DAA LAB EXPERIMENT SOLUTIONS

Anurag Pratap Singh

DSAI- (221020411)

1.1 Binary Search (Recursive and Iterative)

CODE:

```
// 1.1 a Binary search - Recursive.
#include <iostream>
#include <cmath>

using namespace std;

int binarySearch(int arr[], int low, int high, int target) {
    if (high >= low) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == target)
            return mid;

        if (arr[mid] > target)
            return binarySearch(arr, low, mid - 1, target);

        return binarySearch(arr, mid + 1, high, target);
    }

    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 10;
    int result = binarySearch(arr, 0, n - 1, target);
    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in array" << endl;
    return 0;
}
```

Solution:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documents\codes\output'
PS C:\Users\HP\OneDrive\Documents\codes\output> & .\'(1a).exe'
Element found at index 3
PS C:\Users\HP\OneDrive\Documents\codes\output>
```

1.1 b Binary search - Iterative.

CODE:

```
#include <iostream>
#include <cmath>

int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target)
            return mid;

        if (arr[mid] < target)
            left = mid + 1;

        else
            right = mid - 1;
    }

    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int target = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    int result = binarySearch(arr, 0, n - 1, target);

    if (result == -1)
        std::cout << "Element not present in array";
    else
        std::cout << "Element found at index " << result;

    return 0;
}
```

Both Binary Search Iterative and Recursive takes O(logn) time.

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documents
PS C:\Users\HP\OneDrive\Documents\codes\output> &
Element found at index 3
PS C:\Users\HP\OneDrive\Documents\codes\output>
```

1.2 Merge Sort

CODE:

```
#include <iostream>
#include <cmath>

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];


    int i = 0;
    int j = 0;
    int k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }


    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);


        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size) {
```

```cpp
    for (int i = 0; i < size; i++)
        std::cout << A[i] << " ";
    std::cout << std::endl;
}

int main() {
    int arr[] = {27, 1, 13, 59, 60, 37};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    std::cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    return 0;
}
```

TimeComplexity of Merge Sort 0(nlogn)

OUTPUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Docum
PS C:\Users\HP\OneDrive\Documents\codes\output
Given array is
27 1 13 59 60 37

Sorted array is
1 13 27 37 59 60
```

1.3 Quick Sort

CODE:

```cpp
#include <iostream>
#include <cmath>

void swap(int& a, int& b) {
```

```cpp
    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 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);
    }
}

int main() {
    int arr[] = {12, 9, 7, 15, 10, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Given array : ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    quickSort(arr, 0, n - 1);

    std::cout << "Sorted array (Acending order) : ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

TimeComplexity of Quicksort with best case is O(nlogn) and worst case is O(n^2)

OUT PUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Docume
PS C:\Users\HP\OneDrive\Documents\codes\output>
Given array : 12 9 7 15 10 5
Sorted array (Acending order) : 5 7 9 10 12 15
```

1.4

Given a sorted array of non-repeated integers A[1...n], n > 1 then check whether there is an index i for which A[i] = i. Give an algorithm that runs in O(logn) time.

CODE:

```cpp
#include <iostream>
#include <cmath>

bool isIndexEqualToValue(int arr[], int size) {
    int low = 0;
    int high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == mid)
            return true;
        else if (arr[mid] < mid)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return false;
}

int main() {
```

```
    int arr[] = {-10, -5, 0, 3, 7, 9, 12, 15};
    int size = sizeof(arr) / sizeof(arr[0]);

    if (isIndexEqualToValue(arr, size))
        std::cout << "Yes, there exists an index i for which A[i] =
i.\n";
    else
        std::cout << "No, there doesn't exist an index i for which A[i]
= i.\n";

    return 0;
}
```

OUT PUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Document
PS C:\Users\HP\OneDrive\Documents\codes\output> &
Yes, there exists an index i for which A[i] = i.
```

