**WEEK 4**

**Raahat Arora**

**230957216, Roll no - 72**

QUESTION 1

#include <iostream>

#include <chrono>

#include <cstdlib>

using namespace std;

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

int main() {

const int sizes[] = {1000, 5000, 10000, 50000, 100000};

const int numSizes = 5;

for (int i = 0; i < numSizes; i++) {

int SIZE = sizes[i];

int\* arr = new int[SIZE];

for (int j = 0; j < SIZE; j++) {

arr[j] = rand() % 10000;

}

cout << "Sorting array of size: " << SIZE << endl;

auto start = chrono::high\_resolution\_clock::now();

insertionSort(arr, SIZE);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> duration = end - start;

cout << "Time taken for sorting: " << duration.count() << " ms" << endl;

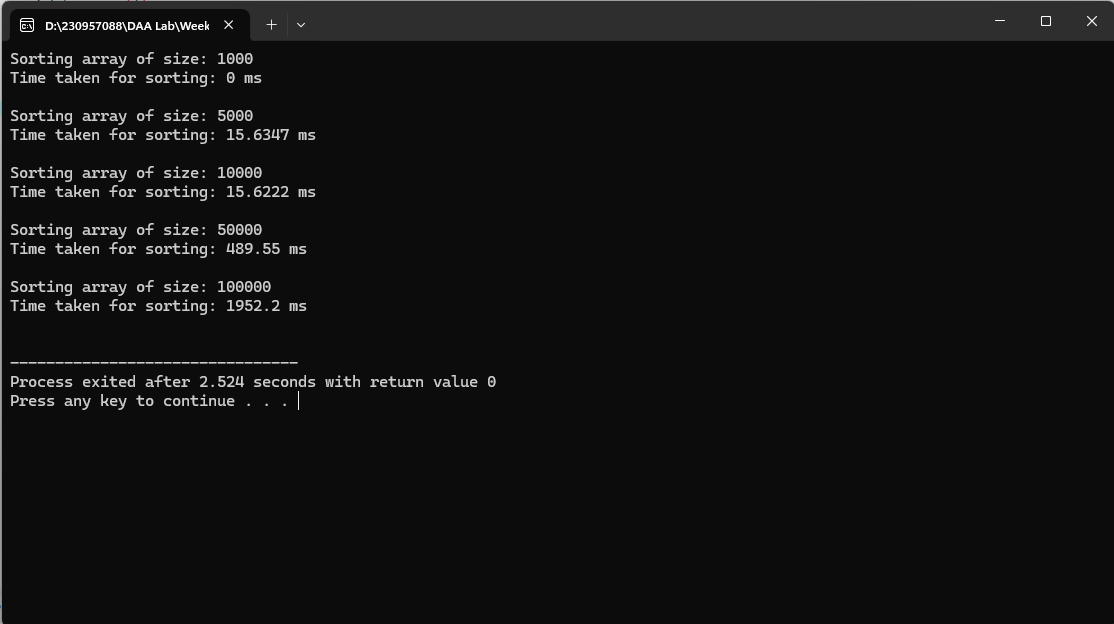
delete[] arr;

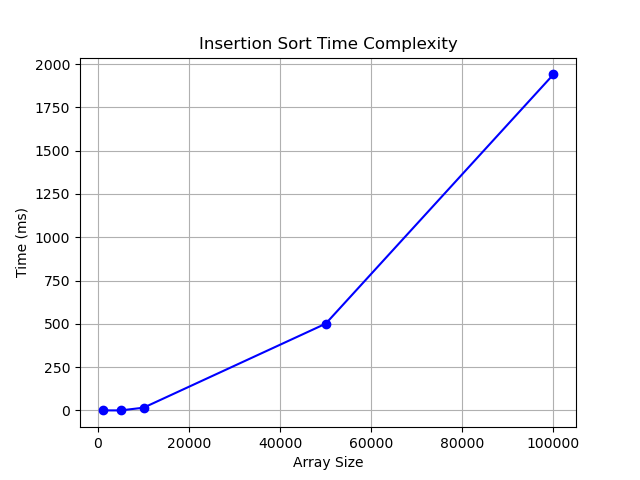
cout << endl;

}

return 0;

}





QUESTION 2

#include <iostream>

#include <chrono>

#include <cstdlib>

using namespace std;

