7205 HW6

Name: Xuebao Zhao NUID: 002108354

Q1:

In this problem, I used the main function which was already given. Wrote two approaches: FibonacciR and FibonacciD.
FibonacciR solves the problem recursively where it calls itself recursively exactly twice. FibonacciD solves the problem recursively but with the support of dynamic programming.

FibonacciD:

```
int FibonacciD(int n)
{
    if (n < 2) return n;
    if (A[n] != -1) return A[n];
    A[n] = FibonacciD(n - 1) + FibonacciD(n - 2);
    return A[n];
}
```

FibonacciR:

```
int FibonacciR(int n)
{
    if (n < 2) return n;
    else return FibonacciR(n - 1) + FibonacciR(n - 2);
}
```

a) Comment on the running time of each function.

Using time_start, time_end of type DWORD to record the time. And using the time_start, time_end to calculate the running time.

```
clock_t time;
cout << "Using dp Methord:" << endl;
time = clock();
//Find Fibonacci sequence n using the Dynamic programming function
cout << " FibonacciD(n) = " << FibonacciD(n) << endl;
time = clock() - time;
cout << "This way costs: " << ((float)time / CLOCKS_PER_SEC) *1000 << " ms" << endl << endl;
```

Test the code with n values 5, 10, 20, 30, 40:

```
n = 5
Using dp Methord:
 FibonacciD(n) = 5
This way costs: 1 ms

Using recursion Methord:
 FibonacciR(n) = 5
This way costs: 0 ms
```

```
n = 10
Using dp Methord:
 FibonacciD(n) = 55
This way costs: 1 ms

Using recursion Methord:
 FibonacciR(n) = 55
This way costs: 0 ms
```

```
n = 20
Using dp Methord:
 FibonacciD(n) = 6765
This way costs: 0 ms

Using recursion Methord:
 FibonacciR(n) = 6765
This way costs: 17 ms
```

```
n = 30
Using dp Methord:
FibonacciD(n) = 832040
This way costs: 0 ms

Using recursion Methord:
FibonacciR(n) = 832040
This way costs: 96 ms
```

```
n = 40
Using dp Methord:
FibonacciD(n) = 102334155
This way costs: 0 ms

Using recursion Methord:
FibonacciR(n) = 102334155
This way costs: 6515 ms
```

The results show that when n is less than 20, there is no big difference between recursive method and dynamic programming method. But when n is not smaller than 20, the speed of recursive method drops significantly because recursive method needs to calculate the same value multiple times.

b) For each function, find the big O asymptotic notation of its running time growth rate.

FibonacciD:

$$T(n) = \begin{cases} T(n-1) + C & n \geq 2 \\ C & n < 2 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + C \\ &= T(n-2) + 2C \\ &= T(n-3) + 3C \\ &= T(1) + (n-1)C \\ &= nC \Rightarrow O(n) \end{aligned}$$

FibonacciR:

$$\begin{cases} T(n-1) + T(n-2) & n \geq 2 \\ C \cdot n & n < 2 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) \\ &= T(n-2) + T(n-3) + T(n-3) + T(n-4) \\ &= \cdots \\ &= 2^{n-1} \cdot nC = 2^n C \Rightarrow O(2^n) \end{aligned}$$

Q2:

Recursive method:

```
// Recurive method
int Recursive_CutRod(int* p, int n)
{
  if (n == 0)
    return 0;
  int q = -1; // Initializes to a minimum
  for (int i = 1; i <= n; i++)
    q = max(q, p[i] + Recursive_CutRod(p, n - i)); // Find the max q
  return q;
}
```

Dynamic Programming method:

```
// Dynamic Program
int Memoized_CutRod_Aux(int* p, int n, int* r)
{
  int q;
  if (r[n] >= 0) return r[n]; // r[n] has been calculated
  if (n == 0) q = 0;
  else{
    q = -1;
    for (int i = 1; i <= n; i++) // For each case
      q = max(q, p[i] + Memoized_CutRod_Aux(p, n - i, r)); // Find the max q
  }

  r[n] = q;
  return q;
}

int Memoized_CutRod(int* p, int n)
{
  int* r = new int[n + 1];
  for (int i = 0; i <= n; i++) r[i] = -1; // Initializes to a minimum
  return Memoized_CutRod_Aux(p, n, r);
}
```

