Branch and Bound

Knapsack Problem

Q5.
Click submit
Q7.

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int n, int wmax, vector<pair<int,int>>& items) {
    // Create a DP array
    vector<vector<int>> dp(n + 1, vector<int>(wmax + 1, 0));

    // Fill DP table
    for (int i = 1; i <= n; ++i) {
        int weight = items[i-1].first;
        int value = items[i-1].second;
        for (int w = 1; w <= wmax; ++w) {
            if (weight <= w) {
                dp[i][w] = max(dp[i-1][w], dp[i-1][w-weight] + value);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }

    // Return the maximum value achievable
    return dp[n][wmax];
}

int main() {
    int n, wmax;
    cin >> n >> wmax;

    vector<pair<int,int>> items(n);
    for (int i = 0; i < n; ++i) {
        cin >> items[i].first >> items[i].second;
    }

    int maxValue = knapsack(n, wmax, items);
    cout << maxValue << endl;

    return 0;
}
```

Traveling Salesman Problem

Q6.

```
upper_bound = infinity
candidate_queue
candidate_queue.push({1, adjacency_matrix})
while (queue not empty):
   best_solution = get_best_solution(candidate_queue)
   if (best_solution.city = 1) : upper_bound = best_solution.cost
   lower_bound = get_lower_bound(best_solution)
   if (lower_bound >= upper_bound) : continue
   candidate_queue.expand(best_solution)
return upper_bound
```

Assignment problem

Q4.
38

Overview of P, NP and NP-Complete Problems
Q7.
Problem is not necessarily P
Problem is always NP
Problem is always NP-Hard