|  |  |
| --- | --- |
| **Activity Selection in C++** | |
| #include <iostream>  #include <algorithm>  #include <vector>  using namespace std;  class Activity {  public:  int start;  int finish;    Activity(int s, int f) {  start = s;  finish = f;  }  };  struct MyCmp {  bool operator()(const Activity& a1, const Activity& a2) const {  return a1.finish < a2.finish;  }  };  int maxActivity(vector<Activity>& arr) {  sort(arr.begin(), arr.end(), MyCmp());  int res = 1;  int prev = 0;  for (int curr = 1; curr < arr.size(); curr++) {  if (arr[curr].start >= arr[prev].finish) {  res++;  prev = curr;  }  }  return res;  }  int main() {  vector<Activity> arr = {Activity(12, 25), Activity(10, 20), Activity(20, 30)};  cout << maxActivity(arr) << endl;  return 0;  } | Activity Selection Problem Summary: Given n activities with start and finish times, select the maximum number of activities that **don’t overlap** and **finish earliest** (greedy approach). 📋 Input Activities (Before Sorting):  | **Index** | **Start** | **Finish** | | --- | --- | --- | | 0 | 12 | 25 | | 1 | 10 | 20 | | 2 | 20 | 30 |  🔀 Step 1: Sort by Finish Time Using the comparator:  return a1.finish < a2.finish; 🧾 After Sorting:  | **Index** | **Start** | **Finish** | | --- | --- | --- | | 1 | 10 | 20 | | 0 | 12 | 25 | | 2 | 20 | 30 |   Sorted vector:  [ {10,20}, {12,25}, {20,30} ] 🧮 Step 2: Activity Selection (Greedy) We initialize:   * res = 1 (we pick the first activity) * prev = 0 (index of the last selected activity)   Now we iterate from curr = 1 to n-1. ➤ Iteration Table:  | **curr** | **Activity (start, finish)** | **prev** | **arr[curr].start >= arr[prev].finish** | **Action** | **res** | **prev** | | --- | --- | --- | --- | --- | --- | --- | | 1 | (12, 25) | 0 | 12 >= 20 → ❌ False | Skip | 1 | 0 | | 2 | (20, 30) | 0 | 20 >= 20 → ✅ True | Select this activity | 2 | 2 |  ✅ Final Result:  * Maximum activities: **2** * Selected activities:   + {10, 20}   + {20, 30}  🖨 Output: 2 |
| 2 | |