DAA LAB

1.GCD

```
#include <iostream>
#include <ctime>
#include <iomanip>
using namespace std;

int euclid(int m, int n);
int recursive(int m, int n);
int middle_school(int m, int n);
int consecutive_integer(int m, int n);
int rem = 0, res = 0;

int main()
{
   int m, n;
   clock_t start, end;
   double cpu_time_used;
   cout << "Enter the first number: ";
   cin >> m;
   cout << "Enter the second number: ";

   cin >> n;
      cout << fixed << setprecision(6);

   if(n != 0 && m != 0)
   {
      start = clock();
      res = euclid(m, n);
      end = clock();
      cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
      cout << "1. Euclidean Method \n GCD = " << res << ", Runtime: " << cpu_time_used << " seconds\n";
```

```
        start = clock();
        res = recursive(m, n);
        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
        cout << "2. Recursive Method \n GCD = " << res << ", Runtime: " << cpu_time_used << " seconds\n";

        start = clock();
        res = middle_school(m, n);
        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
        cout << "3. Middle School Method \n GCD = " << res << ", Runtime: " << cpu_time_used << " seconds\n";

        start = clock();
        res = consecutive_integer(m, n);
        end = clock();
        cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
        cout << "4. Consecutive Integer Method \n GCD = " << res << ", Runtime: " << cpu_time_used << " seconds\n";
    }
    else if(m==0)
        cout << "Invalid inputs!\n";
    return 0;
}

int euclid(int m, int n)
{
    while(n != 0)
    {
        rem = m % n;
        m = n;
        n = rem;
    }
    return m;
```

```
}

int recursive(int m, int n)
{
   rem = m % n;
   if(rem == 0)
      return n;
   else
      return recursive(n, rem);
}

int middle_school(int m, int n)
{
   int gcd = 1;

   int min = (m < n) ? m : n;

   for (int i = 2; i <= min; ++i)
   {
      while (m % i == 0 && n % i == 0)
      {
         gcd *= i;
         m /= i;
         n /= i;
      }
   }

   return gcd;
}

int consecutive_integer(int m, int n)
{
   int min = (m < n) ? m : n;
```

```
    while(min != 0)
    {
        if(m % min == 0)
        {
            if(n % min == 0)
                return min;
            else
                min--;
        }
        else
            min--;
    }
    return 1;
}
```

2.Searching

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <algorithm>
using namespace std;

int linearSearchIterative(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

int linearSearchRecursive(int arr[], int key, int index, int n) {
    if (index == n) {
```

```
        return -1;
    }
    if (arr[index] == key) {
        return index;
    }
    return linearSearchRecursive(arr, key, index + 1, n);
}

int binarySearchIterative(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

int binarySearchRecursive(int arr[], int low, int high, int key) {
    if (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            return binarySearchRecursive(arr, mid + 1, high, key);
        } else {
            return binarySearchRecursive(arr, low, mid - 1, key);
        }
```

```
    }
    return -1;
}

int main() {
    srand(time(0)); // Seed for random number generation
    int n, key;
    clock_t start, end;
    double cpu_time_used;

    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n];

    // Generate a random array
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100; // Assuming elements are in these range 0-99
    }
    sort(arr, arr + n); // Sort the array for binary search
    cout <<"the array \n";
    for(int i=0; i<n; i++){
        cout<<arr[i] << " ";
    }

    cout << "\n Enter the element to search: ";
    cin >> key;

    // Linear Search Iterative
    start = clock();
    int resultLinearIterative = linearSearchIterative(arr, n, key);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    cout << "Linear Search (Iterative): ";
```

```
if (resultLinearIterative != -1) {
    cout << "Element found at index: " << resultLinearIterative << endl;
} else {
    cout << "Element not found in the array." << endl;
}
cout << "Time taken for Linear Search (Iterative): " << cpu_time_used << " seconds\n";

// Linear Search Recursive
start = clock();
int resultLinearRecursive = linearSearchRecursive(arr, key, 0, n);
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
cout << "Linear Search (Recursive): ";
if (resultLinearRecursive != -1) {
    cout << "Element found at index: " << resultLinearRecursive << endl;
} else {
    cout << "Element not found in the array." << endl;
}
cout << "Time taken for Linear Search (Recursive): " << cpu_time_used << " seconds\n";

// Binary Search Iterative
start = clock();
int resultBinaryIterative = binarySearchIterative(arr, n, key);
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
cout << "Binary Search (Iterative): ";
if (resultBinaryIterative != -1) {
    cout << "Element found at index: " << resultBinaryIterative << endl;
} else {
    cout << "Element not found in the array." << endl;
}
cout << "Time taken for Binary Search (Iterative): " << cpu_time_used << " seconds\n";
```

