrix Multiplication

**THEORY:**

Divide and Conquer is a fundamental algorithmic paradigm used in computer science and mathematics. It is a problem-solving strategy where a problem is divided into smaller subproblems of the same type, solved independently, and then combined to produce a solution to the original problem. This approach is commonly used to solve optimization problems, search problems, and various other computational challenges efficiently.

The Divide and Conquer strategy typically involves three steps:

**Divide**: The original problem is divided into smaller, more manageable subproblems. This division continues recursively until the subproblems become simple enough to be solved directly.

**Conquer**: Each subproblem is solved independently. This step may involve applying the same Divide and Conquer strategy recursively or using other techniques, such as dynamic programming or brute-force methods, depending on the nature of the subproblem.

**Combine**: Finally, the solutions to the subproblems are combined to produce a solution to the original problem. This step typically involves merging or aggregating the solutions obtained from the smaller subproblems.

The key to the effectiveness of Divide and Conquer lies in its ability to break down complex problems into simpler, more manageable parts, which can be solved efficiently. By solving these smaller subproblems independently and then combining their solutions, Divide and Conquer algorithms often achieve significant improvements in terms of time complexity compared to naive or brute-force approaches.

1. **Binary Search**

**Date:**

**Problem Statement:**

Search for ‘R’ and ‘F’ from the following given elements:

[ 'B','E','J','K','N','O','P','R','S','V','W'] using binary search.

**Code**

#include<stdio.h>

#include<stdlib.h>

int sizeOfArr;

void PrintArray(char \*arr, int i, int j)

{

    for(int k =0; k < sizeOfArr; k++)

    {

        if(k == i)

            printf("[%c ",arr[k]);

        else if(k == j)

            printf("%c] ",arr[k]);

        else

            printf(" %c ",arr[k]);

    }

    printf("\n");

}

int BinarySerach(char skey, char \*arr, int i, int j)

{

    int mid = (i + j) / 2;

    PrintArray(arr,i,j);

    if(arr[mid] == skey)

    {

        return mid;

    }

    else if(i == j)

    {

        return -1;

    }

    else if(skey < arr[mid])

    {

        j = mid;

        return BinarySerach(skey,arr,i,j);

    }

    else if(skey >  arr[mid])

    {

        i = mid + 1;

        return BinarySerach(skey,arr,i,j);

    }

}

int main()

{

    char \*arr;

    char skey;

    printf("Enter size of array : ");

    scanf("%d",&sizeOfArr);

    arr = (char\*)malloc(sizeOfArr \* sizeof(char) );

    printf("Enter the elements of the array:\n");

    getchar();

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

    {

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

    }

    printf("Enter element to search : ");

    scanf("%c",&skey);

    int pos = BinarySerach(skey,arr, 0, sizeOfArr-1);

    if(pos != -1)

        printf("%c found at pos %d\n", skey, pos+1);

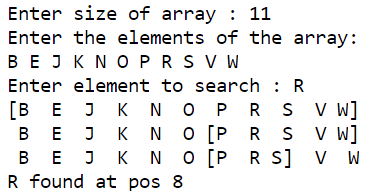
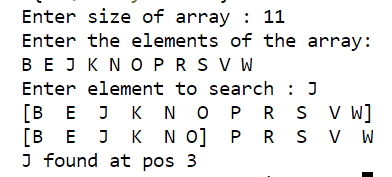
    else

        printf("%c not found in array ", skey);

    return 0;

}

**Output**

** **

1. **Merge Sort**

**Date:**

**Problem Statement:**

Sort the arrays in ascending and descending order respectively

1. J, P, N, S, O, B, V, W, K, E, R

2. 65, 18, 40, 52, 81, 93, 111, 23, 28, 25, 21

**Code : Ascending order**

#include<stdio.h>

void PrintArray(char arr[], int low, int up,int n)

{

    for(int i = 0; i < n; i++)

    {

        if(i >= low && i <= up)

            printf("\033[1m\033[4m%c \033[0m",arr[i]);

        else

            printf("%c ",arr[i]);

        if(i == (low+up)/2)

            printf("|");

    }

    printf("\n");

}

void Merge(char arr[], int low, int up, int mid)