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

if (low < 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]);

int pi = i + 1;

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

const int sizes[] = {1000, 5000, 10000, 50000, 100000};

const int numSizes = 5;

for (int i = 0; i < numSizes; i++) {

int SIZE = sizes[i];

int\* arr = new int[SIZE];

for (int j = 0; j < SIZE; j++) {

arr[j] = rand() % 10000;

}

cout << "Sorting array of size: " << SIZE << endl;

auto start = chrono::high\_resolution\_clock::now();

quickSort(arr, 0, SIZE - 1);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> duration = end - start;

cout << "Time taken for sorting: " << duration.count() << " ms" << endl;

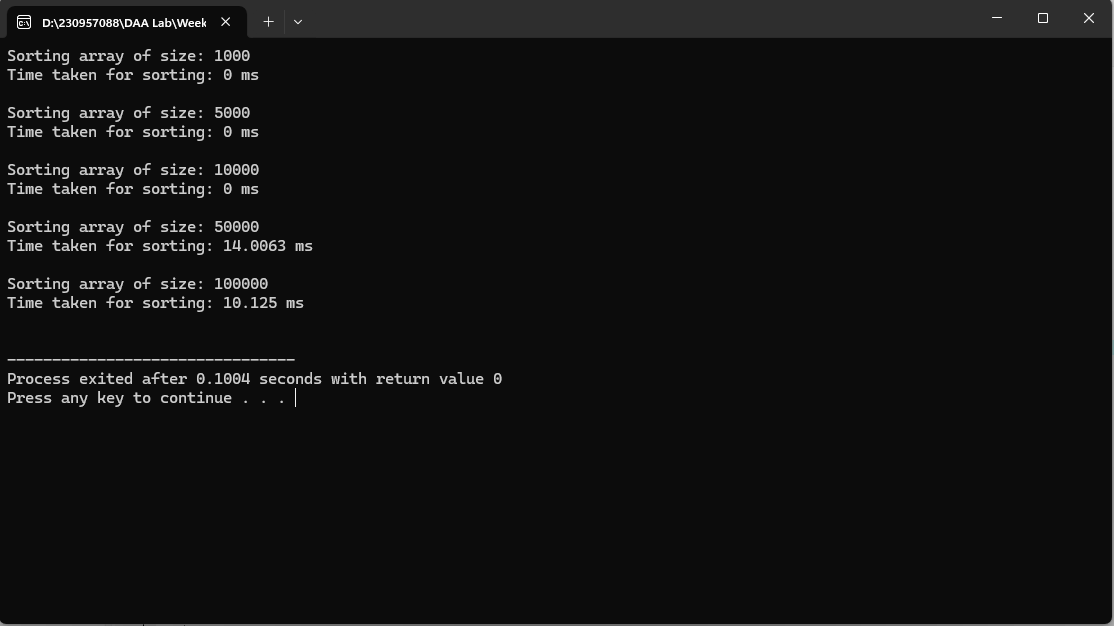
delete[] arr;

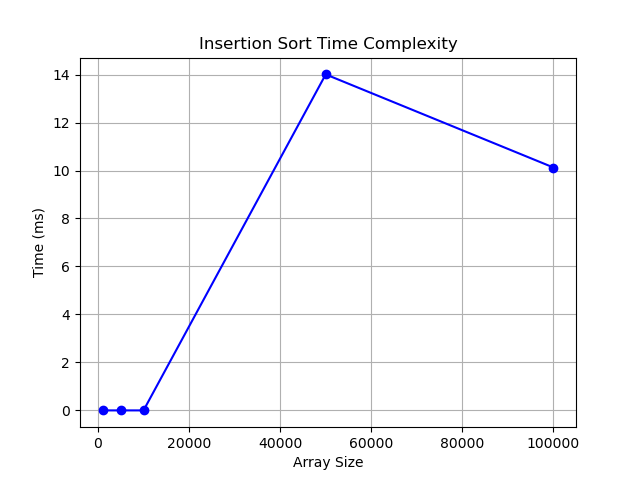
cout << endl;

}

return 0;

}





QUESTION 3

#include <iostream>

#include <chrono>

#include <cstdlib>

using namespace std;

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int\* L = new int[n1];

int\* R = new int[n2];

