chapter 10

a. the worst case is that we search all the lists. For each list with the length of 2^i and are sorted, we can seach it in $O(i)$, because we have i varies from 0 to lgn , the worst time cost is $O(lg^2n)$

b. In the worst case, we will merge $n_0, n_1, , , , n_{k-1}$ to a_k, for each merge the time cost is linear, therefore the total time cost is $O(n)$

we can use accounting method, we set the cost of each insertion is lgn. But only the elements in the Kth list will cost lgn in the time cost. Therefore the total time cost is smaller than $nlgn$, so the time cost of each insertion is $O(lgn)$

chapter 11

```
class Solution:
    def minDistance(self, word1, word2):
        """
        :type word1: str
        :type word2: str
        :rtype: int
        """
        n = len(word1)
        m = len(word2)

        # 有一个字符串为空串
        if n * m == 0:
            return n + m

        # DP 数组
        D = [ [0] * (m + 1) for _ in range(n + 1)]

        # 边界状态初始化
        for i in range(n + 1):
            D[i][0] = i
        for j in range(m + 1):
            D[0][j] = j

        # 计算所有 DP 值
        for i in range(1, n + 1):
            for j in range(1, m + 1):
                left = D[i - 1][j] + 1
                down = D[i][j - 1] + 1
                left_down = D[i - 1][j - 1]
```

```
                if word1[i - 1] != word2[j - 1]:
                    left_down += 1
                D[i][j] = min(left, down, left_down)

        return D[n][m]
```

chapter 12

a. Always give the highest denomination coin that you can without going over. Then, repeat this process until the amount of remaining change drops to 00.

b. Given an optimal solution $(x_0, x_1, \ldots, x_k)$ where x_i indicates the number of coins of denomination c_i . We will first show that we must have $x_i < c$ for every $i < k$. Suppose that we had some $x_i \geq c$, then, we could decrease $x_i$by c and increase x_{i+1} by 1. This collection of coins has the same value and has $c - 1$ fewer coins, so the original solution must of been non-optimal. This configuration of coins is exactly the same as you would get if you kept greedily picking the largest coin possible. This is because to get a total value of V, you would pick $x_k = \lfloor Vc^{-k} \rfloor$ and for $i < k$, $x_i \lfloor (V \mod c^{i+1}) c^{-i} \rfloor$. This is the only solution that satisfies the property that there aren't more than cc of any but the largest denomination because the coin amounts are a base c representation of V mod c^k.

c. Let the coin denominations be {1, 3, 4}, and the value to make change for be 6. The greedy solution would result in the collection of coins {1,1,4} but the optimal solution would be {3, 3}.

d. See algorithm $\text{MAKE-CHANGE}(S, v)$ which does a dynamic programming solution. Since the first for loop runs *n* times, and the inner for loop runs *k* times, and the later while loop runs at most *n* times, the total running time is O(nk).

chapter 13

$Assume$:

$$\begin{aligned} x_1 &= over \quad produce \quad in \quad q1; \\ x_2 &= over \quad produce \quad in \quad q2; \\ x_3 &= over \quad produce \quad in \quad q3; \end{aligned}$$

We get those restrictions:

$$\begin{gathered} x_1 \leq 10 \\ 30 - x_1 + x_2 \leq 40 \\ 20 - x_2 + x_3 \leq 20 \\ x_1, x_2, x_3 \geq 0 \end{gathered}$$

Cost:

$$w = 1176 + 1.2x_1 - 1.1x_2 + 0.7x_3$$

$maxmize$:

$$w1 = -w$$

$subject - to$:

$$
\begin{aligned}
x_4 &= 10 - x_1 \\
x_5 &= 10 + x_1 - x_2 \\
x_6 &= x_2 - x_3 \\
x_1, x_2, x_3, x_4, x_5, x_6 &\geq 0
\end{aligned}
$$