{

    int n1, n2, i, j, k;

    n1 = mid - low + 1;

    n2 = up - mid ;

    char arr1[n1], arr2[n2];

    for(i = 0; i< n1; i++)

        arr1[i] = arr[low+i];

    for(i = 0; i< n2; i++)

        arr2[i] = arr[mid+1+i];

    i = low;

    j = k = 0;

    while(j < n1 && k < n2 && i < up)

    {

        if(arr1[j] <= arr2[k])

            arr[i] = arr1[j++];

        else

            arr[i] = arr2[k++];

        i++;

    }

    while(j < n1 )

        arr[i++] = arr1[j++];

    while(k < n2 )

        arr[i++] = arr2[k++];

}

void MergeSort(char arr[], int low, int up,int n)

{

    int mid = (low + up)/ 2;

    if(low >= up)

        return;

    MergeSort(arr,low,mid,n);

    MergeSort(arr, mid+1, up,n);

    Merge(arr, low, up, mid);

    PrintArray(arr,low,up,n);

}

int main()

{

    char arr[] = { 'J', 'P', 'N', 'S', 'O', 'B', 'V', 'W', 'K', 'E', 'R' };

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given Array : ");

    for(int i = 0; i < arr\_size; i++)

        printf("%c ",arr[i]);

    printf("\n");

    MergeSort(arr,0,arr\_size-1,arr\_size);

    printf("After sorting : ");

    for(int i = 0; i < arr\_size; i++)

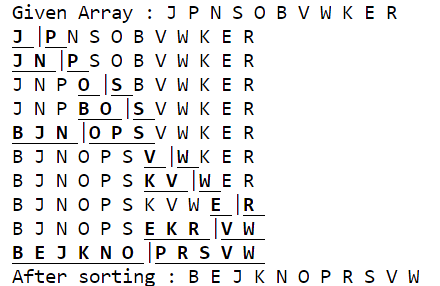
        printf("%c ",arr[i]);

    printf("\n");

    return 0;

}

**Output**

****

**Code : Decending order**

#include<stdio.h>

void PrintArray(int \*arr, int low, int up,int n)

{

    for(int i = 0; i < n; i++)

    {

        if(i >= low && i <= up)

            printf("\033[1m\033[4m%d \033[0m",arr[i]);

        else

            printf("%d ",arr[i]);

        if(i == (low+up)/2)

            printf("|");

    }

    printf("\n");

}

void Merge(int \*arr, int low, int up, int mid)

{

    int n1, n2, i, j, k;

    n1 = mid - low + 1;

    n2 = up - mid ;

    int arr1[n1], arr2[n2];

    for(i = 0; i< n1; i++)

        arr1[i] = arr[low+i];

    for(i = 0; i< n2; i++)

        arr2[i] = arr[mid+1+i];

    i = low;

    j = k = 0;

    while(j < n1 && k < n2 && i < up)

    {

        if(arr1[j] >= arr2[k])

            arr[i] = arr1[j++];

        else

            arr[i] = arr2[k++];

        i++;

    }

    while(j < n1 )

        arr[i++] = arr1[j++];

    while(k < n2 )

        arr[i++] = arr2[k++];

}

void MergeSort(int \*arr, int low, int up,int n)

{

    int mid = (low + up)/ 2;

    if(low >= up)

        return;

    MergeSort(arr,low,mid,n);

    MergeSort(arr, mid+1, up,n);

    Merge(arr, low, up, mid);

    PrintArray(arr,low,up,n);

}

int main()

{

    int arr[] = { 65, 18, 40, 52, 81, 93, 111, 23, 28, 25, 21 };

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given Array : ");

    for(int i = 0; i < arr\_size; i++)

        printf("%d ",arr[i]);

    printf("\n");

    MergeSort(arr,0,arr\_size-1,arr\_size);

    printf("After sorting : ");

    for(int i = 0; i < arr\_size; i++)

