Exercise 1.

a. 由 Dijkstra 算法可求解单源最短路径问题

伪代码：

```
void Dijkstra (Vertex s)        //s代表源顶点
{
  while (1) {                                    O(n)
  Do{ v= mindist (V未添加)  //未添加到S中的所有顶点中dist
      if V不存在                  最小的顶点
       then break;
      S[V] = true;
      for (V的每个邻接点W)
       { if (S[W] == false && dist[V]+ E(v,w) < dist[W])
            Then dist[W] = dist[V] + E<v, w>;        O(n)
                 path[W] = V; }
```

算法复杂性分析：

空间复杂性：运用3个大小为 n 的数组，故空间复杂度为 O(n).

时间复杂性：循环进行 n 次，每次循环时间复杂度 T(n) = O(n)

故总的时间复杂度 T(n) = O(n²)

b. 设A是单源最短路径的一个优化解，其中其有最小路径长的点为 Vi. 则 B = A - {dist[Vi]} 为去掉 Vi 点外所有点所构成的优化解.

证：假设存在较B而言的更优解 B'

则令 A' = B' + {dist[Vi]}.

则 A' 为较 A 更好的优化解，这与 A 为最优解矛盾

故原命题得证。

引理1：

C、设 $V_n=\{V_1, V_2, \cdots, V_n\}$ 为初始时所有点按dist升序排列后所构成的点集。即 $dist[V_1] \le dist[V_2] \cdots \le dist[V_n]$。则必存在一个优化解，使得从特定点 V 到各点的最短路径长度中包含 $dist[V_1]$。

证：当 V 到 $V_1$ 的最短路径即为 $VV_1$ 时，显然成立。

当 V 到 $V_1$ 的最短路径非 $VV_1$ 时，设为 $V, V_i \cdots V_j V_1$ 则其为 V 到 $V_1$ 的最短路径。此时 $D\langle V, V_1\rangle = dist[V_i] + E\langle V_i, V_k\rangle + \cdots + E\langle V_j, V\rangle$，且 $dist[V_i] \ge dist[V_1]$ 故 $D\langle V, V_1\rangle \ge dist[V_1]$。故令 $B=A-\{D\langle V, V_1\rangle\}+\{dist[V_1]\}$，则B也是问题的一个优化解。

原命题得证。

□

引理2：设 $V_i=\{V_j \in V_i \mid V_i = V_{i-1} - S_{i-1}\}$ 中最有最小 dist 的点的下标，设 A 是包含 dist[V] 的优化解，则 $A=\bigcup_{i=1}^{n}\{dist[V_i]\}$。

对 |A| 作归纳法

当 |A|=1 时，由引理1，命题成立

设 |A|=n-1 时，命题成立。

则当 |A|=n 时，由 $A=\{V_1=1\} \cup A_1$

$A_1$ 是 $V_2=\{V_j \in V_2 \mid V_2 = V_{i-1} - S_1\}$ 的优化解

由归纳假设 $A_1=\bigcup_{i=2}^{n}\{dist[V_i]\}$。

于是 $A=\bigcup_{i=1}^{n}\{dist[V_i]\}$。

由引理2知贪心选择方法具有 Greedy 选择性。

Exercise 2.

a. 算法：
按25美分、10美分、5美分、1美分的顺序重复循环遍历。
若当前币值小于剩余应找值，则将该币选入结果中，重新开始遍历，直至剩余应找值为0为止。

证明最优解：
首先证明该问题具有优化子结构：即若$A_n$为该问题的优化解，$A_{n-1}=A_n-e$（e为本次选择的硬币）亦为优化解。
证：设存在更优解$A'_{n-1}$使得$A'_{n-1}$所选硬币数$|A'_{n-1}|<|A_{n-1}|$。
则存在$A'_n=A_{n-1}+e$使得$|A'_n|<|A_n|$
这与$A_n$是该问题优化解矛盾。
故原命题得证。

引理1：首次选择的硬币$e_1$必被某个优化解所包含。
证：设A是一个优化解，则
① A包含$e_1$时，命题成立
② A不包含$e_1$时，则设A的第一个选择为$e'_1$。
由贪心算法知，$e_1\geq e'_1$，令$B=A-\{e'_1\}+\{e_1\}$。
且在币值为25美分、10美分、5美分、1美分的情况中。
必有B中硬币数$|B|\leq|A|$。
由于A是优化解故$|A|$不$\leq|B|$故$|A|=|B|$。
B是一个选择是$e_1$的优化解。