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

L[i] = arr[left + i];

for (int i = 0; i < n2; i++)

R[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

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++;

}

delete[] L;

delete[] R;

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

const int sizes[] = {1000, 5000, 10000, 50000, 100000};

const int numSizes = 5;

for (int i = 0; i < numSizes; i++) {

int SIZE = sizes[i];

int\* arr = new int[SIZE];

for (int j = 0; j < SIZE; j++) {

arr[j] = rand() % 10000;

}

cout << "Sorting array of size: " << SIZE << endl;

auto start = chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, SIZE - 1);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> duration = end - start;

cout << "Time taken for sorting: " << duration.count() << " ms" << endl;

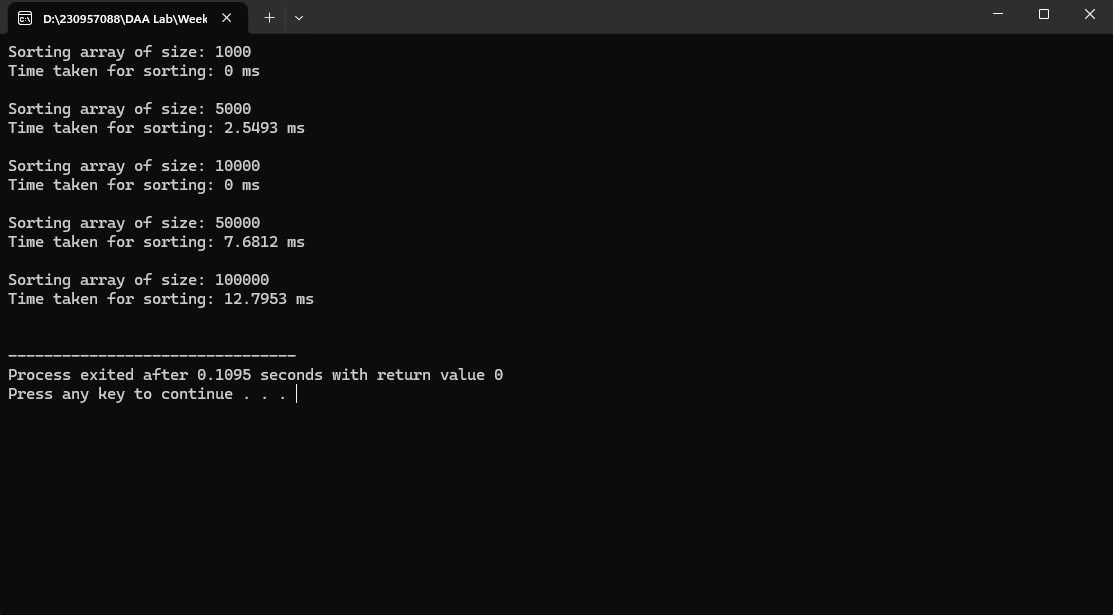
delete[] arr;

cout << endl;

}

return 0;

}



QUESTION 4

#include <iostream>

using namespace std;

class Graph {

public:

int V;

int adj[100][100];

bool visited[100];

Graph(int V) {

this->V = V;

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

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

adj[i][j] = 0;

}

visited[i] = false;

}

}

void addEdge(int u, int v) {

adj[u][v] = 1;

}

void dfsUtil(int v, bool visited[], int topOrder[], int& idx) {

visited[v] = true;

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

if (adj[v][i] == 1 && !visited[i]) {

dfsUtil(i, visited, topOrder, idx);

}

}

topOrder[idx++] = v;

}

void topologicalSortDFS() {

int topOrder[100];

int idx = 0;

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

if (!visited[i]) {

dfsUtil(i, visited, topOrder, idx);

}

}

for (int i = V - 1; i >= 0; i--) {

cout << topOrder[i] << " ";

}

cout << endl;

}

};

int main() {

Graph g(6);

