DIGITAL ASSIGNMENT 1 – [DIVIDE AND CONQUER]

CSE2012–DESIGN AND ANALYSIS OF ALGORITHMS (L25-26)[MRS GAYATHRI P]

JANUARY 12, 2023

ANIRUDH VADERA
20BCE2940

ANIRUDH VADERA

DIGITAL ASSIGNMENT 1 – [DIVIDE AND CONQUER]

QUESTION 1:

Design and implement an algorithm using brute force approach that finds the top and the least scores of students from an online Quiz. Take scores as input and store in an array.

PSEUDOCODE:

1. Take the number of students participating in quiz as input from user - n
2. Allocate an array of size using dynamic memory allocation
3. Take the student marks as input from user
4. Initialize the minimum and maximum scores as the arrays first element. We assume that they are min and max scores at first
5. Then loop through the array and at each element check if that element is less than the current minimum element then updates the minimum element to that element similarly check if that element is larger than the current maximum element then updates the maximum element to that element.
6. At last return the minimum and the maximum scores so far stored
7. At every step of the loop, we are doing 2 comparisons in the worst case. Total no. of comparisons (in worst case) = **2*(n-1) = 2n - 2**
8. **Time complexity = O(n), Space complexity = O(1)**

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    printf("Enter the Number of Students Participating in Online Quiz : \n");
    scanf("%d", &n);
    // Allocating a array for student scores
    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter the Students Scores : \n");
    for (int i = 0; i < n; i++)
    {
        printf("Score for Student %d : \n", i + 1);
        scanf("%d", &arr[i]);
    }
    // Initializing max and min by first element of array
    int max = arr[0];
```

```
    int min = arr[0];
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
        if (arr[i] < min)
        {
            min = arr[i];
        }
    }
    printf("The Top Score of the Students in Online Quiz is : %d\n", max);
    printf("The Least Score of the Students in Online Quiz is : %d\n", min);
    return 0;
}
```

OUTPUT SCREENSHOT:

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question1.exe"
Enter the Number of Students Participating in Online Quiz :
10
Enter the Students Scores :
Score for Student 1 :
7
Score for Student 2 :
6
Score for Student 3 :
0
Score for Student 4 :
5
Score for Student 5 :
4
Score for Student 6 :
7
Score for Student 7 :
8
Score for Student 8 :
9
Score for Student 9 :
4
Score for Student 10 :
6
The Top Score of the Students in Online Quiz is : 9
The Least Score of the Students in Online Quiz is : 0
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question1.exe"
Enter the Number of Students Participating in Online Quiz :
5
Enter the Students Scores :
Score for Student 1 :
10
Score for Student 2 :
10
Score for Student 3 :
9
Score for Student 4 :
6
Score for Student 5 :
4
The Top Score of the Students in Online Quiz is : 10
The Least Score of the Students in Online Quiz is : 4
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```

QUESTION 2:

Take ‘n’ historic sites in Tamil Nadu as input. Design and implement an algorithm using brute force approach that identifies the shortest possible route for a traveller to visit these sites and come back to his starting point.

PSEUDOCODE:

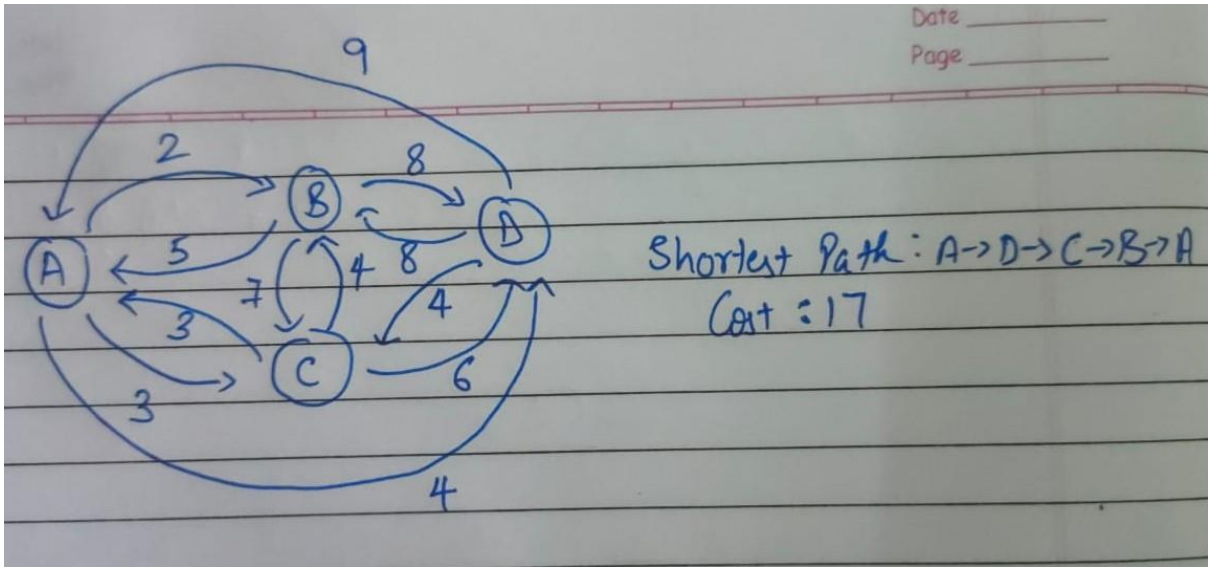
1. Take the number of sites n as input from user
2. Then take the name of all the n sites as input from user
3. Create a cost array which stores the distance of a particular site to each and every other site. Take these distances as input from the user.
4. Initialize the path array which will store the shortest path possible with minimum cost
5. Initialize a variable to store the minimum cost.
6. Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
7. Generate all (n-1)! permutations of cities.
8. Calculate the cost of every permutation and keep track of the minimum cost permutation.
9. Return the permutation with minimum cost.

SOURCE CODE:

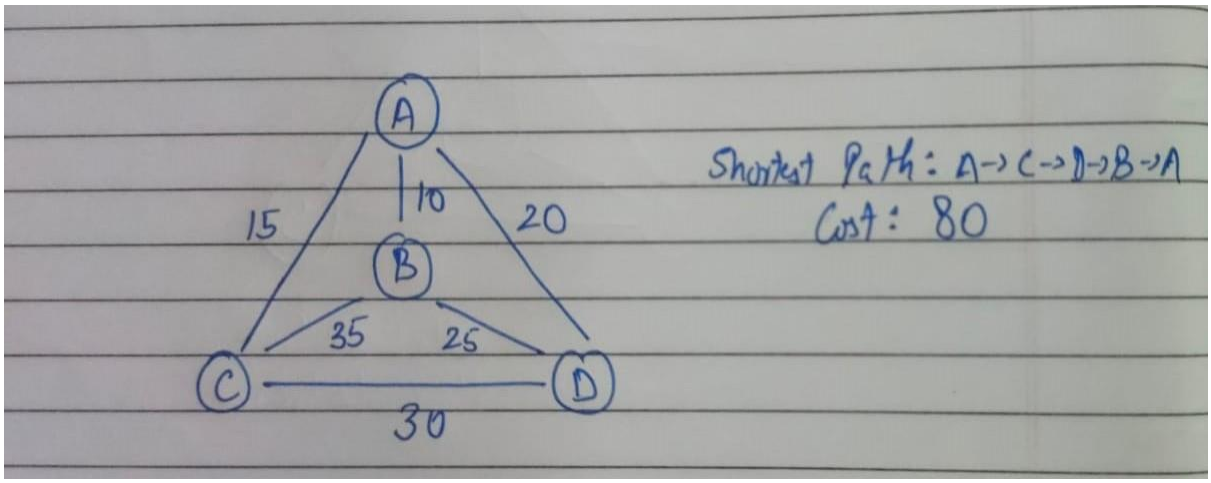
```
#include <bits/stdc++.h>
using namespace std;
#define V 4
int traverse(int **graph, int s, char *path, int n, char *sites)
{
    vector<int> vertex;
    for (int i = 0; i < n; i++)
    {
        if (i != s)
        {
            vertex.push_back(i);
        }
    }