引理2：设$e_i$是第i次选择的硬币，设A是包含$e_1$的优化解
则$A=\bigcup_{i=1}^{k}\{e_i\}$。
证：对$|A|$作归纳法

$|A|=1$时，由引理1，命题成立。
设$|A|\leq k-1$时，命题成立
当$|A|=k$时，由优化子结构$A=\{e_1\}\cup A_1$
$A_1$是剩余应找值$S_1=S-V(e_1)$问题的优化解。
由归纳假设$A_1=\bigcup_{i=2}^{k}\{e_i\}$
得$A=\bigcup_{i=1}^{k}\{e_i\}$。

由引理2知贪心方法具有Greedy选择性。

b. 设最优解为 $(x_0, x_1, \cdots, x_k)$，其中 $x_i$ 指的是币值为 $C^i$ 的硬币的数量

显然，对于任意 $i<k$，$x_i<C$

否则，可将 $x_i - C$，$x_{i+1}+1$ 得到一个硬币数更少的解，与其为最优解相矛盾

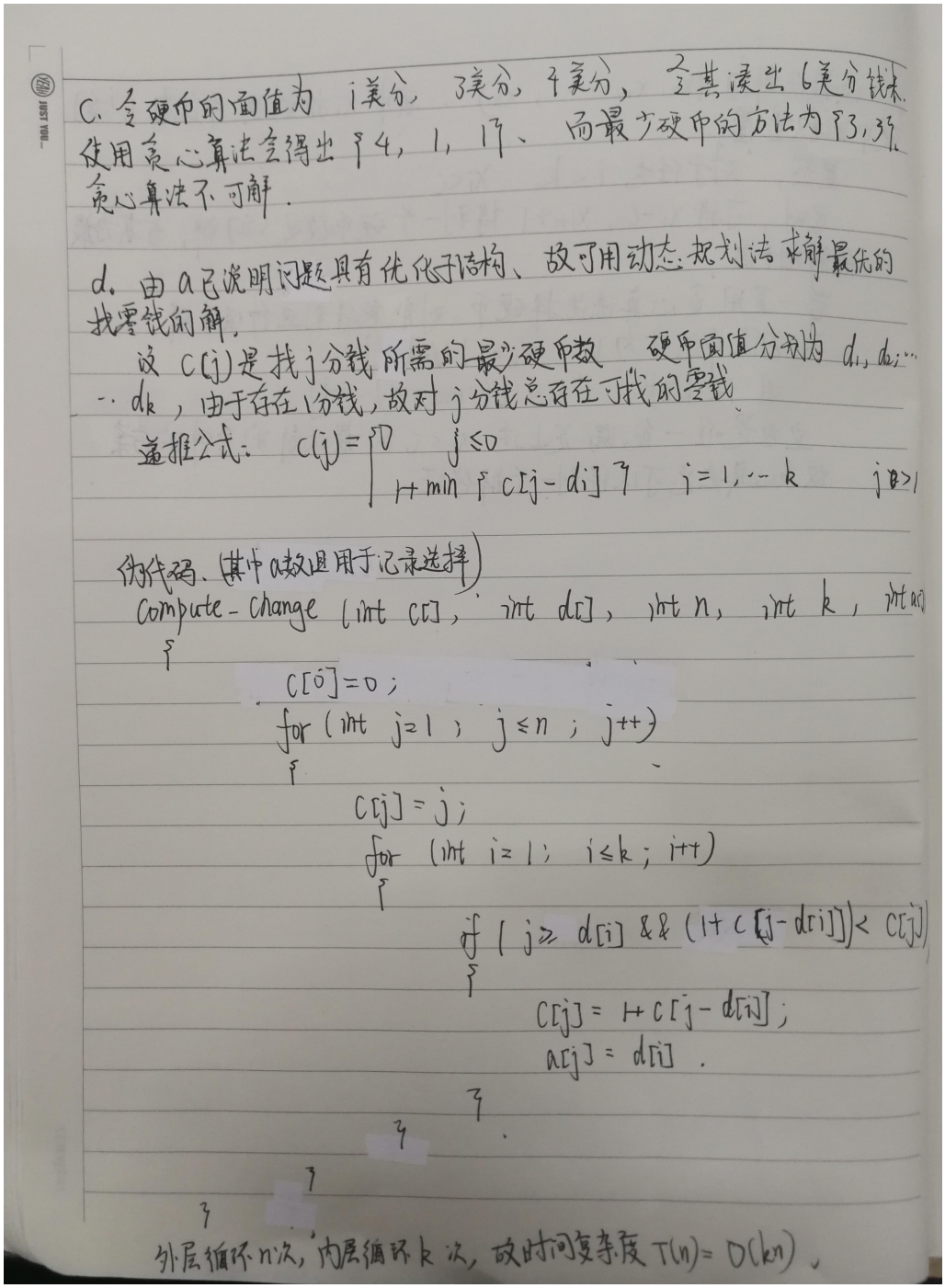
若一直用贪心算法选择硬币，则必能得到这种组合解.

