**DAA LAB WEEK 3**

**ADIT LUHADIA**

**190911112**

**IT A**

**LAB 6 - DIVIDE AND CONQUER – I**

Q1. Quick Sort

#include <iostream>

using namespace std;

int c = 0;

void swap(int \*a, int \*b) // For swapping two numbers

{

    int t = \*a;

    c++;

    \*a = \*b;

    c++;

    \*b = t;

    c++;

}

int partition(int arr[], int low, int high) // For partitioning with pivot = arr[high]

{

    int pivot = arr[high];

    c++;

    int i = (low - 1);

    c++;

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

    {

        c++;

        if (arr[j] < pivot)

        {

            c++;

            i++;

            c++;

            swap(&arr[i], &arr[j]);

            c++;

        }

    }

    c++;

    swap(&arr[i + 1], &arr[high]);

    c++;

    c++;

    return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        c++;

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

        c++;

        quickSort(arr, low, pi - 1);

        c++;

        quickSort(arr, pi + 1, high);

        c++;

    }

}

int main()

{

    int n;

    c++;

    cout << "Enter size of array: ";

    c++;

    cin >> n;

    c++;

    int a[n];

    c++;

    cout << "Enter array elements: ";

    c++;

    for (int i = 0; i < n; i++)

    {

        c++;

        cin >> a[i];

        c++;

    }

    c++;

    quickSort(a, 0, n - 1);

    c++;

    cout << "Sorted array: ";

    c++;

    for (int i = 0; i < n; i++)

    {

        c++;

        cout << a[i] << " ";

        c++;

    }

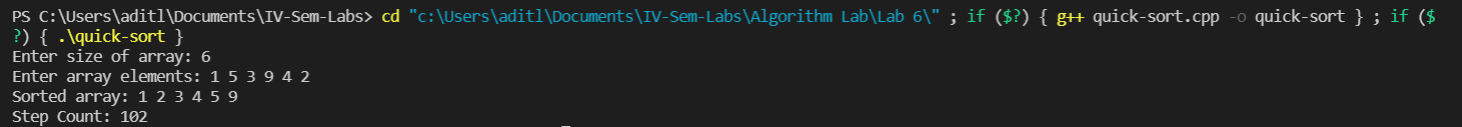
    c++;

    cout << endl;

    cout << "Step Count: " << c;

    return 0;

}



2. Merge Sort

#include <iostream>

using namespace std;

int c = 0;

void merge(int \*a, int l, int m, int r) // For merging the sub arrays

{

    int i, j, k, nl, nr;

    c++;

    nl = m - l + 1;

    c++;

    nr = r - m;

    c++;

    int la[nl], ra[nr];

    c++;

    for (i = 0; i < nl; i++)

    {

        c++;

        la[i] = a[l + i];

        c++;

    }

    c++;

    for (j = 0; j < nr; j++)

    {

        c++;

        ra[j] = a[m + 1 + j];

        c++;

    }

    c++;

    i = 0;

    c++;

    j = 0;

    c++;

    k = l;

    c++;

    while (i < nl && j < nr)

    {

        if (la[i] <= ra[j])

        {

            c++;

            a[k] = la[i];

            c++;

            i++;

            c++;

        }

        else

        {

            c++;

            a[k] = ra[j];

            c++;

            j++;

            c++;

        }

        k++;

        c++;

    }

    c++;

    while (i < nl)

    {

        c++;

        a[k] = la[i];

        c++;

        i++;

        k++;

        c++;

        c++;

    }

    c++;

    while (j < nr)

    {

        c++;

        a[k] = ra[j];

        c++;

        j++;

        k++;

        c++;

        c++;

    }

    c++;

}

void mergeSort(int \*a, int l, int r)

{

    int m;

    c++;

    if (l < r)

    {

        c++;

        int m = l + (r - l) / 2;

        c++;

        mergeSort(a, l, m);

        c++;

        mergeSort(a, m + 1, r);

        c++;

        merge(a, l, m, r);

        c++;

    }

}

int main()

{

    int n;

    c++;

    cout << "Enter number of elements: ";

    c++;

    cin >> n;

    c++;

    int a[n];

    c++;

    cout << "Enter elements:";

    c++;

    for (int i = 0; i < n; i++)

    {

        c++;

        cin >> a[i];

        c++;

    }

    mergeSort(a, 0, n - 1);

    c++;

    cout << "Sorted Array: ";

    c++;