    int min_path = INT_MAX;
    while (next_permutation(vertex.begin(), vertex.end()))
    {
        int current_pathweight = 0;
        int k = s;
        for (int i = 0; i < vertex.size(); i++)
        {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);
        if (min_path == current_pathweight)
        {
            path[0] = sites[s];
            for (int i = 0; i < vertex.size(); i++)
            {
                k = vertex[i];
                path[i + 1] = sites[k];
            }
            path[n] = sites[s];
        }
    }

    return min_path;
}
int main()
{
    int n;
    cout << "Enter the number of historic sites : ";
    cin >> n;
```

```
    cout << "Enter the name of historic sites : " << endl;
    char *sites = (char *)malloc(n * sizeof(char));
    for (int i = 0; i < n; i++)
    {
        cin >> sites[i];
    }

    int **distance = (int **)malloc(n * sizeof(int *));

    cout << "Enter the distance to travel from one city to all others : " <<
endl;
    for (int i = 0; i < n; i++)
    {
        distance[i] = (int *)malloc(n * sizeof(int));
        for (int j = 0; j < n; j++)
        {
            if (i != j)
            {
                cout << "Enter the distance from " << sites[i] << " -> " <<
sites[j] << " : ";
                cin >> distance[i][j];
            }
            else
            {
                distance[i][j] = 0;
            }
        }
    }

    char *path = (char *)malloc((n + 1) * sizeof(char));

    int s = 0;
    cout << "The minimum Cost to Travel acroos all the cities and come back to
the starting point is : " << traverse(distance, s, path, n, sites) << endl;
    cout << "The Path is : [ ";
    for (int i = 0; i < n + 1; i++)
    {
        cout << path[i] << " ";
    }
    cout << "]" << endl;
    return 0;
}
```

OUTPUT SCREENSHOT:




```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question2.exe"
Enter the number of historic sites : 4
Enter the name of historic sites :
A
B
C
D
Enter the distance to travel from one city to all others :
Enter the distance from A -> B : 2
Enter the distance from A -> C : 3
Enter the distance from A -> D : 4
Enter the distance from B -> A : 5
Enter the distance from B -> C : 7
Enter the distance from B -> D : 8
Enter the distance from C -> A : 3
Enter the distance from C -> B : 4
Enter the distance from C -> D : 6
Enter the distance from D -> A : 9
Enter the distance from D -> B : 8
Enter the distance from D -> C : 4
The minimum Cost to Travel acroos all the cities and come back to the starting point is : 17
The Path is : [ A D C B A ]
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```




```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> cd "c:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01"
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question2.exe"
Enter the number of historic sites : 4
Enter the name of historic sites :
a
b
c
d
Enter the distance to travel from one city to all others :
Enter the distance from a -> b : 10
Enter the distance from a -> c : 15
Enter the distance from a -> d : 20
Enter the distance from b -> a : 10
Enter the distance from b -> c : 35
Enter the distance from b -> d : 25
Enter the distance from c -> a : 15
Enter the distance from c -> b : 35
Enter the distance from c -> d : 30
Enter the distance from d -> a : 20
Enter the distance from d -> b : 25
Enter the distance from d -> c : 30
The minimum Cost to Travel acroos all the cities and come back to the starting point is : 80
The Path is : [ a c d b a ]
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```

QUESTION 3:

Design and implement an algorithm using divide and conquer approach that finds the maximum and minimum element from the elements stored in an array.

PSEUDOCODE:

1. Take the number of students participating in quiz as input from user - n
2. Allocate an array of size using dynamic memory allocation
3. Take the student marks as input from user
4. Initialize the minimum and maximum scores as the arrays first element. We assume that they are min and max scores at first

5. we divide the array into two equal parts and recursively find the maximum and minimum.
6. Basically we look for the middle element and check if its larger than the current maximum element if yes then we update the current maximum element. If the middle element is smaller than the current smallest element, then we update the current smallest element
7. We compare the maximum and minimum of those parts to get the maximum and minimum of the whole array.
8. Time complexity = O(n), Space Complexity = O(1)

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>

void dc(int *arr, int st, int end, int *max, int *min)
{
    if (st <= end)
    {
        int mid = st + (end - st) / 2;
        if (arr[mid] < *min)
        {
            *min = arr[mid];
        }
        if (arr[mid] > *max)
        {
            *max = arr[mid];
        }
        dc(arr, mid + 1, end, max, min);
        dc(arr, st, mid - 1, max, min);
    }
}

int main()
{
    int n;
    printf("Enter the Number of Elements in Array : \n");
    scanf("%d", &n);
    // Allocating a array
    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter the elements of array : \n");
    for (int i = 0; i < n; i++)
    {
        printf("Element %d : \n", i + 1);
        scanf("%d", &arr[i]);
    }
```

```
    // Initializing max and min by first element of array
    int max = arr[0];
    int min = arr[0];
    dc(arr, 0, n - 1, &max, &min);
    printf("The Maximum Element of the Array is  : %d\n", max);
    printf("The Minimum Element of the Array is : %d\n", min);
    return 0;
}
```

OUTPUT SCREENSHOT:

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question3.exe"
Enter the Number of Elements in Array :
10
Enter the elements of array :
Element 1 :
6
Element 2 :
5
Element 3 :
3
Element 4 :
8
Element 5 :
9
Element 6 :
1
Element 7 :
0
Element 8 :
7
Element 9 :
8
Element 10 :
5
The Maximum Element of the Array is  : 9
The Minimum Element of the Array is : 0
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question3.exe"
Enter the Number of Elements in Array :
5
Enter the elements of array :
Element 1 :
10
Element 2 :
10
Element 3 :
7
Element 4 :
8
Element 5 :
4
The Maximum Element of the Array is  : 10
The Minimum Element of the Array is : 4
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```

QUESTION 4:

Write a menu driven program to implement matrix multiplication and Strassen's matrix multiplication algorithm using divide and conquer approach.

PSEUDOCODE:

Menu Driver Program:

- Initialise variables by inputting required things such as the number of rows , columns.
- Check if the columns of first matrix are equal to rows of second matrix if not renter the dimensions.
- Check if the dimension of matrices are the power of 2 if not renter the dimensions.
- Input the matrix elements.
- Ask for the option to solve matrix multiplication using divide and conquer or strassen's matrix multiplication.
- If the choice is simple divide and conquer call the divide and conquer function and store the result.
- If the choice is stressen's method call the stressen function and store the result.
- Print the result.

Divide and conquer function:

- Let input matrices be A and B.
- Divide the input matrices into four parts of new size equals to half of original size. (dimension=n/2 x n/2)
- Then call the same function with the new smaller matrix as argument.
- The base case is when the size becomes 1 then return the multiplication of the matrices (which will be simple scaler multiplication now as size =1).
- Then add the four parts of each smaller matrix multiplied appropriately according to formula:
- R00 = A00 x B00 + A01 x B10
- R01 = A00 x B01 + A01 x B11
- R10 = A10 x B00 + A11 x B10
- R11 = A10 x B01 + A11 x B11

- Combine R00 R01 R10 R11 into one single matrix of original size of A and B of the current recursive call.
- And store the result in a result matrix and free the memory.
- Return the result matrix.