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

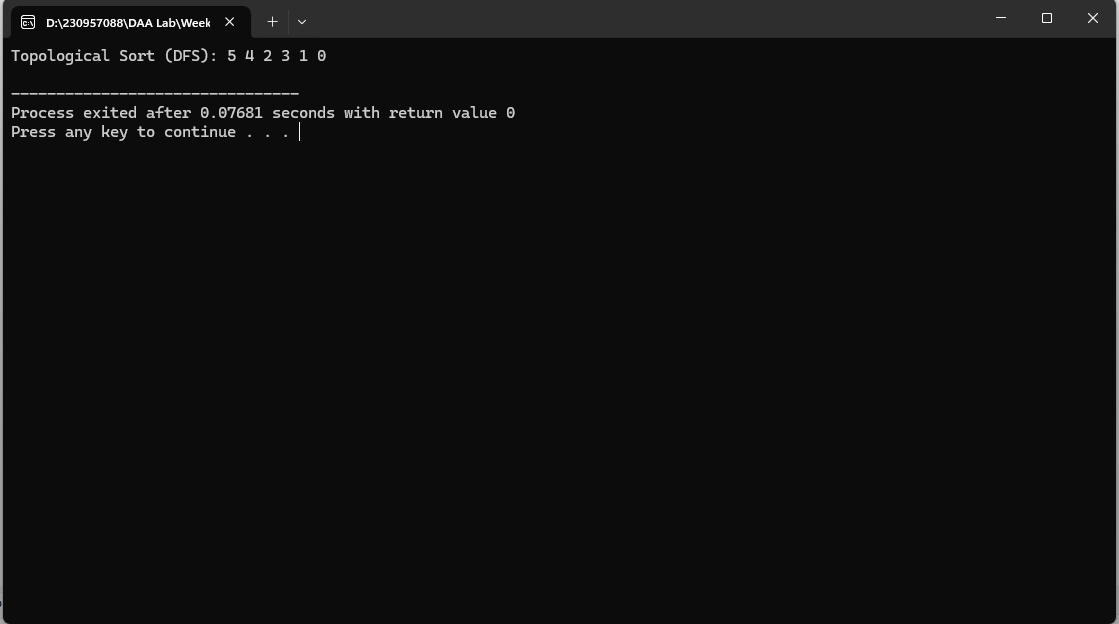
g.addEdge(3, 1);

cout << "Topological Sort (DFS): ";

g.topologicalSortDFS();

return 0;

}



QUESTION 4 – ii

#include <iostream>

using namespace std;

class Graph {

public:

int V;

int adj[100][100]; // Adjacency matrix

int inDegree[100];

Graph(int V) {

this->V = V;

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

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

adj[i][j] = 0; // Initialize adjacency matrix with 0

}

inDegree[i] = 0; // Initialize in-degree array to 0

}

}

void addEdge(int u, int v) {

adj[u][v] = 1; // Mark the edge in the adjacency matrix

inDegree[v]++; // Increment the in-degree of vertex v

}

void topologicalSortSourceRemoval() {

int topOrder[100]; // Array to store topological sort

int idx = 0;

// Repeat the process until all vertices are processed

while (true) {

bool foundSource = false;

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

if (inDegree[i] == 0) {

foundSource = true;

topOrder[idx++] = i;

// Remove the source by decreasing in-degree of its neighbors

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

if (adj[i][j] == 1) {

inDegree[j]--;

}

}

inDegree[i] = -1; // Mark this vertex as processed

break;

}

}

// If no source is found, break the loop (graph might have a cycle)

if (!foundSource) {

break;

}

}

// If not all vertices are processed, there's a cycle

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

if (inDegree[i] != -1) {

cout << "There exists a cycle in the graph!" << endl;

return;

}

}

// Print the topological order

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

cout << topOrder[i] << " ";

}

cout << endl;

}

};

int main() {

Graph g(6);