2. Divide–and–Conquer

2.1 Strassen's Matrix Multiplication

CODE:

```
#include<iostream>
using namespace std;
int main() {
   int z[2][2];
   int i, j;
   int m1, m2, m3, m4 , m5, m6, m7;
      int x[2][2] = {
         {12, 34},
         {22, 10}
      };
   int y[2][2] = {
      {3, 4},
      {2, 1}
   };
   cout<<"The first matrix is: ";
```

```
    for(i = 0; i < 2; i++) {
        cout<<endl;
        for(j = 0; j < 2; j++)
            cout<<x[i][j]<<" ";
    }
    cout<<"\nThe second matrix is: ";
    for(i = 0;i < 2; i++){
        cout<<endl;
        for(j = 0;j < 2; j++)
            cout<<y[i][j]<<" ";
    }

    m1 = (x[0][0] + x[1][1]) * (y[0][0] + y[1][1]);
    m2 = (x[1][0] + x[1][1]) * y[0][0];
    m3 = x[0][0] * (y[0][1] - y[1][1]);
    m4 = x[1][1] * (y[1][0] - y[0][0]);
    m5 = (x[0][0] + x[0][1]) * y[1][1];
    m6 = (x[1][0] - x[0][0]) * (y[0][0]+y[0][1]);
    m7 = (x[0][1] - x[1][1]) * (y[1][0]+y[1][1]);

    z[0][0] = m1 + m4- m5 + m7;
    z[0][1] = m3 + m5;
    z[1][0] = m2 + m4;
    z[1][1] = m1 - m2 + m3 + m6;

    cout<<"\nProduct achieved using Strassen's algorithm: ";
    for(i = 0; i < 2 ; i++) {
        cout<<endl;
        for(j = 0; j < 2; j++)
            cout<<z[i][j]<<" ";
    }
    return 0;
}
```

Timecomplexity of this code is O($n^{\log 7}$)

OUTPUT:

```
The first matrix is:
12 34
22 10
The second matrix is:
3 4
2 1
Product achieved using Strassen's algorithm:
104 82
86 98
```

3. Miscellaneous Examples based on Divide and Conquer Algorithms

3.1 Given an array of n elements. Find whether there are two elements in the array such that their sum is equal to given element K or not? in O(nlogn) time.

CODE:

```
#include <iostream>
#include <cmath>
#include <algorithm>

bool hasPairWithSum(int arr[], int n, int K) {
    std::sort(arr, arr + n);

    int left = 0;
    int right = n - 1;

    while (left < right) {
        int sum = arr[left] + arr[right];
        if (sum == K)
            return true;
        else if (sum < K)
            left++;
        else
            right--;
    }
    return false;
}

int main() {
```

```
    int arr[] = {1, 4, 5, 6, 8, 9};
    int n = sizeof(arr) / sizeof(arr[0]);
    int K = 10;

    if (hasPairWithSum(arr, n, K))
        std::cout << "Yes, there are two elements with sum " << K <<
std::endl;
    else
        std::cout << "No, there are no two elements with sum " << K <<
std::endl;

    return 0;
}
```

OUT PUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Docum
PS C:\Users\HP\OneDrive\Documents\codes\output
Yes, there are two elements with sum 10
```

3.2 Given an array of n elements. Find whether there are three elements in the array such that their sum is equal to given element K or not? in O(n2) time.

CODE:

```
#include <iostream>

using namespace std;

int main() {
    // Write C++ code here
    int n;
    cout << "Enter the size of the array : ";
    cin >> n;

    int a[n];
    for (int i = 0 ; i < n ; i++){
        cout << "Enter the element for the position '" << i << "' in
the array :";
```

```
        cin >> a[i];
    }

    cout << "The entered array is : ";
    for (int i = 0 ; i < n ; i++){
        cout << a[i] << "  ";
    }
    cout << endl;

        int K;
    cout << "Enter the value of sum (k) : ";
    cin >> K;

    bool found = false;

    for (int i = 0; i < n - 2; i++) {
        for (int j = i + 1; j < n - 1; j++) {
            for (int k = j + 1; k < n; k++) {
                if (a[i] + a[j] + a[k] == K) {
                    cout << "Elements found: " << a[i] << ", " << a[j]
<< ", " << a[k] << endl;
                    found = true;
                    break;
                }
            }
            if (found) break;
        }
        if (found) break;
    }

