ADA LAB 3

Fractional Knapsack Problem:

```
#include <iostream>

using namespace std;

struct Item {
    int value;
    int weight;
};

// Function to swap two items
void swap(Item &a, Item &b) {
    Item temp = a;
    a = b;
    b = temp;
}

// Function to sort items by value-to-weight ratio
void sortItems(Item items[], int n) {
    for (int i = 0; i < n-1; ++i) {
        for (int j = 0; j < n-i-1; ++j) {
            double r1 = (double)items[j].value / items[j].weight;
            double r2 = (double)items[j+1].value / items[j+1].weight;
            if (r1 < r2) {
                swap(items[j], items[j+1]);
            }
        }
    }
}

double fractionalKnapsack(int W, Item items[], int n) {
    // Sort items by value-to-weight ratio
    sortItems(items, n);

    double totalValue = 0.0;

    for (int i = 0; i < n; ++i) {
        if (W == 0) break; // If the knapsack is full, break

        if (items[i].weight <= W) {
            // If the item can be fully added
            W -= items[i].weight;
            totalValue += items[i].value;
        } else {
            // If the item can be partially added
            totalValue += items[i].value * ((double)W / items[i].weight);
            W = 0;
        }
    }

    return totalValue;
}

int main() {
    int W = 50; // Total weight capacity of the knapsack
    Item items[] = {{60, 10}, {100, 20}, {120, 30}}; // Items with value and weight
    int n = sizeof(items) / sizeof(items[0]);

    double maxValue = fractionalKnapsack(W, items, n);
    cout << "Maximum value in Knapsack = " << maxValue << endl;

    return 0;
}
```

Job Sequencing Problem:

```
#include <iostream>

using namespace std;

struct Job {
    char id;      // Job Id
    int deadline; // Deadline of job
    int profit;   // Profit if job is completed before or on deadline
};

// Function to swap two jobs
void swap(Job &a, Job &b) {
    Job temp = a;
    a = b;
    b = temp;
}

// Function to sort jobs by profit in descending order
void sortJobs(Job arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j].profit < arr[j+1].profit) {
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

void jobSequencing(Job arr[], int n) {
    // Sort the jobs by profit
    sortJobs(arr, n);

    int result[n]; // To store result (sequence of jobs)
    bool slot[n];  // To keep track of free time slots

    // Initialize all slots to be free
    for (int i = 0; i < n; i++)
        slot[i] = false;

    // Iterate through all given jobs
    for (int i = 0; i < n; i++) {
        // Find a free slot for this job (start from the last possible slot)
        for (int j = min(n, arr[i].deadline) - 1; j >= 0; j--) {
            // Free slot found
            if (!slot[j]) {
                result[j] = i;  // Add this job to result
                slot[j] = true; // Mark this slot as occupied
                break;
            }
        }
    }

    // Print the result
    cout << "Following is the maximum profit sequence of jobs:\n";
    for (int i = 0; i < n; i++)
        if (slot[i])
            cout << arr[result[i]].id << " ";
    cout << endl;
}

int main() {
    Job arr[] = { {'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}, {'d', 1, 25}, {'e', 3, 15} };
    int n = sizeof(arr) / sizeof(arr[0]);

    jobSequencing(arr, n);
    return 0;
}
```