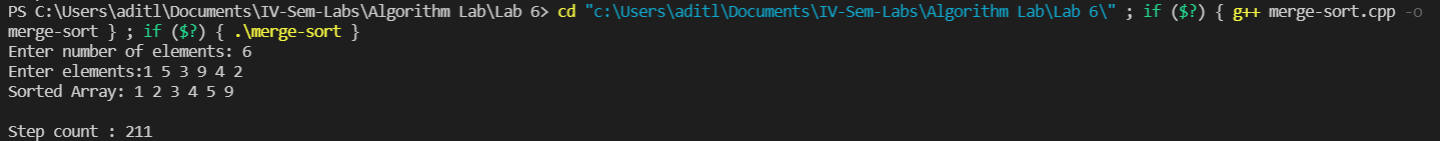
    for (int i = 0; i < n; i++)

        cout << a[i] << " ";

    cout << "\n\nStep count : " << c << endl;

    return 0;

}



**LAB 7 - DIVIDE AND CONQUER – II**

1. Write a Program to multiply two matrices using Strassen’s method and analyze the time complexity.

#include <iostream>

using namespace std;

void strassen\_multiply(int[5][5], int[5][5], int, int, int);

int display(int[5][5], int, int);

int main()

{

    int a[5][5], b[5][5], r1, c1, r2, c2;

    cout << "\n Enter rows for first matrix: ";

    cin >> r1; // Input rows for first matrix

    cout << "\n Enter columns for first matrix: ";

    cin >> c1; // Input columns for first matrix

    cout << "\n Enter rows for second matrix: ";

    cin >> r2; // Input rows for second matrix

    cout << "\n Enter columns for second matrix: ";

    cin >> c2; // Input columns for second matrix

    if (c1 != r2)

        return 0;

    cout << "\n Enter elements of first matrix \n";

    for (int i = 0; i < r1; i++)

    {

        for (int j = 0; j < c1; j++)

            cin >> a[i][j];

    }

    cout << "\n Enter elements of second matrix\n";

    for (int i = 0; i < r2; i++)

    {

        for (int j = 0; j < c2; j++)

            cin >> b[i][j];

    }

    display(a, r1, c1);

    display(b, r2, c2);

    strassen\_multiply(a, b, r1, c2, c1);

    return 0;

}

void strassen\_multiply(int a[5][5], int b[5][5], int row, int col, int c1)

{

    int c[5][5];

    for (int i = 0; i < row; i++)

    {

        for (int j = 0; j < col; j++)

            c[i][j] = 0;

    }

    for (int i = 0; i < row; i++)

    {

        for (int j = 0; j < col; j++)

        {

            for (int k = 0; k < c1; k++)

                c[i][j] += a[i][k] \* b[k][j];

        }

    }

    cout << "\n Matrix c after matrix multiplication is:\n";

    display(c, row, col);

}

int display(int c[5][5], int row, int col)

{

    cout << "\n Matrix is:\n";

    for (int i = 0; i < row; i++)

    {

        for (int j = 0; j < col; j++)

            cout << c[i][j] << " ";

        cout << "\n";

    }

    return 0;

}

Text

Description automatically generated

2. Write a program find the maximum and minimum of an array using divide and conquer technique.

#include <iostream>

using namespace std;

void minmax(int arr[], int l, int r, int &min, int &max)

{

    if (r - l <= 1)

    {

        if (arr[l] > arr[r])

        {

            if (min > arr[r])

                min = arr[r];

            if (max < arr[l])

                max = arr[l];

        }

        else

        {

            if (min > arr[l])

                min = arr[l];

            if (max < arr[l])

                max = arr[r];

        }

    }

    else

    {

        minmax(arr, l, (l + r) / 2, min, max);

        minmax(arr, (l + r) / 2, r, min, max);

    }

}

int main()

{

    int n;

    cout << "Enter size of array\n";

    cin >> n;

    cout << "Enter elements of array\n";

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int max = 0;

    int min = 99999;

    minmax(arr, 0, n - 1, min, max);

    cout << "The minimum element is " << min << endl;

    cout << "The maximum element is " << max << endl;

    return 0;

}

Text

Description automatically generated

**LAB 8 - DYNAMIC PROGRAMMING**

1. Write a Program to implement 0/1 Knapsack using dynamic programming technique and analyze the complexity.

#include <iostream>

using namespace std;

int max(int x, int y)

{

    return (x > y) ? x : y;

}

int knapSack(int W, int w[], int v[], int n)

{

    int i, wt;

    int K[n + 1][W + 1];

    for (i = 0; i <= n; i++)

    {

        for (wt = 0; wt <= W; wt++)

        {

            if (i == 0 || wt == 0)

                K[i][wt] = 0;

            else if (w[i - 1] <= wt)

                K[i][wt] = max(v[i - 1] + K[i - 1][wt - w[i - 1]], K[i - 1][wt]);

            else

                K[i][wt] = K[i - 1][wt];