```
    // Binary Search Recursive
    start = clock();
    int resultBinaryRecursive = binarySearchRecursive(arr, 0, n - 1, key);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    cout << "Binary Search (Recursive): ";
    if (resultBinaryRecursive != -1) {
        cout << "Element found at index: " << resultBinaryRecursive << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }
    cout << "Time taken for Binary Search (Recursive): " << cpu_time_used << " seconds\n";

    return 0;
}
```

3. MERGE_SORT

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

using namespace std;

// Merge function for strings
void merge(vector<string>& arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    vector<string> L(n1), R(n2);
```

```
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    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(vector<string>& 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(const vector<string>& arr) {
    for (const auto& str : arr)
        cout << str << " ";
    cout << endl;
}

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

    vector<string> arr(n);

    cout << "Enter elements (characters or integers): ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    mergeSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    printArray(arr);
```

```
    return 0;
}
```

4.Quick_sort

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <iomanip>
using namespace std;

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[low];
    int i = low;
    int j = high;
    while (i < j) {
        while (arr[i] <= pivot && i <= high) {
            i++;
        }
        while (arr[j] > pivot && j >= low) {
            j--;
        }
        if (i < j) {
            swap(&arr[i], &arr[j]);
        }
    }
```

```
    swap(&arr[low], &arr[j]);
    return j;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int partitionIndex = partition(arr, low, high);
        quickSort(arr, low, partitionIndex - 1);
        quickSort(arr, partitionIndex + 1, high);
    }
}

int main() {
    int n, l = 1;
    clock_t s,e;
    double t;
    cout << fixed << setprecision(6);
    cout << "\nEnter number of elements in the array : ";
    cin >> n;
    srand(time(0));
    int arr1[1000], arr2[1000];

    for (int i = 0; i < n; i++) {
        arr1[i] = rand() % 100;
    }
    cout << "\nOriginal array : ";
    for (int i = 0; i < n; i++) {
        cout << arr1[i] << "\t";
    }
    s = clock();
    quickSort(arr1, 0, n - 1);
    e= clock();
    cout << "\n\n Sorted array : ";
```

```
    for (int i = 0; i < n; i++) {
        cout << arr1[i] << "\t";
    }
    t = (double)(e - s) / CLOCKS_PER_SEC;
    cout << "\n\nTime taken for Quick Sort with pivot at 1st position is " << t<< "seconds\n\n";

    for (int i = 0; i < n; i++) {
        arr2[i] = rand() % 2;
    }
    cout << "\nOriginal array : ";
    for (int i = 0; i < n; i++) {
        cout << arr2[i] << "\t";
    }
    s = clock();
    quickSort(arr2, 0, n - 1);
    e = clock();
    cout << "\n\n Sorted array : ";
    for (int i = 0; i < n; i++) {
        cout << arr2[i] << "\t";
    }
    t = (double)(e - s) / CLOCKS_PER_SEC;
    cout << "\n\nTime taken for Quick Sort of binary numbers with pivot at 1st position is " << t << "seconds\n\n";
    return 0;
}
```

5. DFS

```
#include <iostream>
#include <vector>
#include <stack>
#include<ctime>
#include<iomanip>

using namespace std;
```

```
class Graph {
public:
    vector<vector<int>> adjList;
    int numVertices;

    Graph(int vertices) : numVertices(vertices), adjList(vertices) {}

    void addEdge(int src, int dest) {
        adjList[src].push_back(dest);

        adjList[dest].push_back(src);
    }
};

void dfs(Graph& graph, int startVertex) {
    vector<bool> visited(graph.numVertices, false);
    stack<int> stk;
    stk.push(startVertex);

    while (!stk.empty()) {
        int currentVertex = stk.top();
        stk.pop();

        if (!visited[currentVertex]) {
            cout << currentVertex << " ";
            visited[currentVertex] = true;
        }

        for (int adjVertex : graph.adjList[currentVertex]) {
            if (!visited[adjVertex]) {
                stk.push(adjVertex);
            }
```

```
        }

    }
    cout << endl;
}

int main() {
    int numVertices, numEdges;
    clock_t s,e;
    double t;
    cout << "Enter the number of vertices: ";
    cin >> numVertices;

    Graph graph(numVertices);

    cout << "Enter the number of edges: ";
    cin >> numEdges;

    cout << "Enter the edges (src dest):\n";
    for (int i = 0; i < numEdges; ++i) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    int startVertex;
    cout << "Enter the starting vertex for DFS: ";
    cin >> startVertex;

    cout << "DFS traversal starting from vertex " << startVertex << ":\n";
     s = clock();
    dfs(graph, startVertex);
     e = clock();
```

```
    t = double(e - s) / CLOCKS_PER_SEC;
    cout << "Time taken for DFS: " << t << " seconds" << endl;
    return 0;
}
```

6. BFS