    if (!found) {
        cout << "No such elements found." << endl;
    }

    return 0;
}
```

OUT PUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documents\codes\
PS C:\Users\HP\OneDrive\Documents\codes\output> & .\'(1a)
Enter the size of the array : 5
Enter the element for the position '0' in the array :3
Enter the element for the position '1' in the array :4
Enter the element for the position '2' in the array :2
Enter the element for the position '3' in the array :1
Enter the element for the position '4' in the array :5
The entered array is : 3  4  2  1  5
Enter the value of sum (k) : 15
No such elements found.
```

3.3 Let A and B be two arrays of n elements. Given a number K, draw an O(nlogn) time algorithm for determining whether there exists a ∈ A, b ∈ B such that a+b = K or not?

CODE:

```
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;
void findPairs(int A[], int B[], int n, int K) {
    sort(A, A + n);
    sort(B, B + n);

    int i = 0;
    int j = n - 1;

    while (i < n && j >= 0) {
        int sum = A[i] + B[j];
        if (sum == K) {
            cout << "Pair found: " << A[i] << " + " << B[j] << " = " <<
K << endl;
            i++;
            j--;
        } else if (sum < K) {
            i++;
        } else {
            j--;
        }
```

```
    }
}

int main() {
    int n;
    cout << "Enter the size of arrays A and B: ";
    cin >> n;

    int A[n], B[n];
    cout << "Enter elements of array A: ";
    for (int i = 0; i < n; ++i) {
        cin >> A[i];
    }

    cout << "Enter elements of array B: ";
    for (int i = 0; i < n; ++i) {
        cin >> B[i];
    }

    int K;
    cout << "Enter the value of K: ";
    cin >> K;

    findPairs(A, B, n, K);

    return 0;
}
```

OUT PUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documents\codes\output'
PS C:\Users\HP\OneDrive\Documents\codes\output> & .\'(1a).exe'
Enter the size of arrays A and B: 4,7
Enter elements of array A: Enter elements of array B: Enter the value of K:
PS C:\Users\HP\OneDrive\Documents\codes\output> 4,7,5
4
7
5
```

3.4 Given an array of n elements, give an algorithm for checking whether there are any duplicate elements in the array or not? in O(nlogn) time.

CODE:

```cpp
#include <iostream>

using namespace std;

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    int a[n];

    int maxElement = 0;

    cout << "Enter the elements of the array:\n";
    for (int i = 0; i < n; i++) {
        cout << "Element " << i << ": ";
        cin >> a[i];
        if (a[i] > maxElement) {
            maxElement = a[i];
        }
    }

    int frequency[maxElement + 1] = {0};
    bool duplicatesFound = false;

    cout << "The duplicate values (only 2) are: ";
    for (int i = 0; i < n; i++) {
        frequency[a[i]]++;
    }

    for (int i = 0; i <= maxElement; i++) {
        if (frequency[i] > 1 && frequency[i] <= 2) {
            cout << i << " ";
            duplicatesFound = true;
        }
    }

    cout << "\nThe elements with more than 2 duplicates are: ";
    bool moreThanTwice = false;
    for (int i = 0; i <= maxElement; i++) {
        if (frequency[i] > 2) {
            cout << i << " ";
            moreThanTwice = true;
        }
    }

    if (!duplicatesFound && !moreThanTwice) {
        cout << "None";
```

```
    }

    cout << endl;

    return 0;
}
```

OUT PUT:

```
Enter the size of the array: 20
Enter the elements of the array:
Element 0: 123456789547864325679
Element 1: Element 2: Element 3: Element 4: Element 5: Element 6: Element 7: Element 8: Element 9: Element 10: Element 11: Element 12: Element 13:
 Element 14: Element 15: Element 16: Element 17: Element 18: Element 19:
PS C:\Users\HP\OneDrive\Documents\codes\output>
```

3.5 Given an array of n elements, give an algorithm for finding the element which appears maximum number of times in the array in O(nlogn) time.

CODE:

