HW5

- 姓名：吴欣怡
- 学号：PB21051111

Q1

采用回溯算法：因为这里的value部分只规定了下限，而

```
def knapsack_subset(n, w, v, W, epsilon):
    dp = [[0] * ((1 + epsilon) * W + 1) for _ in range(n +
1)]
    choices = [[False] * ((1 + epsilon) * W + 1) for _ in
range(n + 1)]

    for i in range(1, n + 1):
        for j in range((1 + epsilon) * W + 1):
            if w[i-1] <= j:
                if dp[i-1][j] < dp[i-1][j-w[i-1]] + v[i-1]:
                    dp[i][j] = dp[i-1][j-w[i-1]] + v[i-1]
                    choices[i][j] = True
                else:
                    dp[i][j] = dp[i-1][j]
            else:
                dp[i][j] = dp[i-1][j]

    # 回溯找到解
```

```python
    # 回溯找到解
    selected_items = []
    i, j = n, int((1 + epsilon) * W)
    while i > 0 and j > 0:
        if choices[i][j]:
            selected_items.append(i-1)
            j -= w[i-1]
        i -= 1

    return selected_items
n = 4
w = [5, 4, 1, 6]
v = [3, 6, 4, 11]
W = 10
epsilon = 0.1

result = knapsack_subset(n, w, v, W, epsilon)
print("选中的物品: ", result)
```

Q2

因为A[1]和A[n]不相等，所以一定能够找到所求的索引i
采用递归搜索，每次对搜索的范围对半分，如果这个mid是我们所求的索引，那么就直接返回mid的索引;
如果不是，则继续在左右半边搜索，并且返回其中返回值为正值的部分。

```
function findNonEqualIndex(A, low, high):
    #low和high重合，说明所求索引不在当前这部分，返回负值
    if low == high:
        return -1

    mid = (low + high) // 2

    if A[mid] == A[mid + 1]:
        # 如果中间元素与下一个元素相等，则在左右两侧都进行递归搜
索
        left_result = findNonEqualIndex(A, low, mid)
        right_result = findNonEqualIndex(A, mid + 1, high)

        # 返回左右两边的正值：若均为负数，则直接返回-1，说明所求
值不在当前这个大半边；
        若其中一个为整数，说明找到了所求索引，直接返回这个值；若两
个都为整数，那么选择返回左边的值（只要求返回一个满足条件的就行）
        if left_result>0
            return left_result
        else if right_result>0
            return right_result
        else
            return -1
    else:
        # 如果中间元素与下一个元素不相等，则中间元素就是非相等元
素
        return mid
findNonEqualIndex(A, 1, n)
```

每次减半递归，初始问题是对规模为n的数组，递归部分的操作次数为O(log_2n)，后面比较left_result和right_result是O(log_2n)的倍数量级，所以整个算法的复杂度为O(log(n))

Q3

基础事件是H0的情况：H_0 的情况对应的H_0 v=v(此时n=0)
每次求解时，把当前的v值拆成左右两部分v_1和v_2，分别对应$H_{n-1}v_1 + H_{n-1}v_2$和$H_{n-1}v_1 - H_{n-1}v_2$两个部分

```
function split(v):
    k = length(v)

    if k == 1:
        return v  // 基本事件
    n=k/2
    v1 = v[:n]
    v2 = v[n:]

    // 递归
    H1 = split(v1)
    H2 = split(v2)

    result = zeros(k)

    for i = 0 to k/2 - 1:
        result[i] = H1[i] + H2[i]
        result[i + k] = H1[i] - H2[i]

    return result
```

原始数据的规模是 2^n ，所以递归的部分深度为O($log_2 2^n$)=O(n)，然后每一层递归后求新的result，循环从0到k/2 - 1，这一部分的最大不超过(2*2^{n-1}=2^n)。综上，算法总体复杂度为O(n*2^n)

Q4

(1)

设一共有p道题
给每个同学赋一个权重$W_{i,t}$表示同学i在第t题后、第（t+1）题前时的权重。 定义：
y_t表示第y题的正确答案。 $J_{t,good} = i|f_{i,t} = y_t$也就是对第t题，预测正确的同学的集合 $J_{t,bad} = i|f_{i,t}! = y_t$也就是对第t题，预测错误的同学的集合
$W_t = \sum_{i=1}^{n} W_{i,t}$，即第t题时所有同学的权重之和
W_0=n*1=n 为初始总权重。
$W_{t,J}$，对第t题，集合J中同学的权重之和
首先证明一个结论：$W_{t+1} \leq (1+\beta)/2W_t$
因为对于第t题，若是小牛采用了大家加权权重更大的结果，但是却出错了，说明$W_{t,J_{t,good}}$占W_t的比例小于0.5，相应的 $W_{t,J_{t,bad}}$占W_t的比例大于0.5。
$W_{t+1} = W_{t,J_{t,good}} + \beta * [W_t - W_{t,J_{t,good}}]$
$= (1-\beta) * W_{t,J_{t,good}} + \beta * W_t$
$\leq (1+\beta)/2 * W_t$
据此结论我们有：$W_p \leq n * [(1+\beta)/2]^{loss_p}$
其中$loss_p$表示p次答题之后小牛犯错误的总次数。
又因为W_p是p题后所有同学的权重求和，一定大于对小牛帮助最大的同学的权限。而此同学p题后的权重最小为β^m 因此有$\beta^m \leq n * [(1+\beta)/2]^{loss_p}$，因此$loss_p \geq log_{(1+\beta)/2}(\beta^m/n)$ 代入$\beta = 0.5$可得：$loss_p$至多不超过[1/(2-$log_2 3)] * (m + log_2 n)$

(2)

设一共有p道题，（按照做题的顺序）其中第t道题的错误选项权重为F_t，X表示总的错题数，$E(X)=\sum_{t=1}^{p}F_t$.

考虑所有同学的权重之和为W，在结束所有答题后的新权重为$W^{'}$。用W_k表示结束的k题的答题后的新权重总和。易知：

W_0=n*1=n 为初始总权重。

在第t题后的$W_k=(1-(1-\beta)F_t)W_{t-1}$

其中$(1-\beta)F_t$是错误后损失的权重

$$W^{'}=W_0\prod_{t=1}^{p}(1-(1-\beta)F_t)$$

则有：$=n\prod_{t=1}^{p}(1-(1-\beta)F_t)$ （1）式

因为W是所有人的权重之和，必然大于任意一个人的单人权重值，而对小牛帮助最大的人的权重值在结束所有答题后最小为：β^m，即所有ta无法答对的题目都选择了错误答案，原始权重1乘m次β。

故有：$W^{'}\geq\beta^m$ 又根据（1）式两边取对数得到

$m*ln\beta\leq lnn+\sum_{t=1}^{p}ln(1-(1-\beta)F_t)$

由$ln(1+x)=\sum_{k=1}(-1)_{k-1}*x_k/k$

得到：$ln(1-(1-\beta)F_t)=\sum_{k\geq1}(-1)*[(1-\beta)F_t]/k<-(1-\beta)F_t$ 代入有：

$m*ln\beta\leq lnn-\sum_{t=1}^{p}(1-\beta)F_t=lnn-(1-\beta)E[X]$

于是有：

$E[X]\geq(lnn-mln\beta)/(1-\beta)=(m*ln(1/\beta)+lnn)/(1-\beta)$