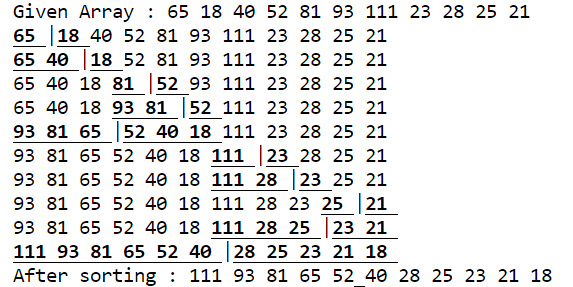
        printf("%d ",arr[i]);

    printf("\n");

    return 0;

}

**Output**

****

1. **Finding maximum and minimum element**

**Date:**

**Problem Statement:**

Find minimum and maximum of

**Code**

#include<stdio.h>

#include<stdlib.h>

struct block

{

    int low, high, min, max,space;

};

struct block \*stack = NULL;

int stackTop = -1;

int space  = -1;

void PrintSearchTree()

{

    for(int i = stackTop; i >= 0; i-- )

    {

        for(int j = 1; j <= stack[i].space; j++){

                printf("     ");

        }

        printf("low : %d, high : %d, min : %d, max : %d\n",stack[i].low,stack[i].high,stack[i].min,stack[i].max);

    }

}

void MinMax(int \*arr, int i, int j, int \*min, int \*max)

{

    int min1, min2, max1, max2, mid;

    space++;

    if(i == j -1)

    {

        if(arr[i] < arr[j])

        {

            \*min = arr[i];

            \*max = arr[j];

        }

        else{

            \*min = arr[j];

            \*max = arr[i];

        }

    }

    else if(i == j)

    {

        \*max = \*min = arr[i];

    }

    else{

        mid = (i+j)/2 ;

        MinMax(arr, i, mid, &min1, &max1);

        MinMax(arr, mid+1, j, &min2, &max2);

        if( max1 < max2)

            \*max = max2;

        else

            \*max = max1;

        if(min1 > min2)

            \*min = min2;

        else

            \*min = min1;

    }

    stack[++stackTop].low = i;

    stack[stackTop].high = j;

    stack[stackTop].min = \*min;

    stack[stackTop].max = \*max;

    stack[stackTop].space = space;

    space--;

}

int main()

{

    int arr[] = { 83,10,77,43,95,-5,12,81,39};

    int min, max;

    int n = sizeof(arr) / sizeof(arr[0]);

    stack = (struct block\*)malloc(sizeof(struct block)\*n);

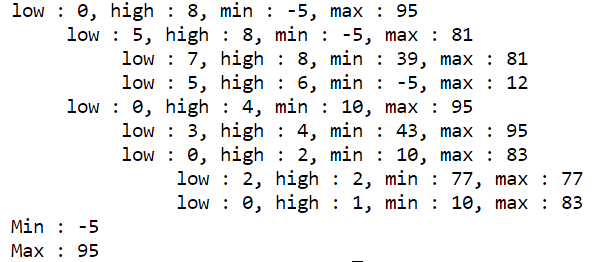
    MinMax(arr,0,n-1,&min,&max);

    PrintSearchTree();

    printf("Min : %d\nMax : %d",min,max);

}

**Output**

****

1. **Quick Sort**

**Date:**

**Problem Statement:**

**Code**

#include <stdio.h>

#include <limits.h>

void PrintArray(int \*arr, int n, int piviot)

{

    for(int i = 0; i < n-1 ; i++){

        if(i == piviot)

            printf("\033[4m\033[1m%d\033[0m  ",arr[i]);

        else

            printf("%d  ",arr[i]);

    }

    printf("\n");

}

int Partition(int \*arr, int p, int q)

{

    int v = arr[p];

    int i = p;

    int j = q;

    do

    {

        do

        {

            i++;

        }while (arr[i] <= v);

        do

        {

            j--;

        }while (arr[j] > v);

        if (i < j)

        {

            int temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    } while (i < j);

    arr[p] = arr[j];

    arr[j] = v;

    return j;

}

void QuickSort(int \*arr, int p, int q, int n)

{

    if (p < q)

    {

        int j = Partition(arr, p, q );

        PrintArray(arr,n,j);

        QuickSort(arr, p, j, n);

        QuickSort(arr, j + 1, q, n);

    }

}

int main()

{

    int arr[] = { 55, 11, 33, 23, -17, 89, -11, 72, 43, INT\_MAX};