```
#include <iostream>
#include <cmath>
#include <algorithm>

int findMaxFrequency(int arr[], int n) {

    std::sort(arr, arr + n);

    int maxCount = 1;
    int res = arr[0];
    int currCount = 1;

    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1]) {
            currCount++;
        } else {
            if (currCount > maxCount) {
                maxCount = currCount;
                res = arr[i - 1];
            }
            currCount = 1;
        }
    }

    if (currCount > maxCount) {
        maxCount = currCount;
```

```
        res = arr[n - 1];
    }

    return res;
}

int main() {
    int arr[] = {3, 2, 1, 2, 2, 3, 4, 5, 2, 2};
    int n = sizeof(arr) / size of(arr[0]);

    std::cout << "Element with maximum frequency: " <<
findMaxFrequency(arr, n) << std::endl;

    return 0;
}
```

OUT PUT:

```
PS C:\Users\HP\OneDrive\Documents\codes\ou
Element with maximum frequency: 2
```

4. Greedy Method

4.1 Knapsack Problem

CODE:

```
// 4.1 Knapsack Problem
#include <bits/stdc++.h>
using namespace std;

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)
{
    if (n == 0 || W == 0)
        return 0;

    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
    else
        return max(
            val[n - 1]
                + knapSack(W - wt[n - 1], wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}
```

```
int main()
{
    int profit[] = { 60, 100, 120 };
    int weight[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    cout << knapSack(W, weight, profit, n);
    return 0;
}
```

Time Complexity: O(2N)

OUT PUT:

```
220
```

4.2 Job sequencing with deadlines algorithm

CODE:

```
#include <algorithm>

#include <iostream>

using namespace std;


struct Job {

    char id;

    int dead;

    int profit;

};


bool comparison(Job a, Job b) {

    return (a.profit > b.profit);

}
```

```
void printJobScheduling(Job arr[], int n) {
    sort(arr, arr + n, comparison);

    int result[n];
    bool slot[n];

    for (int i = 0; i < n; i++)
        slot[i] = false;

    for (int i = 0; i < n; i++) {
        for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {
            if (slot[j] == false) {
                result[j] = i;
                slot[j] = true;
                break;
            }
        }
    }

    for (int i = 0; i < n; i++)
        if (slot[i])
            cout << arr[result[i]].id << " ";
}

int main() {
    Job arr[] = { { 'a', 2, 100 },
                  { 'b', 1, 19 },
                  { 'c', 2, 27 },
```

```
                { 'd', 1, 25 },
                { 'e', 3, 15 } };

    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Following is maximum profit sequence of jobs " << endl;

    printJobScheduling(arr, n);
    return 0;
}
```

OUT PUT:

```
Following is maximum profit sequence of jobs
c a e
```

TimeComplexity : O(N2)

4.3 Prim’s Algorithm for finding the minimal spanning trees.

CODE:

```
#include <bits/stdc++.h>
using namespace std;

#define V 5

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
```

```
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t"
            << graph[i][parent[i]] << " \n";
}

void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
```

```
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

int main() {
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };

    primMST(graph)
```

TimeComplexity : O(V2)

OUTPUT:

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

TimeComplexity :

4.4 Krushkal’s Algorithm for finding the minimal spanning trees.

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Edge {
    int src, dest, weight;
};

struct Subset {
    int parent;
    int rank;
};

class Graph {
    int V, E;
    vector<Edge> edges;

public:
    Graph(int V, int E) {
        this->V = V;
        this->E = E;
    }

    void addEdge(int src, int dest, int weight) {
        Edge edge;
        edge.src = src;
        edge.dest = dest;
        edge.weight = weight;
        edges.push_back(edge);
    }

    int find(Subset subsets[], int i) {
        if (subsets[i].parent != i)
            subsets[i].parent = find(subsets, subsets[i].parent);
        return subsets[i].parent;
    }

    void Union(Subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
```

```
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    void KruskalMST() {
        vector<Edge> result;

        sort(edges.begin(), edges.end(), [](const Edge &a, const Edge
&b) {
            return a.weight < b.weight;
        });

        Subset *subsets = new Subset[V];

        for (int v = 0; v < V; ++v) {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        int i = 0, e = 0;
        while (e < V - 1 && i < E) {

            Edge next_edge = edges[i++];

            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);

            if (x != y) {
                result.push_back(next_edge);
                Union(subsets, x, y);
                e++;
            }
        }

        cout << "Edges of MST:\n";
        int minimumCost = 0;
        for (i = 0; i < e; ++i) {
            cout << result[i].src << " - " << result[i].dest << "
 Weight: " << result[i].weight << endl;
            minimumCost += result[i].weight;
        }
        cout << "Minimum Cost of MST: " << minimumCost << endl;

        delete[] subsets;
```

```
    }
};

int main() {
    int V = 4;
    int E = 5;
    Graph graph(V, E);

    graph.addEdge(0, 1, 10);
    graph.addEdge(0, 2, 6);
    graph.addEdge(0, 3, 5);
    graph.addEdge(1, 3, 15);
    graph.addEdge(2, 3, 4);

    graph.KruskalMST();

    return 0;
}
```

TimeComplexity: O(E * logV)

OUT PUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive
PS C:\Users\HP\OneDrive\Documents\codes\
Edges of MST:
2 - 3  Weight: 4
0 - 3  Weight: 5
0 - 1  Weight: 10
Minimum Cost of MST: 19
```

4.5 Dijkstra’s Algorithm

CODE:

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f

typedef pair<int, int> iPair;

class Graph {
```

```
    int V;
    list<pair<int, int> >* adj;

public:
    Graph(int V);

    void addEdge(int u, int v, int w);

    void shortestPath(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<iPair>[V];
}

void Graph::addEdge(int u, int v, int w)
{
    adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w));
}

void Graph::shortestPath(int src)
{
    priority_queue<iPair, vector<iPair>, greater<iPair> >
        pq;
    vector<int> dist(V, INF);
    pq.push(make_pair(0, src));
    dist[src] = 0;
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        list<pair<int, int> >::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i) {
            int v = (*i).first;
            int weight = (*i).second;
            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
            }
        }
    }
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}
```

```
int main()
{
    int V = 8;
    Graph g(V);

    g.addEdge(1 , 2 , 2);
    g.addEdge(1 , 4 , 1);
    g.addEdge(3 , 1 , 4);
    g.addEdge(4 , 3 , 2);
    g.addEdge(3 , 6 , 5);
    g.addEdge(4 , 6 , 8);
    g.addEdge(7 , 6 , 1);
    g.addEdge(4 , 7 , 4);
    g.addEdge(4 , 5 , 2);
    g.addEdge(5 , 7 , 6);
    g.addEdge(2 , 4 , 3);
    g.addEdge(2 , 5 , 10);

    g.shortestPath(1);

    return 0;
}
```

Time Complexity: O(V2)

OUT PUT:

```
PS C:\Users\HP\OneDrive\Documents\code
Vertex Distance from Source
0               1061109567
1               0
2               2
3               3
4               1
5               3
6               6
7               5
```

5. Dynamic Programming

5.1 Finding the optimal order of multiplying n matrices.

CODE:

```cpp
#include <iostream>
#include <climits>

using namespace std;

int matrixChainOrder(int p[], int n) {
    int m[n][n];

    for (int i = 1; i < n; i++)
        m[i][i] = 0;

    for (int L = 2; L < n; L++) {
        for (int i = 1; i < n - L + 1; i++) {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n-1];
}

int main() {
    int arr[] = {10, 20, 30, 40, 30};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Minimum number of multiplications is: " <<
matrixChainOrder(arr, n) << endl;

    return 0;
}
```

TimeComplexity : 0(n3)

OUT PUT:-

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documen
PS C:\Users\HP\OneDrive\Documents\codes\output>
Minimum number of multiplications is: 30000
```

5.2 Construction of OBST

CODE:

```
// 5.2 Construction of OBST.
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

int cost(const vector<int>& freq, int i, int j) {
    int sum = 0;
    for (int k = i; k <= j; k++) {
        sum += freq[k];
    }
    return sum;
}

int constructOBST(const vector<int>& keys, const vector<int>& freq) {
    int n = keys.size();
    vector<vector<int>> dp(n + 1, vector<int>(n + 1, 0));

    for (int i = 0; i < n; i++) {
        dp[i][i] = freq[i];
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len + 1; i++) {
            int j = i + len - 1;
            dp[i][j] = INT_MAX;
            for (int r = i; r <= j; r++) {
                int c = ((r > i) ? dp[i][r - 1] : 0) +
                        ((r < j) ? dp[r + 1][j] : 0) + cost(freq, i,
j);
                if (c < dp[i][j]) {
                    dp[i][j] = c;
                }
            }
        }
    }
    return dp[0][n - 1];
}

int main() {
    vector<int> keys = {10, 12, 20, 35};
    vector<int> freq = {34, 8, 50, 25};
    int minCost = constructOBST(keys, freq);
```

```
    cout << "The cost of constructing optimal BST is: " << minCost <<
endl;
    return 0;
}
```

TimeComplexity: O(n3)

OUTPUT:

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documen
PS C:\Users\HP\OneDrive\Documents\codes\output>
The cost of constructing optimal BST is: 192
```

5.3 0/1 Knapsack Problem.

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

int knapsack(int W, vector<int>& wt, vector<int>& val, int n) {
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (wt[i - 1] <= w) {
                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]],
dp[i - 1][w]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[n][W];
}

int main() {
    int W = 50;
    vector<int> val = {50, 110, 180};
    vector<int> wt = {10, 20, 30};
    int n = val.size();
```

```
    cout << "Maximum value that can be obtained: " << knapsack(W, wt,
val, n) << endl;

    return 0;
}
```

OUT PUT:

TimeComplexity: O(n×W)

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Docume
PS C:\Users\HP\OneDrive\Documents\codes\output>
Maximum value that can be obtained: 290
```

5.4 All pairs shortest path problem

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

const int INF = 1e9;

void floydWarshall(vector<vector<int>>& graph, int V) {
    vector<vector<int>> dist(V, vector<int>(V));

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
        }
    }

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

                if (dist[i][k] != INF && dist[k][j] != INF &&
dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
```

```
    }

    cout << "Shortest distances between all pairs of vertices:\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF) {
                cout << "INF ";
            } else {
                cout << dist[i][j] << " ";
            }
        }
        cout << endl;
    }
}

int main() {
    int V = 4;
    vector<vector<int>> graph = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };

    floydWarshall(graph, V);

    return 0;
}
```

OUT PUT:

TimeComplexity : O(V3)

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Documents
PS C:\Users\HP\OneDrive\Documents\codes\output> &
Shortest distances between all pairs of vertices:
0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0
```

5.5 Traveling Salesmen Problem

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

const int INF = 1e9;

int tsp(const vector<vector<int>>& graph, int n) {
    vector<vector<int>> dp(1 << n, vector<int>(n, INF));
    dp[1][0] = 0;

    for (int mask = 1; mask < (1 << n); mask++) {
        for (int u = 0; u < n; u++) {
            if ((mask & (1 << u)) == 0) continue;
            for (int v = 0; v < n; v++) {
                if (u == v || (mask & (1 << v))) continue;
                dp[mask | (1 << v)][v] = min(dp[mask | (1 << v)][v],
dp[mask][u] + graph[u][v]);
            }
        }
    }

    int ans = INF;
    for (int i = 0; i < n; i++) {
        ans = min(ans, dp[(1 << n) - 1][i] + graph[i][0]);
    }
    return ans;
}

int main() {
    int n = 4;
    vector<vector<int>> graph = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    int minCost = tsp(graph, n);
    cout << "Minimum cost to visit all cities: " << minCost << endl;

    return 0;
}
```
OUT PUT:

TImeComplexity: O(2^n X n^2)

```
PS C:\Users\HP> cd 'c:\Users\HP\OneDrive\Doc
PS C:\Users\HP\OneDrive\Documents\codes\outp
Minimum cost to visit all cities: 80
```