Store price:

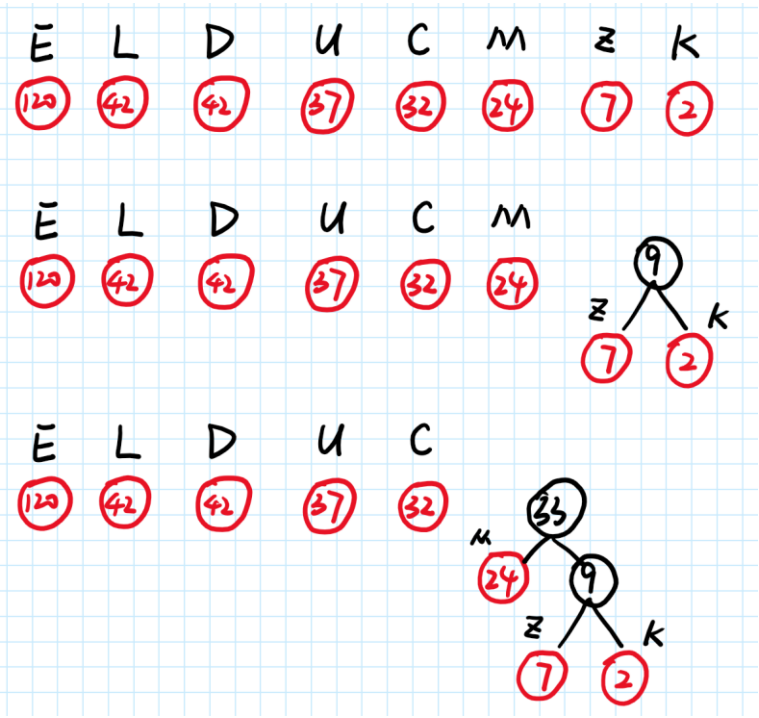
```
// Define the prices
p[0] = n;
for (int i = 1; i <= n; i++) {
  if (i == 1) p[i] = 2;
  else if (i == n) p[i] = (float)n * 2.5 - 1;
  else p[i] = floor(i * 2.5);
}
```

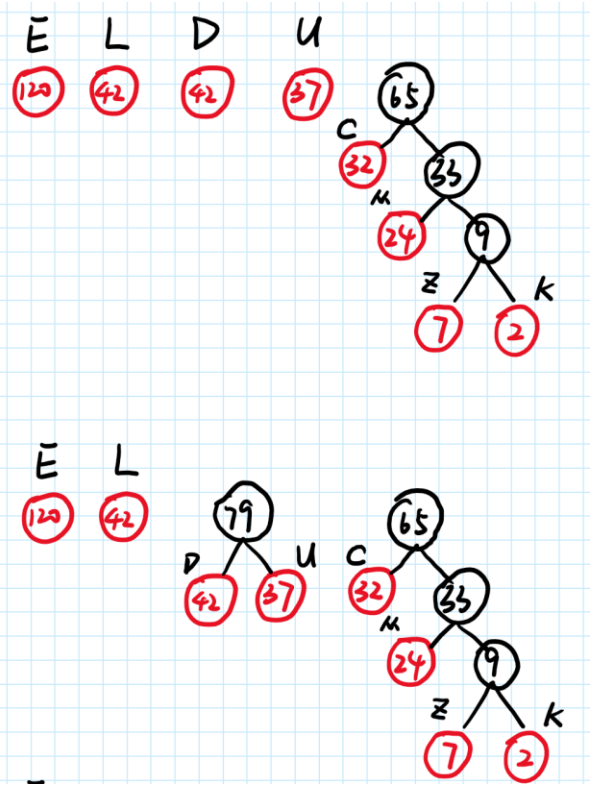
Result:

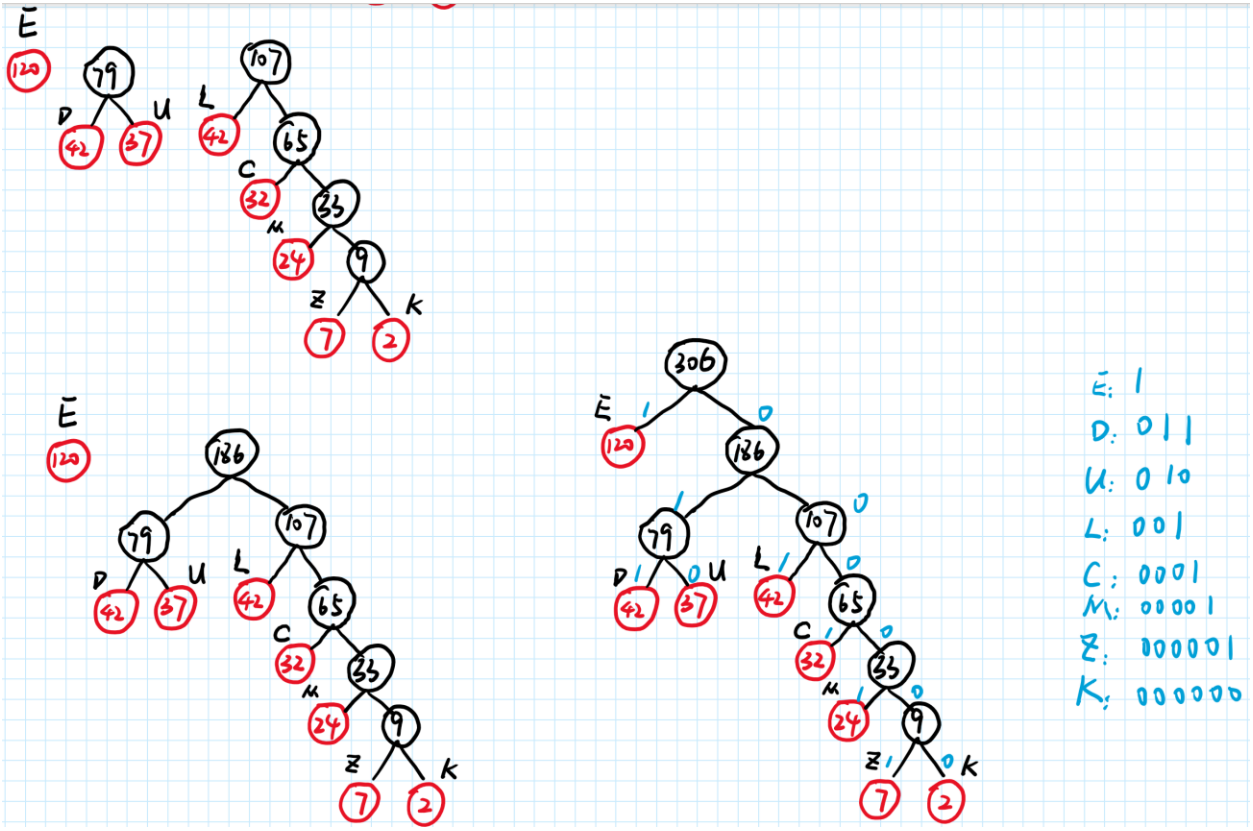
Rod Size	Recursive Time	Recursive Max Revenue	Dynamic Time	Dynamic Max Revenue
5	0	12	0	12
10	0	25	0	25
15	0	37	0	37
20	10000	50	0	50
25	410000	62	0	62
30	1.307e+07	75	0	75
35	No solution	No solution	0	87
40	No solution	No solution	0	100
45	No solution	No solution	0	112
50	No solution	No solution	0	125

Form the table we can see that the speed of dynamic programming is always zero while the speed of recursive method is increasing rapidly.

Q3:




E L D U
120 42 42 37
65
C 32 33
M 24 9
Z 7 K 2
E L
120 42
79
D 42 U 37
65
C 32 33
M 24 9
Z 7 K 2


E
120
79
D 42 U 37
107
L 42 65
C 32 33
M 24 9
Z 7 K 2
E
120
186
79
D 42 U 37
107
L 42 65
C 32 33
M 24 9
Z 7 K 2
306
E 1 120 0 186
1 79 0 107
D 1 42 0 U 37
L 1 42 0 65
C 1 32 0 33
M 1 24 0 9
Z 1 7 0 K 2
E: 1
D: 011
U: 010
L: 001
C: 0001
M: 00001
Z: 000001
K: 000000