    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Array before sorting :");

    for (int i = 0; i < n-1; i++)

        printf("%d  ", arr[i]);

    printf("\n");

    QuickSort(arr, 0, n - 1, n);

    printf("\nArray after sorting : ");

    for (int i = 0; i < n-1; i++)

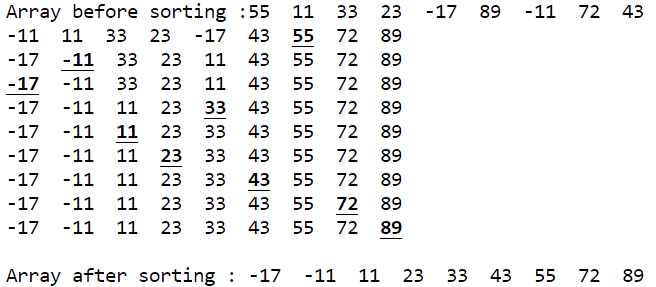
        printf("%d  ", arr[i]);

    printf("\n");

    return 0;

}

**Output**

****

1. **Finding kth smallest / Largest element**

**Date:**

**Problem Statement:**

Find the kth largest / smallest element respectively

Largest : 6th and 10th

Smallest: 4th and 9th

**Code : Kth largest**

#include<stdio.h>

#include<limits.h>

void PrintArray(int \*arr, int low, int up, int n, int j)

{

    for(int i = 0; i < n-1; i++){

        if(i == low)

            printf("[%d ",arr[i]);

        else

            printf("%d ",arr[i]);

        if(i == up -2)

            printf("]");

    }

    printf("Piviot : %d -> %d\n",j,arr[j]);

}

int Partition(int \*arr, int p, int q)

{

    int v = arr[p];

    int i = p;

    int j = q;

    do

    {

        do

        {

            i++;

        }while (arr[i] <= v);

        do

        {

            j--;

        }while (arr[j] > v);

        if (i < j)

        {

            int temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    } while (i < j);

    arr[p] = arr[j];

    arr[j] = v;

    return j;

}

int Select(int \*arr, int n, int k)

{

    int j, low, up;

    if(k < n)

    {

        low = 0;

        up = n;

        do

        {

            j = Partition(arr,low,up);

            PrintArray(arr,low,up,n,j);

            if(k == j)

                return arr[k];

            else if(k < j)

                up = j;

            else

                low = j+1;

        }while(1);

    }

    else{

        printf("\nSearch parameter exceeds maximum value\n");

        return -1;

    }

}

int main()

{

    int arr[] = {63, 18, 23, 9, 79, 89, 54, 69, 12, 72, 41, INT\_MAX};

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    int k;

    printf("Given array : ");

    for(int i = 0; i < arr\_size -1; i++)

        printf("%d  ",arr[i]);

    printf("\n Enter the k for the kth largest element : ");

    scanf("%d",&k);

    int value = Select(arr,arr\_size, arr\_size-k-1);

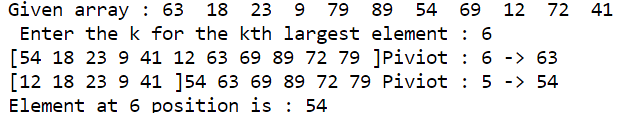
    if(value != -1)

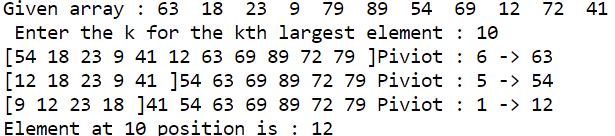
        printf("Element at %d position is : %d\n",k,value);

    return 0;

}

**Output**

****

****

**Code : Kth smallest**

#include<stdio.h>

#include<limits.h>

void PrintArray(char \*arr, int low, int up, int n, int j)

{

    for(int i = 0; i < n-1; i++){

        if(i == low)

            printf("[%c ",arr[i]);

        else

            printf("%c ",arr[i]);

        if(i == up -2)

            printf("]");

    }

    printf("Piviot : %d -> %c\n",j,arr[j]);

}

int Partition(char \*arr, int p, int q)

{

    char v = arr[p];

    int i = p;

    int j = q;

    do

    {

        do

        {

            i++;

