Design and Analysis of Algorithms

Assignment 2

By Himanshu Sardana

Question 1

Given the input: `start[] = {1, 3, 0, 5, 8, 5}`, `finish[] = {2, 4, 6, 7, 9, 9}` using activity selection. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
        vector<int> start = {1, 3, 0, 5, 8, 5};
        vector<int> finish = {2, 4, 6, 7, 9, 9};

        int n = start.size();

        vector<pair<int, int>> activities;

        for (int i = 0; i < n; i++) {
                activities.push_back({finish[i], start[i]});
        }

        sort(activities.begin(), activities.end());

        int count = 1;
        int i = 0;

        for (int j = 1; j < n; j++) {
                if (activities[j].second >= activities[i].first) {
```

```
                        count++;
                        i = j;
                }
        }

        cout << count << endl;
}
```

Output:

```
4
```

Question 2

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline.

Job ID	Deadline	Profit
a	4	20
b	1	10
c	1	40
d	1	30

Maximize the total profit if only one job can be scheduled at a time.

Output:

```
Total profit: 60
```

Question 3

Given the weights and profits of N items in the form of {profit, weight}, Input: `arr[] = {{60, 10}, {100, 20}, {120, 30}}` **, put these items in a knapsack of capacity W = 50 to get the maximum total profit in the knpacksack. Use Fractional Knapsack, and breaks items for maximizing the total value of the knapsack.**

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
        vector<pair<int, int>> items = {{60, 10}, {100, 20}, {120, 30}};
        int capacity = 50;

        sort(items.begin(), items.end(), [](pair<int, int> a, pair<int, int> b) {
                return (double)a.first / a.second > (double)b.first / b.second;
        });

        double profit = 0;
        for (auto item : items) {
                if (capacity == 0) {
                        break;
                }

                int weight = min(capacity, item.second);
                profit += (double)weight * item.first / item.second;
                capacity -= weight;
        }

        cout << profit << endl;
}
```

Output:

```
240
```