Strassen Matrix Multiplication Method function:

- Let input matrices be A and B.
- The matrix is first divided into four parts and then these four parts are combined back to get the resultant array these 4 parts are C11 C12 C13 C14 for the result matrix A11 A12 A21 A22 for the first matrix and B11 B12 B21 B22 for the second matrix
- Each element (eg: c11) contains 1/4 elements of the input matrix (dimension=n/2 x n/2)
- And this division is done until we reach the minimum most cases that is of size 2
- We call the same function recursively and use the following formula for calculation:
- C11 = S1 + S4 – S5 + S7
- C12 = S3 + S5
- C21 = S2 + S4
- C22 = S1 + S3 – S2 + S6
- Where,
- S1 = (A11 + A22) * (B11 + B22)
- S2 = (A21 + A22) * B11
- S3 = A11 * (B12 – B22)
- S4 = A22 * (B21 – B11)
- S5 = (A11 + A12) * B22
- S6 = (A21 – A11) * (B11 + B12)
- S7 = (A12 – A22) * (B21 + B22)
- Let us check if it is the same as the conventional approach.
- C12 = S3 + S5
- = A11 * (B12 – B22) + (A11 + A12) * B22
- = A11 * B12 – A11 * B22 + A11 * B22 + A12 * B22
- = A11 * B12 + A12 * B22
- The results are same.-
- Then after the recursive call, the matrices are combined back to get the matrix of the original size and the unused memory is freed and the result is returned.

Addition and Subtraction of two matrices takes $O(N^2)$ time.

Time Complexity for Simple Matrix Multiplicaiton using divide and Conquer:

Recurrence relation: **$T(N) = 8T(N/2) + O(N^2)$**

From Master's Theorem, time complexity is **$O(N^3)$**

Time Complexity for Strassen matrix multiplication:

Recurrence relation: **$T(N) = 7T(N/2) + O(N^2)$**

From Master's Theorem, time complexity is **$O(N^{Log7})$** which is approximately **$O(N^{2.8074})$**

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

// To Simple Print a matrix
void print(int **arr, int r, int c)
{
    printf("[");
    for (int i = 0; i < r; i++)
    {
        printf("\n  [ ");
        for (int j = 0; j < c; j++)
        {
            printf("%d ", arr[i][j]);
        }
        printf("]");
    }
    printf("\n]\n");
}

// Adding two matrix and storing in third
void add_matrix(int **a, int **b, int **c, int row, int column)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < column; j++)
```

```
        {
            c[i][j] += a[i][j] + b[i][j];
        }
    }
}

// Subtracting two matrix and storing in third
void sub_matrix(int **a, int **b, int **c, int row, int column)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < column; j++)
        {
            c[i][j] += a[i][j] - b[i][j];
        }
    }
}

// Simple Divide and Conquer Approach
int **dac(int **a, int **b, int r1, int c1, int r2, int c2)
{

    int **res = (int *)malloc(r1 * sizeof(int *));
    for (int i = 0; i < r1; i++)
    {
        res[i] = (int *)malloc(c2 * sizeof(int));
    }

    if (r1 == 1 && r2 == 1 && c1 == 1 && c2 == 1)
    {
        res[0][0] = a[0][0] * b[0][0];
    }
    else
    {
        // Allocating and Initializing the Arrays
        // 4 arrays for A and 4 for B
        int **a00 = malloc((r1 / 2) * sizeof(int *));
        int **a01 = malloc((r1 / 2) * sizeof(int *));
        int **a10 = malloc((r1 / 2) * sizeof(int *));
        int **a11 = malloc((r1 / 2) * sizeof(int *));

        int **b00 = malloc((r2 / 2) * sizeof(int *));
        int **b01 = malloc((r2 / 2) * sizeof(int *));
        int **b10 = malloc((r2 / 2) * sizeof(int *));
        int **b11 = malloc((r2 / 2) * sizeof(int *));

        for (int i = 0; i < (r1 / 2); i++)
        {
```

```
        a00[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a01[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a10[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a11[i] = (int *)malloc((c1 / 2) * sizeof(int));
    }

    for (int i = 0; i < (r2 / 2); i++)
    {
        b00[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b01[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b10[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b11[i] = (int *)malloc((c2 / 2) * sizeof(int));
    }

    for (int i = 0; i < (r1 / 2); i++)
    {
        for (int j = 0; j < (c1 / 2); j++)
        {
            a00[i][j] = a[i][j];
            a01[i][j] = a[i][j + (c1 / 2)];
            a10[i][j] = a[i + (r1 / 2)][j];
            a11[i][j] = a[i + (r1 / 2)][j + (c1 / 2)];
        }
    }

    for (int i = 0; i < (r2 / 2); i++)
    {
        for (int j = 0; j < (c2 / 2); j++)
        {
            b00[i][j] = b[i][j];
            b01[i][j] = b[i][j + (c2 / 2)];
            b10[i][j] = b[i + (r2 / 2)][j];
            b11[i][j] = b[i + (r2 / 2)][j + (c2 / 2)];
        }
    }

