16.1-2

这个方法与书上的方法相比就是反过来罢了。假设我们选了一个活动，我们想让剩下的活动能选得最多，意思就是我要让剩下的时间越多，反过来想就是本次选择的活动时间要越少。如果所有活动的结束时间按递增排好序，那么从左到右扫描一遍，等同于所有活动的开始时间按递减排好序，从左到右扫描一遍，伪代码如下

GREEDY-ACTIVITY-SELECTOR（S，f）
1. n ← length[S]
2. A ← $\{a_1\}$
3. k ← 1
4. for m ← 2 up to n
5. if s[m] ≤ f[k]
6.   A ← A ∪ $\{a_m\}$
7.   k ← m
8. return A

此算法总是考虑当下最优的选择，不考虑将来的效应，而且与书上的经典算法针对同一个集合总是求得相同的解，所以求得的是最优解。

16.2-6

1. 求出价质比(v_i/w_i)
2. 将价质比的中位数作为主元，小于这个数的价质比放在 L 数组中，等于的放在 E 数组中，

大于的放在 G 数组中，求出每个数组的总重$W_L, W_E 和 W_G$

3. $\begin{cases} if\ W_G > W，令 G 作为新集合，执行 2,3 \\ if\ W_G \le W \le W_G + W_E，则把 G 集合的都拿了，E 集合的尽量多拿，算法结束 \\ if\ W_G + W_E < W，把 G 和 E 中的全拿，令 L 作为新集合，W = W - W_G - W_E, 执行 2,3 \end{cases}$

伪代码如下

LINEAR-TIME-FRACTION-KNAPSACK(V, W, W_1)
1. let A[1..n] be a new array
2. for i = 1 to n
3. A[i] = V[i] / W[i]
4. return INNER(A, w_1, sum)

INNER(A, w_1, sum)

1. let L, E, G be a new array
2. n = A.length / 2
3. put $\frac{v_i}{w_i}\ in\ A \begin{cases} < A[n]\ into\ L \\ = A[n]\ into\ E \\ > A[n]\ into\ G \end{cases}$
4. if ($W_G > W$)
5. A=G
6. INNER(A, w_1, sum)
7. else if ($W_G \le W \le W_G + W_E$)
8. sum = sum+$V_G + as\ much\ as\ V_E\ correspond\ to\ W - W_G$
9. return sum
10. else
11. sum = sum + $V_G + V_E$
12. $w_1 = w_1 - W_G - W_E$
13. INNER(L, w_1, sum)

16.3-7

可以用个最小堆来做，每个字符出现的频率作为关键字，伪代码如下：

TERNARY-HUFFMAN(C)
1. add a new node with frequency 0 to C if C.length is even
2. n = C.length
3. H = C
4. for i = 1 to $\lfloor \frac{n}{2} \rfloor$
5. allocate a new node as z
6. z.left = H.min
7. z.mid = H.min
8. z.right = H.min
9. z.freq = z.left+z.mid+z.right
10. INSERT(H,z)
11. return H.MIN

$$\because \text{分别用 } 0,1,2 \text{ 来表示当前最小频率的三个} key\text{，所以是前缀编码，所以是最优的}$$