        }

    }

    return K[n][W];

}

int main()

{

    cout << "Enter the number of items in a Knapsack:";

    int n, W;

    cin >> n;

    int v[n], w[n];

    for (int i = 0; i < n; i++)

    {

        cout << "Enter value and weight for item " << i << ":";

        cin >> v[i];

        cin >> w[i];

    }

    cout << "Enter the capacity of knapsack: ";

    cin >> W;

    cout << "The maximum profit is " << knapSack(W, w, v, n);

    return 0;

}

Text

Description automatically generated

2. Write a program to find the optimal order of multiplication in a chain of matrices and analyze the complexity.

#include <iostream>

#include <climits>

using namespace std;

int count = 0;

int matrixChainMultiplication(int dims[], int i, int j)

{

    if (j <= i + 1)

    {

        count++;

        return 0;

    }

    int min = INT\_MAX;

    for (int k = i + 1; k <= j - 1; k++)

    {

        count++;

        int cost = matrixChainMultiplication(dims, i, k);

        cost += matrixChainMultiplication(dims, k, j);

        cost += dims[i] \* dims[k] \* dims[j];

        if (cost < min)

        {

            count++;

            min = cost;

        }

    }

    return min;

}

int main()

{

    int n;

    cout << "Enter the size: ";

    cin >> n;

    int dims[n];

    cout << "Enter the sequence of matrices: ";

    for (int i = 0; i < n; i++)

    {

        count++;

        cin >> dims[i];

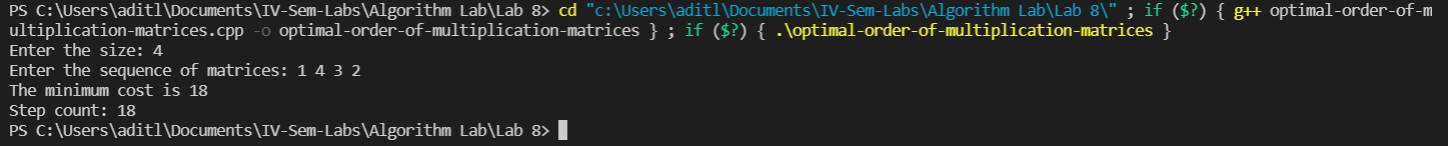
    }

    cout << "The minimum cost is " << matrixChainMultiplication(dims, 0, n - 1);

    cout << "\nStep count: " << count;

    return 0;

}



3. Write a program to find the shortest path between every pair of vertices in a given graph using dynamic programming technique.

#include <iostream>

#include <climits>

using namespace std;

#define N 4

#define I INT\_MAX

int count = 0;

void printPath(int path[][N], int v, int u)

{

    if (path[v][u] == v)

    {

        count++;

        return;

    }

    printPath(path, v, path[v][u]);

    cout << path[v][u] << " ";

}

void printSolution(int cost[N][N], int path[N][N])

{

    for (int v = 0; v < N; v++)

    {

        count++;

        for (int u = 0; u < N; u++)

        {

            count++;

            if (u != v && path[v][u] != -1)

            {

                count++;

                cout << "The shortest path from " << v << "<<" << u << " is (" << v << " ";

                printPath(path, v, u);

                cout << u << ")" << endl;

            }

        }

    }

}

void floydWarshall(int adjMatrix[][N])

{

    int cost[N][N], path[N][N];

    for (int v = 0; v < N; v++)

    {

        count++;

        for (int u = 0; u < N; u++)

        {

            count++;

            cost[v][u] = adjMatrix[v][u];

            if (v == u)

            {

                count++;

                path[v][u] = 0;

            }

            else if (cost[v][u] != INT\_MAX)

            {

                count++;

                path[v][u] = v;

            }

            else

            {

                count++;

                path[v][u] = -1;

            }

        }

    }

    for (int k = 0; k < N; k++)

    {

        count++;

        for (int v = 0; v < N; v++)

        {

            count++;

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

            {

                count++;

                if (cost[v][k] != INT\_MAX && cost[k][u] != INT\_MAX && cost[v][k] + cost[k][u] < cost[v][u])

                {

                    count++;

                    cost[v][u] = cost[v][k] + cost[k][u];

                    path[v][u] = path[k][u];

                }

            }

            if (cost[v][v] < 0)

            {

                count++;

                cout << "Negative-weight cycle found!!";

                return;

            }

        }

    }

    printSolution(cost, path);

}

int main()

{

    int adjMatrix[N][N] =

        {

            {1, 0, -2, I},

            {I, 1, 3, I},

            {3, I, 0, 2},

            {0, -1, -2, 0}};

    floydWarshall(adjMatrix);

    return 0;

}

Text

Description automatically generated