    // Allocating the result array
    int **res00 = malloc((r1 / 2) * sizeof(int *));
    int **res01 = malloc((r1 / 2) * sizeof(int *));
    int **res10 = malloc((r1 / 2) * sizeof(int *));
    int **res11 = malloc((r1 / 2) * sizeof(int *));

    for (int i = 0; i < (r1 / 2); i++)
    {
        res00[i] = (int *)malloc((c2 / 2) * sizeof(int));
        res01[i] = (int *)malloc((c2 / 2) * sizeof(int));
        res10[i] = (int *)malloc((c2 / 2) * sizeof(int));
        res11[i] = (int *)malloc((c2 / 2) * sizeof(int));
```

```
            for (int j = 0; j < (c2 / 2); j++)
            {
                res00[i][j] = 0;
                res01[i][j] = 0;
                res10[i][j] = 0;
                res11[i][j] = 0;
            }
        }

        // Performing addition on 2subparts of the problem
        // Total 8 divisions as the funciton is called 8 times
        // Total 4 additions as add function is called 4 times
        add_matrix(dac(a00, b00, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)),
dac(a01, b10, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)), res00, (r1 / 2), (c2 /
2));
        add_matrix(dac(a00, b01, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)),
dac(a01, b11, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)), res01, (r1 / 2), (c2 /
2));
        add_matrix(dac(a10, b00, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)),
dac(a11, b10, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)), res10, (r1 / 2), (c2 /
2));
        add_matrix(dac(a10, b01, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)),
dac(a11, b11, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2)), res11, (r1 / 2), (c2 /
2));

        // Storing the result in final matrix
        for (int i = 0; i < (r1 / 2); i++)
        {
            for (int j = 0; j < (c2 / 2); j++)
            {
                res[i][j] = res00[i][j];
                res[i][j + (c2 / 2)] = res01[i][j];
                res[(r1 / 2) + i][j] = res10[i][j];
                res[i + (r1 / 2)][j + (c2 / 2)] = res11[i][j];
            }
        }

        // Deallocating the unrequired space
        for (int i = 0; i < (r1 / 2); i++)
        {
            free(a00[i]);
            free(a01[i]);
            free(a10[i]);
            free(a11[i]);
        }
        free(a00);
        free(a01);
        free(a10);
```

```
        free(a11);
        for (int i = 0; i < (r2 / 2); i++)
        {
            free(b00[i]);
            free(b01[i]);
            free(b10[i]);
            free(b11[i]);
        }
        free(b00);
        free(b01);
        free(b10);
        free(b11);
        for (int i = 0; i < (r1 / 2); i++)
        {
            free(res00[i]);
            free(res01[i]);
            free(res10[i]);
            free(res11[i]);
        }
        free(res00);
        free(res01);
        free(res10);
        free(res11);
    }
    return res;
}

int **strassen(int **a, int **b, int r1, int c1, int r2, int c2)
{

    int **res = (int *)malloc(r1 * sizeof(int *));
    for (int i = 0; i < r1; i++)
    {
        res[i] = (int *)malloc(c2 * sizeof(int));
    }

    if (r1 == 1 && r2 == 1 && c1 == 1 && c2 == 1)
    {
        res[0][0] = a[0][0] * b[0][0];
    }
    else
    {
        // Allocating and Initializing the Arrays
        // 4 arrays for A and 4 for B
        int **a00 = malloc((r1 / 2) * sizeof(int *));
        int **a01 = malloc((r1 / 2) * sizeof(int *));
        int **a10 = malloc((r1 / 2) * sizeof(int *));
        int **a11 = malloc((r1 / 2) * sizeof(int *));
```

```
    int **b00 = malloc((r2 / 2) * sizeof(int *));
    int **b01 = malloc((r2 / 2) * sizeof(int *));
    int **b10 = malloc((r2 / 2) * sizeof(int *));
    int **b11 = malloc((r2 / 2) * sizeof(int *));

    for (int i = 0; i < (r1 / 2); i++)
    {
        a00[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a01[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a10[i] = (int *)malloc((c1 / 2) * sizeof(int));
        a11[i] = (int *)malloc((c1 / 2) * sizeof(int));
    }

    for (int i = 0; i < (r2 / 2); i++)
    {
        b00[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b01[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b10[i] = (int *)malloc((c2 / 2) * sizeof(int));
        b11[i] = (int *)malloc((c2 / 2) * sizeof(int));
    }

    for (int i = 0; i < (r1 / 2); i++)
    {
        for (int j = 0; j < (c1 / 2); j++)
        {
            a00[i][j] = a[i][j];
            a01[i][j] = a[i][j + (c1 / 2)];
            a10[i][j] = a[i + (r1 / 2)][j];
            a11[i][j] = a[i + (r1 / 2)][j + (c1 / 2)];
        }
    }

    for (int i = 0; i < (r2 / 2); i++)
    {
        for (int j = 0; j < (c2 / 2); j++)
        {
            b00[i][j] = b[i][j];
            b01[i][j] = b[i][j + (c2 / 2)];
            b10[i][j] = b[i + (r2 / 2)][j];
            b11[i][j] = b[i + (r2 / 2)][j + (c2 / 2)];
        }
    }