        }while (arr[i] <= v);

        do

        {

            j--;

        }while (arr[j] > v);

        if (i < j)

        {

            char temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    } while (i < j);

    arr[p] = arr[j];

    arr[j] = v;

    return j;

}

char Select(char \*arr, int n, int k)

{

    int j, low, up;

    if(k < n)

    {

        low = 0;

        up = n;

        do

        {

            j = Partition(arr,low,up);

            PrintArray(arr,low,up,n,j);

            if(k == j)

                return arr[k];

            else if(k < j)

                up = j;

            else

                low = j+1;

        }while(1);

    }

    else{

        printf("\nSearch parameter exceeds maximum value\n");

        return '0';

    }

}

int main()

{

    char arr[] = { 'T', 'J', 'F', 'R', 'I', 'V','L', 'N', 'Z', 'B', 'U', CHAR\_MAX};

    int arr\_size = sizeof(arr) / sizeof(arr[0]);

    int k;

    printf("Given array : ");

    for(int i = 0; i < arr\_size -1; i++)

        printf("%c  ",arr[i]);

    printf("\n Enter the k for the kth smallest element : ");

    scanf("%d",&k);

    char value = Select(arr,arr\_size, k-1);

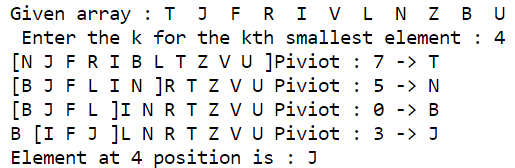
    if(value != '0')

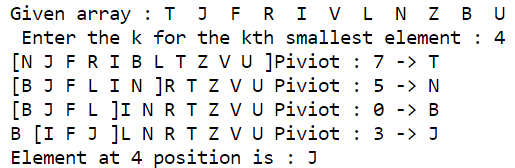
        printf("Element at %d position is : %c\n",k,value);

    return 0;

}

**Output**

****

****

1. **Strassen’s Matrix Multiplication**

**Date:**

**Problem Statement:**

**Code**

#include<stdio.h>

#include<stdlib.h>

int arrSize;

typedef struct Matrix

{

    int \*\*arr;

} matrix;

void InitMatrix(matrix \* arr, int n)

{

    arr->arr = malloc(n\*sizeof(\*arr->arr));

    for(int i=0; i<n; i++){

        arr->arr[i] = malloc(n\*sizeof(\*arr->arr[i]));

    }

}

void PrintMatrix(matrix arr, int row, int col, int Dimension)

{

    for(int i=0; i<Dimension; i++){

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

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

        printf("\n");

    }

}

void Add(matrix A, int row1, int col1, matrix B, int row2, int col2, matrix C, int row3, int col3, int Dimension)

{

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

    {

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

        {

            C.arr[row3 + i][col3 + j] = A.arr[row1 + i][col1 + j] + B.arr[row2 + i][col2 + j];

        }

    }

}

void Subtract(matrix A,int row1, int col1, matrix B, int row2, int col2, matrix C, int Dimension)

{

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

    {

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

        {

            C.arr[i][j] = A.arr[row1 + i][col1 + j] - B.arr[row2 + i][col2 + j];

        }

    }

}

void Copy(matrix A ,int row, int col, matrix B, int Dimension)

{

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

    {

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

        {

            B.arr[i][j] = A.arr[row + i][col + j];

        }

    }

}

void Multiply(matrix A, matrix B, matrix C, int Dimension)

{

    int mid = Dimension/2;

    if(Dimension < 2 && Dimension == 1)

    {

        C.arr[0][0] = A.arr[0][0] \* B.arr[0][0];

        return;

    }

// intialising temprory variables

    matrix temp1, temp2, P, Q, R, S, T, U, V;

    InitMatrix(&temp1,mid);

    InitMatrix(&temp2,mid);

    InitMatrix(&P,mid);

    InitMatrix(&Q,mid);

    InitMatrix(&R,mid);

    InitMatrix(&S,mid);

    InitMatrix(&T,mid);

    InitMatrix(&U,mid);

    InitMatrix(&V,mid);

// finding values of equation by reccuresion

    // P = (A11 + A22)(B11 + B22)

