Lab 3: Implementation of greedy algorithms (Fractional Knapsack and Job Sequencing with Deadlines)

Objective:

To implement greedy algorithm to solve fractional knapsack and job sequencing with deadline problem.

Theory:

1. Greedy Algorithm: Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum solution. In these algorithms, decisions are made based on the information available at the current moment without considering the consequences of these decisions in the future. The key idea is to select the best possible choice at each step, leading to a solution tat may not always be the most optimal but is often good enough for many problems.
   Steps:
   a. Define the problem: Clearly state the problem to be solved and the objective to be optimized.
   b. Identify the greedy choice: Determine to locally optimal choice at each step based on the current state.
   c. Mate the greedy choice: Select the greedy choice and update the current state.
   d. Repeat: Continue making greedy choice until a solution is reached.

2. Application of Greedy Algorithm:
   a. Fractional Knapsack problem: The knapsack problem states that "given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum."
      Algorithm :
      i. Consider all the items with their weights and profits mentioned respectively.
      ii. Calculate P_i/W_i of all the items and sort the items in descending order based on their P_i/W_i values.
      iii. Without exceeding the limit, add the items into the knapsack.
      iv. If the knapsack can still store some weight, but the weights of other items exceed the limit, the fractional part of the next item is added.

b. Job Sequencing with deadlines: Job sequencing algorithm is applied to schedule jobs on a single processor the maximize the profits.
c. The greedy approach of the job scheduling algorithm states that, “Given ‘n’ number of jobs with a starting time and ending time, they need to scheduled in such a way that maximum profit is received within the deadline.”

Algorithm :

i. Arrang the jobs in descending order of profits.
ii. Select jobs with highest profits without exceeding the deadline.
iii. The selected jobs are the output.

Observation:

1. Fractional Knapsack Problem:

```
#include <iostream>
#include <vector>
#include <gettime.h>
#include <algorithm>

using namespace std;

float sequencer(vector<pair<int, int>> inputs, int capacity)
{
    sort(inputs.begin(), inputs.end(), [](pair<int, int> a, pair<int, int> b) ->
bool
         { return a.first / b.second > a.first / b.second; });

    float res = 0;

    for (pair<int, int> input : inputs)
    {

        if (input.second <= capacity)
        {
            res += input.first;
            capacity -= input.second;
        }
        else
        {
            res += input.first * ((float)capacity / (float)input.second);
            break;
        }
    }

    return res;
}

int main()
{
    vector<int> sizes;
    int start_size = 1000000;
    int increment = 500000;
    sizes.push_back(start_size);
    for (size_t i = 0; i < 4; i++)
    {
        sizes.push_back(sizes.back() + increment);
```

```
    }
    srand(time(0));
    for (int size : sizes)
    {

        vector<pair<int, int>> arr;

        for (int i = 0; i < size; i++)
        {
            arr.push_back({rand(), rand()});
        }

        long long time = getTime([&]
                                 { sequencer(arr, rand()); });

        cout << "Size: " << size << " Time: " << time << endl;
    }

    return 0;
}
```

Output:

```
Size: 1000000 Time: 384010100
Size: 1500000 Time: 622231400
Size: 2000000 Time: 832764000
Size: 2500000 Time: 960262300
Size: 3000000 Time: 1134811900
```

2. Job Sequencing with Deadline:

```
#include <iostream>
#include <vector>
#include <gettime.h>
#include <algorithm>

using namespace std;

class Job
{
public:
    string id;
    int deadline;
    int profit;
};
```

```
vector<Job> sequencer(vector<Job> inputs)
{
    sort(inputs.begin(), inputs.end(), [](Job a, Job b) -> bool
         { return a.profit > b.profit; });

    int n = inputs.size();
    vector<bool> slot(n - 1, false);
    vector<Job> res(n - 1, Job());
    for (Job input : inputs)
    {
        for (int j = min(n, input.deadline) - 1; j >= 0; j--)
        {
            if (slot[j] == false)
            {
                res[j] = input;
                slot[j] = true;
                break;
            }
        }
    }

    return res;
}

int main()
{
    vector<int> sizes;
    int start_size = 1000000;
    int increment = 500000;
    sizes.push_back(start_size);
    for (size_t i = 0; i < 4; i++)
    {
        sizes.push_back(sizes.back() + increment);
    }
    srand(time(0));
    for (int size : sizes)
    {

        vector<Job> arr;
        string id = "a";

        for (int i = 0; i < size; i++)
        {
            Job job;
```

```
            job.id = id;
            job.deadline = rand() % 10 + 1;
            job.profit = rand() % 100 + 1;
            arr.push_back(job);
        }

        long long time = getTime([&]
                                 { sequencer(arr); });

        cout << "Size: " << size << " Time: " << time << endl;
    }

    return 0;
}
```

Output:

```
Size: 1000000 Time: 1209338100
Size: 1500000 Time: 1679701400
Size: 2000000 Time: 2755124500
Size: 2500000 Time: 2434707000
Size: 3000000 Time: 3305479200
```

Conclusion:

Greedy solutions for Fractional Knapsack and Job Sequencing with Deadline was implemented with C++.