    // Allocating the result array
    int **res00 = malloc((r1 / 2) * sizeof(int *));
    int **res01 = malloc((r1 / 2) * sizeof(int *));
    int **res10 = malloc((r1 / 2) * sizeof(int *));
```

```
        int **res11 = malloc((r1 / 2) * sizeof(int *));

        for (int i = 0; i < (r1 / 2); i++)
        {
            res00[i] = (int *)malloc((c2 / 2) * sizeof(int));
            res01[i] = (int *)malloc((c2 / 2) * sizeof(int));
            res10[i] = (int *)malloc((c2 / 2) * sizeof(int));
            res11[i] = (int *)malloc((c2 / 2) * sizeof(int));
            for (int j = 0; j < (c2 / 2); j++)
            {
                res00[i][j] = 0;
                res01[i][j] = 0;
                res10[i][j] = 0;
                res11[i][j] = 0;
            }
        }

        // Allocating the 10 sum arrays which will store the computation of
arrays
        int **s1 = (int **)malloc((r2 / 2) * sizeof(int *));
        int **s2 = (int **)malloc((r1 / 2) * sizeof(int *));
        int **s3 = (int **)malloc((r1 / 2) * sizeof(int *));
        int **s4 = (int **)malloc((r2 / 2) * sizeof(int *));
        int **s5 = (int **)malloc((r1 / 2) * sizeof(int *));
        int **s6 = (int **)malloc((r2 / 2) * sizeof(int *));
        int **s7 = (int **)malloc((r1 / 2) * sizeof(int *));
        int **s8 = (int **)malloc((r2 / 2) * sizeof(int *));
        int **s9 = (int **)malloc((r1 / 2) * sizeof(int *));
        int **s10 = (int **)malloc((r2 / 2) * sizeof(int *));

        for (int i = 0; i < (r1 / 2); i++)
        {
            s2[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s3[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s5[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s7[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s9[i] = (int *)malloc((c2 / 2) * sizeof(int));
            for (int j = 0; j < (c1 / 2); j++)
            {
                s2[i][j] = 0;
                s3[i][j] = 0;
                s5[i][j] = 0;
                s7[i][j] = 0;
                s9[i][j] = 0;
            }
        }
        for (int i = 0; i < (r2 / 2); i++)
        {
```

```
            s1[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s4[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s6[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s8[i] = (int *)malloc((c2 / 2) * sizeof(int));
            s10[i] = (int *)malloc((c2 / 2) * sizeof(int));
            for (int j = 0; j < (c2 / 2); j++)
            {
                s1[i][j] = 0;
                s4[i][j] = 0;
                s6[i][j] = 0;
                s8[i][j] = 0;
                s10[i][j] = 0;
            }
        }

        // Performing 5 addition and 5 subtraction operations
        sub_matrix(b01, b11, s1, (r2 / 2), (c2 / 2));
        add_matrix(a00, a01, s2, (r1 / 2), (c1 / 2));
        add_matrix(a10, a11, s3, (r1 / 2), (c1 / 2));
        sub_matrix(b10, b00, s4, (r2 / 2), (c2 / 2));
        add_matrix(a00, a11, s5, (r1 / 2), (c1 / 2));
        add_matrix(b00, b11, s6, (r2 / 2), (c2 / 2));
        sub_matrix(a01, a11, s7, (r1 / 2), (c1 / 2));
        add_matrix(b10, b11, s8, (r2 / 2), (c2 / 2));
        sub_matrix(a00, a10, s9, (r1 / 2), (c1 / 2));
        add_matrix(b00, b01, s10, (r2 / 2), (c2 / 2));

        // Performing a total of 7 divisons
        int **p1 = strassen(a00, s1, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p2 = strassen(s2, b11, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p3 = strassen(s3, b00, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p4 = strassen(a11, s4, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p5 = strassen(s5, s6, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p6 = strassen(s7, s8, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));
        int **p7 = strassen(s9, s10, (r1 / 2), (c1 / 2), (r2 / 2), (c2 / 2));

        // Finally performing 3 addition and 3 subtraction operations to store
the result
        add_matrix(p5, p4, res00, (r1 / 2), (c2 / 2));
        sub_matrix(p6, p2, res00, (r1 / 2), (c2 / 2));

        add_matrix(p1, p2, res01, (r1 / 2), (c2 / 2));

        add_matrix(p3, p4, res10, (r1 / 2), (c2 / 2));

        sub_matrix(p5, p3, res11, (r1 / 2), (c2 / 2));
        sub_matrix(p1, p7, res11, (r1 / 2), (c2 / 2));
```