∵ 为使币总值为 $V$，$x_k = \lfloor V \cdot C^{-k} \rfloor$

且 $i<k$ 时，$x_i = \lfloor (V \bmod C^{i+1}) \cdot C^{-i} \rfloor$

这也是唯一能满足上述 $x_i < C$ 的最优解的条件的方案

故贪心算法总可以得到一个解得证.




## Exercise 3

思路：读入两行数字，存入两数组中并按升序排序。而后利用贪心算法，用尽量小的饼干满足孩子。

```
public class Biscuit_allocation
{
    public static void main(String[] args)
    {
        //读入两行数字，存为两个数组并按升序排序。
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Input:");
        String g = sc.nextLine();
        String s = sc.nextLine();
        List<Integer> glist = new ArrayList<>();
        List<Integer> slist = new ArrayList<>();
        int i , j;
        String[] gstrarray = g.split(" ");
        String[] sstrarray = s.split(" ");
        for(i = 0; i < gstrarray.length; i++)
        {
            glist.add(Integer.valueOf(gstrarray[i]));
        }
        for(i = 0; i < sstrarray.length; i++)
        {
            slist.add(Integer.valueOf(sstrarray[i]));
        }
        Collections.sort(glist);
        Collections.sort(slist);

        // 贪心算法
        // 用尽量小的饼干满足孩子
        i = 0;
        j = 0;
        while(i < glist.size() && j < slist.size())
        {
            if(glist.get(i) <= slist.get(j))
            {
                i++;
            }
            j++;
        }
        System.out.println("Output:");
        System.out.println(i);
    }

}
```

## Exercise 4

（1）

思路：

利用贪心算法的思想，先把每个孩子的糖初始化为 1，经过两轮扫描：

第一轮：保证后面比前面权值高的孩子，必定多得 1 个糖果；
第二轮：保证前面比后面权值高的孩子，必定多得 1 个糖果。

```
  public class Candy
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Input:");
        String g = sc.nextLine();
        String[] gstr = g.split("\\[|\\]|,");
        int i , sum = 0;
        List<Integer> A = new ArrayList<>();
        List<Integer> Candy = new ArrayList<>();
        for(i = 1 ; i < gstr.length ; i++)
        {
            A.add(Integer.valueOf(gstr[i]));
        }
        //每个孩子先分配一颗糖果
        for(i = 0 ; i < A.size() ; i++)
        {
            Candy.add(1);
        }
        //从前往后扫描，如满足相邻两个小孩后面的权重大于前面的权重的情况，后面的小孩在前面的小孩的糖果数的基础上加一个。  O(n)
        for(i = 0 ; i < A.size() ; i++)
        {
            if(i >= 1)
            {
                if(A.get(i) > A.get(i-1))
                {
                    Candy.set(i, Candy.get(i-1)+1);
                }
            }

        }
        //从后往前扫描，如满足相邻两个小孩前面的权重大于前面的权重的情况。  O(n)
        //这样两遍扫描下来就可以保证权重高的孩子比相邻权重低的孩子的糖果多。
        for(i = A.size() - 1 ; i >= 0  ; i--)
        {
            if(i <= A.size() - 2)
            {
                if(A.get(i) > A.get(i+1))
                {
```

```
                    Candy.set(i, Candy.get(i+1)+1);
                }
            }

        }
        //求和  O(n)
        for(i = 0 ; i < A.size() ; i++)
        {
            System.out.println(Candy.get(i));
            sum += Candy.get(i);
        }
        System.out.println("Output:");
        System.out.println(sum);

    }
}
```

（2）

时间复杂度：  如代码注释中分析，T(n) = O(n)

空间复杂度：  使用两个大小为 n 的数组，故为 O(n)