**Name: Sai Gulve**

**PRN No: 202201040145**

**Batch: A4**

**Roll No: 63**

**Problem Statement:**

Given **N activities**, each defined by a **start time** and an **end time**, the objective is to **select the maximum number of activities** that can be performed by a single person, assuming that a person can only work on **one activity at a time**. Implement an efficient **Greedy Algorithm** to solve this problem and analyze its execution time.

**Introduction:**

The **Activity Selection Problem** is a fundamental **scheduling problem** where we aim to **maximize the number of non-overlapping activities** that can be performed within a given schedule.

This problem finds applications in various domains, including:

* **Task Scheduling** in operating systems
* **Conference Room Allocation**
* **Project Management**
* **Event Scheduling**

A **Greedy Algorithm** provides an **optimal solution** to this problem by always selecting the **activity that finishes the earliest** while ensuring that it does not overlap with previously selected activities.

**Approach & Algorithm:**

The algorithm follows these steps:

1. **Sort activities** in **ascending order of their end times**.
2. **Select the first activity** (since it has the earliest finish time).
3. **For each subsequent activity**, select it **only if it starts after or when the previous activity ends**.
4. **Continue this process** until all activities are considered.
5. **Measure the execution time** of the algorithm using the chrono library.

**Time Complexity Analysis:**

1. **Sorting Activities** (Bubble Sort) → **O(n²)**
2. **Activity Selection** → **O(n)**
3. **Overall Complexity** → **O(n²)** (Can be improved to **O(n log n)**

Code :- #include <bits/stdc++.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Activity {

int start, finish;

};

// Partition function for QuickSort (sorting by finish time)

int partition(Activity arr[], int low, int high) {

Activity pivot = arr[high]; // Choosing the last element as the pivot

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j].finish < pivot.finish) {

i++;

swap(arr[i], arr[j]);

}

}

swap(arr[i + 1], arr[high]);

return i + 1;

}

// QuickSort function

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Function to print maximum number of non-overlapping activities

void printMaxActivities(Activity arr[], int n) {

quickSort(arr, 0, n - 1); // Sorting activities based on finish time

cout << "Following activities are selected:\n";

int i = 0;

cout << "(" << arr[i].start << ", " << arr[i].finish << ")";

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

if (arr[j].start >= arr[i].finish) {

cout << ", (" << arr[j].start << ", " << arr[j].finish << ")";

i = j;

}

}

cout << endl;

}

int main() {

int n;

cout << "Enter number of activities: ";

cin >> n;

Activity arr[n];

cout << "Enter start and finish times of activities:\n";

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

cin >> arr[i].start >> arr[i].finish;

}

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

printMaxActivities(arr, n);

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

auto duration = duration\_cast<microseconds>(end - start);

cout << "Execution time: " << duration.count() << " microseconds\n";

return 0;

}