```
#include <iostream>
#include <vector>
#include <queue>
#include <ctime>

using namespace std;

class Graph {
public:
    vector<vector<int>> adjList;
    int numVertices;

    Graph(int vertices) : numVertices(vertices), adjList(vertices) {}

    void addEdge(int src, int dest) {
        adjList[src].push_back(dest);
        if (src != dest) {
            adjList[dest].push_back(src);
        }
    }
};

void bfs(Graph& graph, int startVertex) {
    vector<bool> visited(graph.numVertices, false);
    queue<int> q;
```

```
    visited[startVertex] = true;
    q.push(startVertex);

    while (!q.empty()) {
        int currentVertex = q.front();
        q.pop();
        cout << currentVertex << " ";

        for (int adjVertex : graph.adjList[currentVertex]) {
            if (!visited[adjVertex]) {
                visited[adjVertex] = true;
                q.push(adjVertex);
            }
        }
    }
    cout << endl;
}

int main() {
    int numVertices, numEdges;
    cout << "Enter the number of vertices: ";
    cin >> numVertices;

    Graph graph(numVertices);

    cout << "Enter the number of edges: ";
    cin >> numEdges;

    cout << "Enter the edges (src dest):\n";
    for (int i = 0; i < numEdges; ++i) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
```

```
    }

    int startVertex;
    cout << "Enter the starting vertex for BFS: ";
    cin >> startVertex;

    cout << "BFS traversal starting from vertex " << startVertex << ":\n";

    clock_t startTime = clock();
    bfs(graph, startVertex);
    clock_t endTime = clock();

    double timeTaken = double(endTime - startTime) / CLOCKS_PER_SEC;
    cout << "Time taken for BFS: " << timeTaken << " seconds" << endl;

    return 0;
}
```

7. TOPOLOGICAL_SORTING

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

using namespace std;

class Graph {
public:
    vector<vector<int>> adjList;
    int numVertices;

    Graph(int vertices) : numVertices(vertices), adjList(vertices) {}
```

```
    void addEdge(int src, int dest) {
        adjList[src].push_back(dest);
    }
};

void dfsUtil(int v, vector<bool>& visited, stack<int>& Stack, const vector<vector<int>>& adjList) {
    visited[v] = true;

    for (int i : adjList[v]) {
        if (!visited[i]) {
            dfsUtil(i, visited, Stack, adjList);
        }
    }

    Stack.push(v);
}

void topologicalSort(Graph& graph) {
    stack<int> Stack;
    vector<bool> visited(graph.numVertices, false);

    for (int i = 0; i < graph.numVertices; i++) {
        if (!visited[i]) {
            dfsUtil(i, visited, Stack, graph.adjList);
        }
    }

    while (!Stack.empty()) {
        cout << Stack.top() << " ";
        Stack.pop();
    }
    cout << endl;
```

```
}

int main() {
    int numVertices, numEdges;
    cout << "Enter the number of vertices: ";
    cin >> numVertices;

    Graph graph(numVertices);

    cout << "Enter the number of edges: ";
    cin >> numEdges;

    cout << "Enter the edges (src dest):\n";
    for (int i = 0; i < numEdges; ++i) {
        int src, dest;
        cin >> src >> dest;
        graph.addEdge(src, dest);
    }

    cout << "Topological Sort of the graph:\n";
    topologicalSort(graph);

    return 0;
}
```

8. PRIM'S

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
```

```
typedef pair<int, int> pii;

void addEdge(vector<vector<pii>>& graph, int u, int v, int weight) {
    graph[u].push_back(make_pair(weight, v));
    graph[v].push_back(make_pair(weight, u));
}

int primMST(vector<vector<pii>>& graph, int V, int startVertex) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    vector<int> key(V, INT_MAX);
    vector<int> parent(V, -1);
    vector<bool> inMST(V, false);

    key[startVertex] = 0;
    pq.push(make_pair(0, startVertex));

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        inMST[u] = true;

        for (auto& i : graph[u]) {
            int v = i.second;
            int weight = i.first;

            if (!inMST[v] && key[v] > weight) {
                key[v] = weight;
                pq.push(make_pair(key[v], v));
                parent[v] = u;
            }
        }
    }
```

```
    int totalCost = 0;
    cout << "Edges in the MST:" << endl;
    for (int i = 0; i < V; ++i) {
        if (parent[i] != -1) {
            cout << parent[i] << " - " << i << " (weight: " << key[i] << ")" << endl;
            totalCost += key[i];
        }
    }

    return totalCost;
}

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the number of edges: ";
    cin >> E;

    vector<vector<pii>> graph(V);

    cout << "Enter edges (u, v, weight):" << endl;
    for (int i = 0; i < E; ++i) {
        int u, v, weight;
        cin >> u >> v >> weight;
        addEdge(graph, u, v, weight);
    }

    int startVertex;
    cout << "Enter the starting vertex (0 to " << V - 1 << "): ";
    cin >> startVertex;
```