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

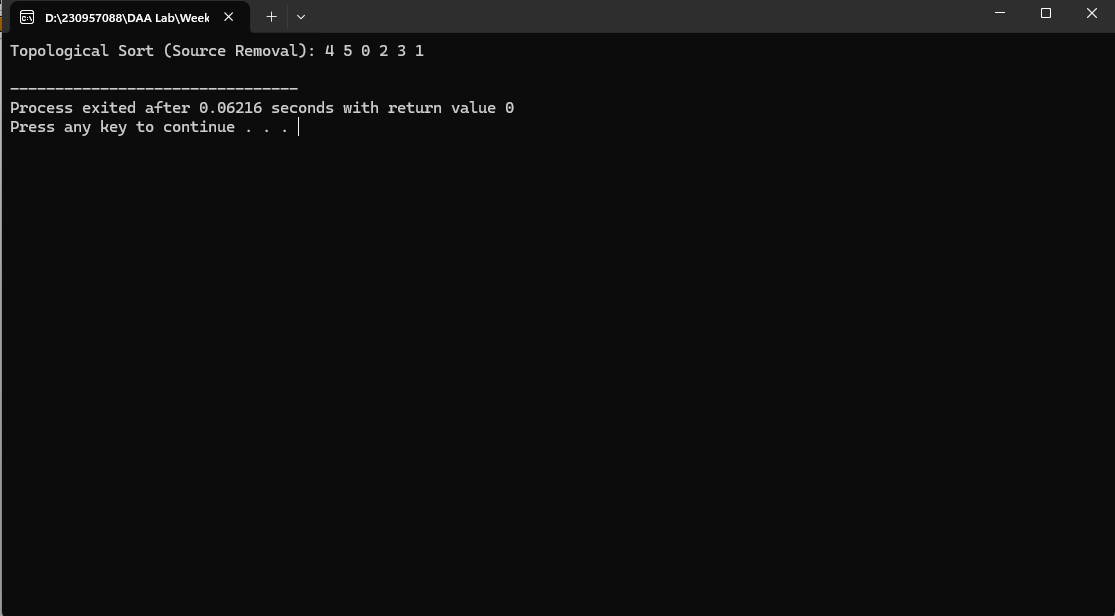
g.addEdge(3, 1);

cout << "Topological Sort (Source Removal): ";

g.topologicalSortSourceRemoval();

return 0;

}



**Binary Search using Divide and Conquer**

**Solution:**

#include <iostream>   
using namespace std;   
   
int binarySearch(int arr[], int left, int right, int key) {   
    if (left > right) return -1;   
   
    int mid = left + (right - left) / 2;   
    if (arr[mid] == key) return mid;   
   
    if (arr[mid] > key) return binarySearch(arr, left, mid - 1, key);   
    return binarySearch(arr, mid + 1, right, key);   
}   
   
int main() {   
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};   
    int n = sizeof(arr) / sizeof(arr[0]);   
    int key = 5;   
   
    int result = binarySearch(arr, 0, n - 1, key);   
    cout << "Element found at index: " << result << endl;   
   
    return 0;   
}

**Output:**

Element found at index: 4

**Finding a^n using Divide and Conquer**

**Solution:**

#include <iostream>   
using namespace std;   
   
int power(int a, int n) {   
    if (n == 0) return 1;   
    int temp = power(a, n / 2);   
    if (n % 2 == 0) return temp \* temp;   
    else return a \* temp \* temp;   
}   
   
int main() {   
    int a = 2, n = 5;   
    cout << a << "^" << n << " = " << power(a, n) << endl;   
    return 0;   
}

**Output:**

2^5 = 32

**Student Management System Scheduling using Decrease and Conquer**

**Solution:**

#include <iostream>   
#include <vector>   
#include <queue>   
   
using namespace std;   
   
void scheduleTasks(int V, vector<vector<int>>& dependencies) {   
    vector<int> in\_degree(V, 0);   
    for (auto& dep : dependencies) {   
        for (int task : dep) {   
            in\_degree[task]++;   
        }   
    }   
   
    queue<int> q;   
    for (int i = 0; i < V; i++) {   
        if (in\_degree[i] == 0) {   
            q.push(i);   
        }   
    }   
   
    while (!q.empty()) {   
        int task = q.front();   
        q.pop();   
        cout << "Execute Task: " << task << endl;   
   
        for (int dependent : dependencies[task]) {   
            if (--in\_degree[dependent] == 0) {   
                q.push(dependent);   
            }   
        }   
    }   
}   
   
int main() {   
    int V = 5;   
    vector<vector<int>> dependencies(V);   
    dependencies[0] = {2, 3};   
    dependencies[1] = {3, 4};   
    dependencies[2] = {4};   
    dependencies[3] = {};   
    dependencies[4] = {};   
   
    cout << "Task Execution Order:" << endl;   
    scheduleTasks(V, dependencies);   
   
    return 0;   
}

**Output:**

Task Execution Order:   
Execute Task: 0   
Execute Task: 1   
Execute Task: 2   
Execute Task: 3   
Execute Task: 4