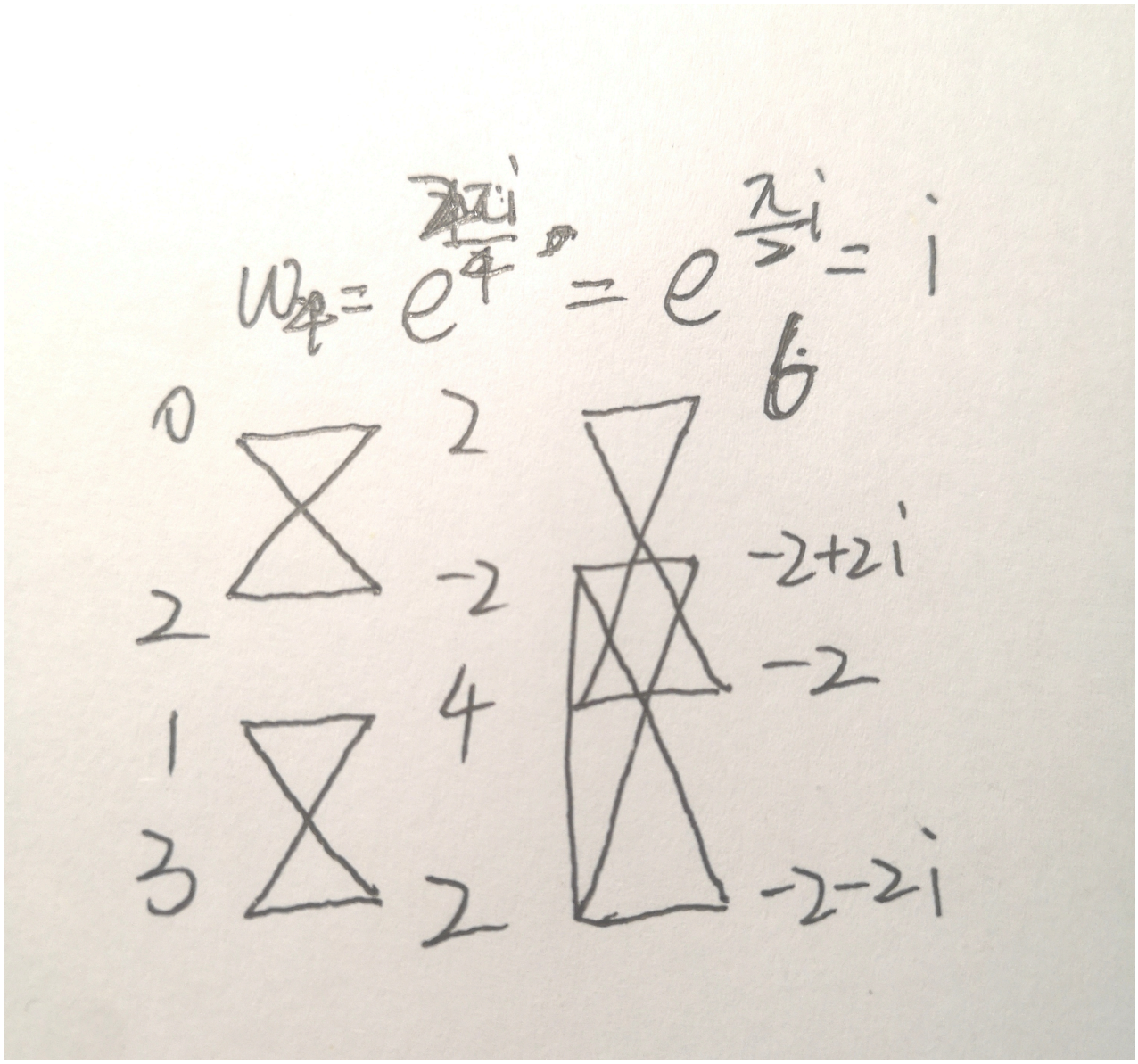
Then we can rewrite the linear program:

$$
\begin{aligned}
W &= -1165 - 0.1x_1 - 0.7x_3 - 1.1x_5 \\
x_4 &= 10 - x_1 \\
x_2 &= 10 + x_1 - x_5 \\
x_6 &= 10 + x_1 - x_3 - x_5
\end{aligned}
$$

We can get the answer:

$$
\begin{aligned}
prodoction_{quarter-1} &= 20 \\
prodoction_{quarter-2} &= 40 \\
prodoction_{quarter-3} &= 10 \\
prodoction_{quarter-4} &= 10 \\
Cost = 20 * 15 + 40 * 14 + 20 * 0.2 + 10 * 15.3 + 10 * 14.8 &= 1165
\end{aligned}
$$

chapter 14



chapter 15

hamiltonian-path problem is to find if there exists a path to travel all the point once in the map, while the

HAM-CYCLE problem is to find a loop to travel all the point once in the map.

if we find a loop which is the solution of HAM-CYCLE problem such as $(a_1, a_2, a_3 \ldots, a_n, a1)$, the $(a_1, a_2, a_3, \ldots, a_n)$ is the solution of hamiltonian-path problem in the map.

because HAM-CYCLE problem is npc, so hamiltonian-path problem is npc too.