```
    int mstCost = primMST(graph, V, startVertex);
    cout << "Total cost of MST: " << mstCost << endl;

    return 0;
}
```

9. DIJKSTRA

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

typedef pair<int, int> pii;

void addEdge(vector<vector<pii>>& graph, int u, int v, int weight) {
    graph[u].push_back(make_pair(weight, v));
    graph[v].push_back(make_pair(weight, u));
}

void printPath(vector<int>& parent, int j) {
    vector<int> path;
    while (j != -1) {
        path.push_back(j);
        j = parent[j];
    }
    for (int i = path.size() - 1; i > 0; --i) {
        cout << path[i] << " -> ";
    }
    cout << path[0];
}
```

```
void dijkstra(vector<vector<pii>>& graph, int V, int startVertex) {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    vector<int> dist(V, INT_MAX);
    vector<int> parent(V, -1);

    dist[startVertex] = 0;
    pq.push(make_pair(0, startVertex));

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (auto& i : graph[u]) {
            int v = i.second;
            int weight = i.first;

            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
                parent[v] = u;
            }
        }
    }

    cout << "Shortest paths from vertex " << startVertex << " to all other vertices:" << endl;
    for (int i = 0; i < V; ++i) {
        if (i != startVertex) {
            cout << "Path to vertex " << i << ": ";
            printPath(parent, i);
            cout << " - Distance: " << dist[i] << endl;
        }
    }
}
```

```
int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the number of edges: ";
    cin >> E;

    vector<vector<pii>> graph(V);

    cout << "Enter edges (u, v, weight):" << endl;
    for (int i = 0; i < E; ++i) {
        int u, v, weight;
        cin >> u >> v >> weight;
        addEdge(graph, u, v, weight);
    }

    int startVertex;
    cout << "Enter the starting vertex (0 to " << V - 1 << "): ";
    cin >> startVertex;

    dijkstra(graph, V, startVertex);

    return 0;
}
```

10. KNAPSACK

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

// Function to solve 0/1 Knapsack problem and track items included
```

```
int knapSack(int W, int wt[], int val[], int n, vector<vector<int>>& dp, vector<int>& parent) {
    // Create a 2D vector to store the maximum value for each subproblem
    dp.resize(n + 1, vector<int>(W + 1, 0));
    parent.resize(n + 1, -1);

    // Build dp table in bottom up manner
    for (int i = 1; i <= n; ++i) {
        for (int w = 1; w <= W; ++w) {
            if (wt[i - 1] > w) {
                dp[i][w] = dp[i - 1][w];
            } else {
                dp[i][w] = max(dp[i - 1][w], val[i - 1] + dp[i - 1][w - wt[i - 1]]);
                if (dp[i][w] == val[i - 1] + dp[i - 1][w - wt[i - 1]]) {
                    parent[i] = w - wt[i - 1];
                }
            }
        }
    }

    // The maximum value will be in dp[n][W]
    return dp[n][W];
}

// Function to print items included in the knapsack
void printItemsIncluded(int W, int wt[], int val[], int n, vector<vector<int>>& dp) {
    // Track items included
    vector<int> itemsIncluded;
    int totalWeight = W;
    for (int i = n; i > 0 && totalWeight > 0; --i) {
        if (dp[i][totalWeight] != dp[i - 1][totalWeight]) {
            itemsIncluded.push_back(i - 1);
            totalWeight -= wt[i - 1];
        }
```

```
    }

    // Print items included
    cout << "Items included in the knapsack:" << endl;
    for (int i = itemsIncluded.size() - 1; i >= 0; --i) {
        cout << "Item " << (itemsIncluded[i] + 1) << " (Value: " << val[itemsIncluded[i]] << ", Weight: " << wt[itemsIncluded[i]]
<< ")" << endl;
    }
}

// Driver code
int main() {
    int n;
    cout << "Enter the number of items: ";
    cin >> n;

    int val[n], wt[n];
    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; ++i) {
        cin >> wt[i];
    }

    cout << "Enter the profits of the items: ";
    for (int i = 0; i < n; ++i) {
        cin >> val[i];
    }

    int W;
    cout << "Enter the weight capacity of the knapsack: ";
    cin >> W;

    vector<vector<int>> dp;
    vector<int> parent;
```

```
    int maxProfit = knapSack(W, wt, val, n, dp, parent);
    cout << "Maximum profit that can be put in the knapsack: " << maxProfit << endl;

    printItemsIncluded(W, wt, val, n, dp);

    return 0;
}
```