Name: Tran Ngoc Son (陳玉山)
ID: 0710185

Homework 5 report

I. Introduction

- In this homework, we will use the dynamic programming method to solve the rod-cut problem. This means that we will try to find out the maximum value can achieve with a certain amount of rod length.

II. Methodology

- There are two methods that we can reference to solve this problem, top-down and bottom-up methods.

1. Top-down

```
Cut-Rod(p, n)
    if n = 0
        return 0
    else q = −∞
        for i = 1 to n
            q = max(q, p[i] + Cut-Rod(p, n − i))
    return q
```

Figure 1:Top-down method pseudo code

For the top-down method, we will pass in the array p, which contains the price for the small segments of the rod and the length n. The code would recursively call itself and pass in the smaller value of n until it reaches the base case, n = 0. Each time of being called, Cut-Rod function will return the best value that we can get from selling the rod of length n.

2. Memoized method

- Although we can refer to the above pseudo-code to solve the problem, the running time would cost $T(2^n)$ and it's not sufficient when the value of n becomes large. Therefore, we can apply the method called memorized, which can help to reduce the running time to $T(n^2)$. This method is using an array r[] to save the return value after calling the cut-rod function. Therefore, we can reuse the result without re-calculating it when meeting the same cases as previous.

```
Memoized-Cut-Rod-Aux(p, n, r)
    if r[n] ≥ 0
        return r[n]
    else q = −∞
        for i = 1 to n
            q = max(q, p[i] + Memoized-Cut-Rod-Aux(p, n − i, r)
        r[n] = q
        return q
```

Figure 2: Apply Memoized method

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

```cpp
int cut_rod(int p[], int n,int* memoized)
{   int q = INT16_MIN;


    if(memoized[n]>0)
    {
        return memoized[n];
    }

    if (n == 0)
    {
        return 0;
    }
    else
    {

        for (int i=1; i<=n;i++)
        {
            q = max(q,p[i]+cut_rod(p,n-i,memoized));

        }
        // cout << "n = " << n;
    }
    memoized[n] = q;
    return q;
}
```

Figure 3: The code of top-down method with memoization.

3. Bottom-up method

- The running time of this method is roughly the same as the top-down method with applying memoization, which is O(n^2). However, it doesn't have to go through all possible cases using recursive.

Bottom-up-Cut-Rod(p, n)

```
Let r[0...n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

Figure 4: Pseudo code of bottom-up method.

Name: Tran Ngoc Son (陳玉山)
ID: 0710185

```cpp
int bottom_up_cut_rod(int p[],int n,int r[])
{


    r[0] = 0;
    cout << n;
    for (int j=1;j<=n;j++)
    {
        int q = INT16_MIN;
        for (int i = 1;i<=j;i++)
        {
            q = max(q,p[i]+r[j-i]);
        }
        cout << "q = " << q << endl;
        r[j] = q;
    }
    return r[n];
}
```

Figure 5: The code of bottom-up method with memoization.

III. Discussing

- If we want to find the minimum price for the given rod length n. I think we can use the same method as above, however, we need to change the initial value of q to infinity. In addition, instead of picking the maximum between q and q[i] + r[j-i], we rather pick the minimum of it. The changing parts are included in the code below.

```cpp
int bottom_up_cut_rod(int p[],int n,int r[])
{


    r[0] = 0;
    cout << n;
    for (int j=1;j<=n;j++)
    {
        int q = INT16_MAX;
        for (int i = 1;i<=j;i++)
        {
            q = min(q,p[i]+r[j-i]);
        }
        cout << "q = " << q << endl;
        r[j] = q;
    }
    return r[n];
}
```

Figure 6: The changing in code to achieve the minimum value of the rod of length n.