```
        for (int i = 0; i < (r1 / 2); i++)
        {
            for (int j = 0; j < (c2 / 2); j++)
            {
                res[i][j] = res00[i][j];
                res[i][j + (c2 / 2)] = res01[i][j];
                res[(r1 / 2) + i][j] = res10[i][j];
                res[i + (r1 / 2)][j + (c2 / 2)] = res11[i][j];
            }
        }

        // Deallocating the unrequired memory
        for (int i = 0; i < (r1 / 2); i++)
        {
            free(a00[i]);
            free(a01[i]);
            free(a10[i]);
            free(a11[i]);
        }
        free(a00);
        free(a01);
        free(a10);
        free(a11);
        for (int i = 0; i < (r2 / 2); i++)
        {
            free(b00[i]);
            free(b01[i]);
            free(b10[i]);
            free(b11[i]);
        }
        free(b00);
        free(b01);
        free(b10);
        free(b11);
        for (int i = 0; i < (r1 / 2); i++)
        {
            free(res00[i]);
            free(res01[i]);
            free(res10[i]);
            free(res11[i]);
        }
        free(res00);
        free(res01);
        free(res10);
        free(res11);
        for (int i = 0; i < (r1 / 2); i++)
        {
            free(s2[i]);
```

```
            free(s3[i]);
            free(s5[i]);
            free(s7[i]);
            free(s9[i]);
        }
        for (int i = 0; i < (r2 / 2); i++)
        {
            free(s1[i]);
            free(s4[i]);
            free(s6[i]);
            free(s8[i]);
            free(s10[i]);
        }
        free(s1);
        free(s2);
        free(s3);
        free(s4);
        free(s5);
        free(s6);
        free(s7);
        free(s8);
        free(s9);
        free(s10);
    }
    return res;
}

int main()
{
    // Menu Driven Program
    int flag = 1;

    while (flag == 1)
    {
        int r1, r2, c1, c2;
        printf("Enter the Number of rows of matrix A : ");
        scanf("%d", &r1);
        printf("Enter the Number of columns of matrix A : ");
        scanf("%d", &c1);
        printf("Enter the Number of rows of matrix B : ");
        scanf("%d", &r2);
        printf("Enter the Number of columns of matrix B : ");
        scanf("%d", &c2);
        if (c1 != r2)
        {
            printf("The Number of Columns of Matrix A must match with Number
of rows of matrix B\n");
            printf("Enter the Dimmensions again : \n");
```

```
            printf("Enter the Number of rows of matrix A : ");
            scanf("%d", &r1);
            printf("\nEnter the Number of columns of matrix A : ");
            scanf("%d", &c1);
            printf("\nEnter the Number of rows of matrix B : ");
            scanf("%d", &r2);
            printf("\nEnter the Number of columns of matrix B : \n");
            scanf("%d", &c2);
        }
        if ((ceil(log2(r1)) != floor(log2(r1))) || (ceil(log2(r2)) !=
floor(log2(r2))) || (ceil(log2(c1)) != floor(log2(c1))) || (ceil(log2(c2)) !=
floor(log2(c2))))
        {
            printf("The Dimmensions must be a power of 2\n");
            printf("Enter the Dimmensions again : \n");
            printf("Enter the Number of rows of matrix A : ");
            scanf("%d", &r1);
            printf("\nEnter the Number of columns of matrix A : ");
            scanf("%d", &c1);
            printf("\nEnter the Number of rows of matrix B : ");
            scanf("%d", &r2);
            printf("\nEnter the Number of columns of matrix B : \n");
            scanf("%d", &c2);
        }
        // Allocating the main arrays
        int **A = (int **)malloc(r1 * sizeof(int *));
        int **B = (int **)malloc(r2 * sizeof(int *));
        printf("\nEnter the Elements of matrix A rowwise : \n");
        for (int i = 0; i < r1; i++)
        {
            A[i] = (int *)malloc(c1 * sizeof(int));
            for (int j = 0; j < c1; j++)
            {
                printf("El: i = %d,j = %d : ", i, j);
                scanf("%d", &A[i][j]);
            }
        }
        printf("\nEnter the Elements of matrix B rowwise : \n");
        for (int i = 0; i < r2; i++)
        {
            B[i] = (int *)malloc(c2 * sizeof(int));
            for (int j = 0; j < c2; j++)
            {
                printf("El: i = %d,j = %d : ", i, j);
                scanf("%d", &B[i][j]);
            }
        }
        // Checking the choice
```

