8-4.集合划分问题的 NP 完全性证明：

问题即为给定一个整数集合 S，求集合 S 的一个划分 S1 和 S2（即：S=S1∪S2 且 S1∩S2=Φ），使得 S1 中的元素之和等于 S2 中的元素之和，证明此问题是 NP 难的。

证明：

由题目可知，只需要证明该集合划分问题是 NP 完全问题即可。为了证明任意一个问题 A 是 NP 完全问题，需要做到下面四个步骤。

1，存在一个非确定性多项式时间算法能解决 A，例如 A∈NP；

2，任意 NP 完全问题 B 能简化为 A；

3，由 B 到 A 的简化在多项式时间里可以完成；

4，当且仅当 B 有解时原始问题 A 有解。

现在来证明集合划分问题是 NP 完全的。

1，SET-PARTITION∈NP：假设两个划分，证明这两个划分有相同的元素之和；

2，子集和问题到集合划分问题的简化：子集和问题定义如下：给定一个整数集合 X 和目标值 t，找到一个子集 Y⊆X 使得集合 Y 中的成员之和为 t。假设 s 为 X 中的成员之和。将 X’ = X∪{s － 2t}加入集合划分中；

3，这个简化可在多项式时间内完成；

4，只需证明当且仅当<X’>∈集合划分问题时<X,t>∈子集和问题。X’的成员之和为 2s-2t；如果 X 中存在一个子集，其和为 t，那么 X 中剩余成员之和为 s-t。因此，存在 X’的一个划分使得划分的两个子集之和均为 s-t。如果存在一个 X’的划分使得划分的两个子集之和均为 s-t。两个子集之一包含数字 s － 2t。移除这个数字，我们得到一个子集，其和为 t，并且所有子集成员均属于集合 X。

命题得证。

8-5.最长简单回路问题：

证明：假设算法 Longest 能够解决图 G=(V,E)的最长简单回路问题，并返回该回路的顶点个数。因此，可以将哈密顿回路问题 HAM-CYCLE 在多项式时间内变换为最长简单回路问题 LONGEST-SIMPLE-CYCLE 如下：

```
Hamilton(G) {
    if (Longest(G) == |V|)
        return true;
    else
        return false;
}
```

故 HAM-CYCLE $\propto_p$ LONGEST-SIMPLE-CYCLE。

任给图 G=(V,E)的一条简单回路的猜想 y（y 是回路顶点 v 依次连接形成的串），容易验证 $v_i v_{i+1} \in E(1 \le i \le k-1)$且$v_k v_1 \in E$是否成立，若成立继续验证回路顶点个数$k \ge n$是否成立。显然，可在$O(k)$时间内判定图 G 是否存在一条顶点个数大于 n 的简单回路。

故 LONGEST-SIMPLE-CYCLE ∈ NP。

又由于哈密顿回路问题 HAM-CYCLE 是 NP 完全的，故最长简单回路问题 LONGEST-SIMPLE-CYCLE 也是 NP 完全的。

8-15.多机调度问题的近似算法：

可用贪心算法求得该问题的近似解。当作业数不超过机器数，即$n \le m$时，显然只需将每一

个作业单独分配给一台机器即可，所需加工时间为最长作业处理时间，算法所需时间为$O(1)$。当作业数大于机器数，即$n > m$时，先将作业处理时间按降序进行排序，耗时$O(nlogn)$；然后初始化小顶堆，用于存放机器，耗时$O(m)$；而关于小顶堆的 push 和 pop 操作共耗时$O(nlogm)$。此时，算法所需时间为$O(nlogn + nlogm) = O(nlogn)$。

证明：近似算法所得解的相对误差

$$\lambda = \left|\frac{c^* - c}{c^*}\right| \leq \frac{1}{3} - \frac{1}{3m}$$

假设 LPT 算法的最长处理时间为C_{LPT}，而多机调度问题的最优解为C_{OPT}。现将n个作业按处理时间的非增序排列，即$t_1 \geq t_2 \geq \cdots \geq t_n$。根据 LPT 算法，假设处理时间最长的机器上，最后一个要处理的作业为k，且在时间为t时开始处理，则在时间t之前所有机器均处于工作状态。从而有：

$$\sum_{i=1}^{k-1} t_i \geq m \cdot t$$

又

$$\sum_{i=1}^{k} t_i \leq \sum_{i=1}^{n} t_i \leq m \cdot C_{OPT}$$

因此，

$$t \leq \frac{1}{m}\left(\sum_{i=1}^{k} t_i - t_k\right) \leq C_{OPT} - \frac{t_k}{m}$$

故

$$C_{LPT} = t + t_k \leq C_{OPT} + (1 - \frac{1}{m})t_k$$

事实上，若$k \neq n$，即k不是处理时间最短的作业时，可以将处理时间小于t_k的作业删去，而 LPT 算法的解C_{LPT}仍保持不变。

当$t_k > \frac{1}{3}C_{OPT}$时，OPT 算法中每台机器最多只能处理 2 项作业，否则某一机器的处理时间$t' \geq 3t_k > C_{OPT}$，这与 OPT 算法的最优解相矛盾。这说明最优解是每台机器至多安排 2 项作业，在该前提下显然有$C_{LPT} = C_{OPT}$，相对误差$\lambda = 0$。

当$t_k \leq \frac{1}{3}C_{OPT}$时，有

$$C_{LPT} \leq C_{OPT} + (\frac{1}{3} - \frac{1}{3m})C_{OPT}$$

故

$$\lambda = \left|\frac{c^* - c}{c^*}\right| \leq \frac{1}{3} - \frac{1}{3m}$$

代码如下：

```
#include <iostream>
#include <vector>
#include <queue>
```

```cpp
#include <algorithm>
using namespace std;

//各个作业所需时间
vector<float> a = { 2, 14, 4, 16, 6, 5, 3 };

struct work {
    int id;           //作业编号
    float time;       //所需处理时间
    work(int a, float b) : id(a), time(b) {}
    bool operator <(const work& x)const {
        return x.time < time;
    }
};

struct machine {
    int id;           //机器编号
    float avail;      //完成当前作业所需时间
    machine(int a, float b) : id(a), avail(b) {}
    bool operator <(const machine& x)const {
        return x.avail < avail;
    }
};
//贪心算法
float LPT(int m) {
    int n = a.size();
    if (n <= m) {
        cout << "为每个作业分配一台机器" << endl;
        float total_time = a[0];
        for (int i = 1; i < n; i++)
            total_time = (a[i] > total_time) ? a[i] : total_time;
        return total_time;
    }

    //初始化作业
    vector<work> job;
    for (int i = 0; i < n; i++)
        job.push_back(work(i + 1, a[i]));
    //按作业处理时间降序排序
    sort(job.begin(), job.end());

    //初始化机器，建立小顶堆
    priority_queue<machine> MinHeap;
    for (int i = 0; i < m; i++)
```

```
        MinHeap.push(machine(i + 1, 0));

    //取堆顶机器，分配工作
    for (int i = 0; i < n; i++) {
        machine x = MinHeap.top();
        MinHeap.pop();
        cout << "将机器" << x.id << "从" << x.avail << "到" << x.avail + job[i].time
            << "的时间段分配给工作" << job[i].id << endl;
        x.avail += job[i].time;
        MinHeap.push(x);
    }

    //取最长工作时间
    for (int i = 0; i < m - 1; i++)
        MinHeap.pop();
    return MinHeap.top().avail;
}

int main(void) {
    int t = LPT(3);
    cout << "最短时间：" << t << endl;
    return 0;
}
```