        Add(A,0,0,A,mid,mid,temp1,0,0,mid);

        Add(B,0,0,B,mid,mid,temp2,0,0,mid);

    Multiply(temp1,temp2,P,mid);

    //Q = (A21 + A22)B11

        Add(A,mid,0,A,mid,mid,temp1,0,0,mid);

        Copy(B,0,0,temp2,mid);

    Multiply(temp1,temp2,Q,mid);

    //R = A11(B12 - B22)

        Copy(A,0,0,temp1,mid);

        Subtract(B,0,mid,B,mid,mid,temp2,mid);

    Multiply(temp1,temp2,R,mid);

    //S = A22(B21 - B11)

        Copy(A,mid,mid,temp1,mid);

        Subtract(B,mid,0,B,0,0,temp2,mid);

    Multiply(temp1,temp2,S,mid);

    //T = (A11 + A12)B22

        Add(A,0,0,A,0,mid,temp1,0,0,mid);

        Copy(B,mid,mid,temp2,mid);

    Multiply(temp1,temp2,T,mid);

    //U = (A21 - A11)(B11 + B12)

        Subtract(A,mid,0,A,0,0,temp1,mid);

        Add(B,0,0,B,0,mid,temp2,0,0,mid);

    Multiply(temp1,temp2,U,mid);

    //V = (A12 - A22)(B21 + B22)

        Subtract(A,0,mid,A,mid,mid,temp1,mid);

        Add(B,mid,0,B,mid,mid,temp2,0,0,mid);

    Multiply(temp1,temp2,V,mid);

// adding to final result

    //C11 = P + S - T + V

        Add(P,0,0,V,0,0,temp1,0,0,mid);

        Subtract(S,0,0,T,0,0,temp2,mid);

    Add(temp1,0,0,temp2,0,0,C,0,0,mid);

    //C12 = R + T

    Add(R,0,0,T,0,0,C,0,mid,mid);

    //C21 = Q + S

    Add(Q,0,0,S,0,0,C,mid,0,mid);

    //C22 = P + R - Q + U

        Add(P,0,0,U,0,0,temp1,0,0,mid);

        Subtract(R,0,0,Q,0,0,temp2,mid);

    Add(temp1,0,0,temp2,0,0,C,mid,mid,mid);

//Printing the output of Intermediate metrices

    printf("Intermediate Matrices of %d\*%d matrix\n",Dimension,Dimension);

    printf("P: \n");

    PrintMatrix(P,0,0,mid);

    printf("Q: \n");

    PrintMatrix(Q,0,0,mid);

    printf("R: \n");

    PrintMatrix(R,0,0,mid);

    printf("S: \n");

    PrintMatrix(S,0,0,mid);

    printf("T: \n");

    PrintMatrix(T,0,0,mid);

    printf("U: \n");

    PrintMatrix(U,0,0,mid);

    printf("V: \n");

    PrintMatrix(V,0,0,mid);

    printf("C11: \n");

    PrintMatrix(C,0,0,mid);

    printf("C12: \n");

    PrintMatrix(C,0,mid,mid);

    printf("C21: \n");

    PrintMatrix(C,mid,0,mid);

    printf("C22: \n");

    PrintMatrix(C,mid,mid,mid);

    for(int i=0; i<mid; i++)

    {

        free(temp1.arr[i]);

        free(temp2.arr[i]);

        free(P.arr[i]);

        free(Q.arr[i]);

        free(R.arr[i]);

        free(S.arr[i]);

        free(T.arr[i]);

        free(U.arr[i]);

        free(V.arr[i]);

    }

    free(temp1.arr);

    free(temp2.arr);

    free(P.arr);

    free(Q.arr);

    free(R.arr);

    free(S.arr);

    free(T.arr);

    free(U.arr);

    free(V.arr);

}

int main()

{

    int n;

    printf("Enter size of square matrix : ");

    scanf("%d",&n);

    arrSize = n;

    matrix A, B, C;

    InitMatrix(&A,n);

    InitMatrix(&B,n);

    InitMatrix(&C,n);

    printf("Enter matrix elements of matrix A : \n");

    for(int i=0; i<n; i++)

        for(int j=0; j<n; j++)