```
        printf("\n1 : Matrix Multiplication using Simple Divide and Conquer :
\n");
        printf("2 : Matrix Multiplication using Strassen's matrix
multiplication : \n");
        int choice;
        printf("Choice : ");
        scanf("%d", &choice);
        if (choice != 1 && choice != 2)
        {
            printf("Enter Correct Choice : \n");
            printf("Choice : ");
            scanf("%d", choice);
        }
        int **C;
        if (choice == 1)
        {
            C = dac(A, B, r1, c1, r2, c2);
        }
        else
        {
            C = strassen(A, B, r1, c1, r2, c2);
        }
        printf("The Matrix A is : \n");
        print(A, r1, c1);
        printf("The Matrix B is : \n");
        print(B, r2, c2);
        printf("The Result Matrix C is : \n");
        print(C, r1, c2);
        printf("Do You want to continue : 1: Yes , 0: No : ");
        scanf("%d", &flag);
    }

    return 0;
}
```

OUTPUT SCREENSHOT:

The Number of columns of matrix A must match number of columns of matrix B

```
PS C:\Users\Anirudh\OneDrive\Desktop\c codes> cd "c:\Users\Anirudh\OneDrive\Desktop\c co
des\DAA\AS01"
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01> & .\"question4.exe"
Enter the Number of rows of matrix A : 4
Enter the Number of columns of matrix A : 4
Enter the Number of rows of matrix B : 16
Enter the Number of columns of matrix B : 4
The Number of Columns of Matrix A must match with Number of rows of matrix B
Enter the Dimmensions again :
Enter the Number of rows of matrix A :
```

The Dimensions must be a power of 2

```
Enter the Dimmensions again :
Enter the Number of rows of matrix A : 3
Enter the Number of columns of matrix A : 4
Enter the Number of rows of matrix B : 4
Enter the Number of columns of matrix B : 3
The Dimmensions must be a power of 2
Enter the Dimmensions again :
Enter the Number of rows of matrix A :
```

Entering the Elements of the array

```
Enter the Elements of matrix A rowwise :
El: i = 0,j = 0 : 1
El: i = 0,j = 1 : 2
El: i = 0,j = 2 : 3
El: i = 0,j = 3 : 4
El: i = 1,j = 0 : 5
El: i = 1,j = 1 : 6
El: i = 1,j = 2 : 7
El: i = 1,j = 3 : 8
El: i = 2,j = 0 : 9
El: i = 2,j = 1 : 10
El: i = 2,j = 2 : 11
El: i = 2,j = 3 : 12
El: i = 3,j = 0 : 13
El: i = 3,j = 1 : 14
El: i = 3,j = 2 : 15
El: i = 3,j = 3 : 16

Enter the Elements of matrix B rowwise :
El: i = 0,j = 0 : 1
El: i = 0,j = 1 : 1
El: i = 0,j = 2 : 1
El: i = 0,j = 3 : 1
El: i = 1,j = 0 : 2
El: i = 1,j = 1 : 2
El: i = 1,j = 2 : 2
El: i = 1,j = 3 : 2
El: i = 2,j = 0 : 3
El: i = 2,j = 1 : 3
El: i = 2,j = 2 : 3
El: i = 2,j = 3 : 3
El: i = 3,j = 0 : 4
El: i = 3,j = 1 : 5
El: i = 3,j = 2 : 6
El: i = 3,j = 3 : 7
```

Output for Simple Divide and Conquer Matrix Multiplication

```
1 : Matrix Multiplication using Simple Divide and Conquer :
2 : Matrix Multiplication using Strassen's matrix multiplication :
Choice : 1
The Matrix A is :
[
  [ 1 2 3 4 ]
  [ 5 6 7 8 ]
  [ 9 10 11 12 ]
  [ 13 14 15 16 ]
]
The Matrix B is :
[
  [ 1 1 1 1 ]
  [ 2 2 2 2 ]
  [ 3 3 3 3 ]
  [ 4 5 6 7 ]
]
The Result Matrix C is :
[
  [ 30 34 38 42 ]
  [ 70 78 86 94 ]
  [ 110 122 134 146 ]
  [ 150 166 182 198 ]
]
Do You want to continue : 1: Yes , 0: No :
```

Continue the program

```
]
Do You want to continue : 1: Yes , 0: No : 1
Enter the Number of rows of matrix A : 4
Enter the Number of columns of matrix A : 4
Enter the Number of rows of matrix B : 4
Enter the Number of columns of matrix B : 4
```

Output for Strassen's Matrix Multiplication

```
1 : Matrix Multiplication using Simple Divide and Conquer :
2 : Matrix Multiplication using Strassen's matrix multiplication :
Choice : 2
The Matrix A is :
[
  [ 1 1 1 1 ]
  [ 2 2 2 2 ]
  [ 3 3 3 3 ]
  [ 2 2 2 2 ]
]
The Matrix B is :
[
  [ 1 1 1 1 ]
  [ 2 2 2 2 ]
  [ 3 3 3 3 ]
  [ 2 2 2 2 ]
]
The Result Matrix C is :
[
  [ 8 8 8 8 ]
  [ 16 16 16 16 ]
  [ 24 24 24 24 ]
  [ 16 16 16 16 ]
]
Do You want to continue : 1: Yes , 0: No :
```

End the Program

```
]
Do You want to continue : 1: Yes , 0: No : 0
PS C:\Users\Anirudh\OneDrive\Desktop\c codes\DAA\AS01>
```