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

    printf("Enter matrix elements of matrix B : \n");

    for(int i=0; i<n; i++)

        for(int j=0; j<n; j++)

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

    Multiply(A,B,C,n);

    printf("Matrix Multiplication of A \* B\n");

    for(int i=0; i<n; i++){

        for(int j=0; j<n; j++)

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

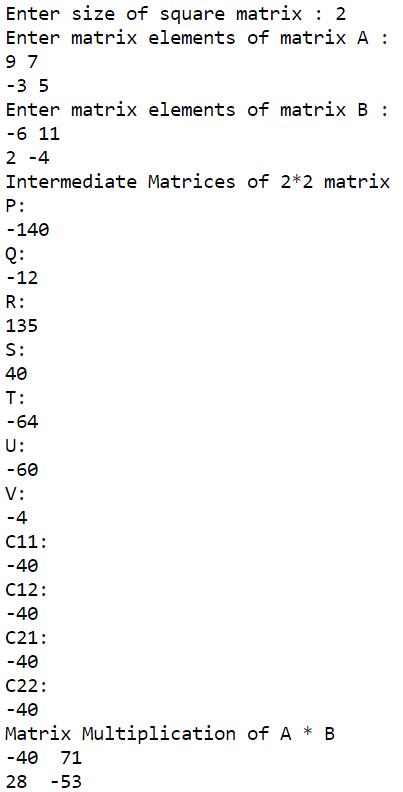
        printf("\n");

    }

    return 0;

}

**Output: 2 \* 2 matrix**

****

**Output: 4 \* 4 matrix**

Enter size of square matrix : 4

Enter matrix elements of matrix A :

3 10 -3 11

-6 4 5 8

-2 7 -12 3

8 -2 10 8

Enter matrix elements of matrix B :

5 3 4 -2

-9 -2 -3 1

8 4 7 -5

11 6 -4 9

Intermediate Matrices of 2\*2 matrix

P:

57

Q:

192

R:

81

S:

-300

T:

28

U:

130

V:

-6

C11:

-277

C12:

-277

C21:

-277

C22:

-277

Intermediate Matrices of 2\*2 matrix

P:

-24

Q:

120

R:

-70

S:

-84

T:

8

U:

256

V:

-44

C11:

-160

C12:

-160

C21:

-160

C22:

-160

Intermediate Matrices of 2\*2 matrix

P:

-77

Q:

6

R:

33

S:

16

T:

-104

U:

0

V:

-42

C11:

1

C12:

1

C21:

1

C22:

1

Intermediate Matrices of 2\*2 matrix

P:

-44

Q:

54

R:

84

S:

136

T:

-72

U:

88

V:

-140

C11:

24

C12:

24

C21:

24

C22:

24

Intermediate Matrices of 2\*2 matrix

P:

192

Q:

77

R:

0

S:

-132

T:

189

U:

-2

V:

45

C11:

-84

C12:

-84

C21:

-84

C22:

-84

Intermediate Matrices of 2\*2 matrix

P:

-88

Q:

72

R:

-10

S:

126

T:

8

U:

190

V:

-39

C11:

-9

C12:

-9

C21:

-9

C22:

-9

Intermediate Matrices of 2\*2 matrix

P:

270

Q:

-75

R:

-144

S:

0

T:

255

U:

-196

V:

176

C11:

191

C12:

191

C21:

191

C22:

191

Intermediate Matrices of 4\*4 matrix

P:

-277 109

-108 76

Q:

-160 -62

36 42

R:

22 -50

S:

24 12

190 74

T:

-84 189

-55 113

U:

-9 -2

198 20

V:

191 111

-75 5

C11:

22 43

62 42

C12:

22 43

62 42

C21:

22 43

62 42

C22:

22 43

62 42

Matrix Multiplication of A \* B

22 43 -83 118

62 42 -33 63

-136 -50 -125 98

226 116 76 4

**CONCLUSION:**

Divide and Conquer strategy was studied. The programs for (a) Binary search (b) Merge sort, (c) Finding minimum and maximum element (d) Quick sort (e) finding kth smallest element and (f) Strassen’s Matrix multiplication algorithms were studied and implemented successfully.