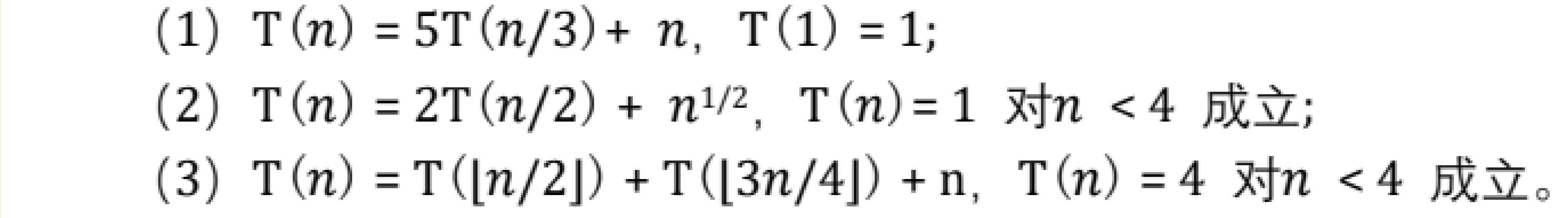


(1) $T(n) = 5T(n/3) + n$，$T(1) = 1$;

(2) $T(n) = 2T(n/2) + n^{1/2}$，$T(n) = 1$ 对$n < 4$ 成立;

(3) $T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor 3n/4 \rfloor) + n$，$T(n) = 4$ 对$n < 4$ 成立。

(1) $T(n) = 5T(n/3) + n, T(1) = 1$

$a=5, b=3, n^{\log_b a} = n^{\log_3 5}$ $\qquad 1 < \log_3 5 < 2$

$f(n) = n.$ $\quad \therefore n^{\log_b a} > f(n)$，且是多项式的大

$\therefore T(n) = \Theta(n^{\log_3 5})$

(2) $T(n) = 2T(n/2) + n^{\frac{1}{2}}$，$T(n)=1$，对$n<4$成立

$a=2, b=2, n^{\log_b a} = n^{\log_2 2} = n$

$f(n) = n^{\frac{1}{2}}$

$\therefore n^{\log_b a} > f(n)$

$\therefore T(n) = \Theta(n)$

(3) $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{3n}{4} \rfloor) + n$，当$n<4$时，$T(n)=4$

猜测你猜不出来.

(1) $T(n) = T\lfloor \frac{n}{2} \rfloor + T\lfloor \frac{n}{\frac{4}{3}} \rfloor + n \leq 2 \cdot T(\frac{n}{\frac{4}{3}}) + n$

$a=2, b=\frac{4}{3}, \log_b a = \log_{\frac{4}{3}} 2 \in (2,3)$

$f(n) = n.$

$\therefore f(n) < n^{\log_{\frac{4}{3}} 2}$，$\therefore T(n) = O(n^{\log_{\frac{4}{3}} 2})$

(2) $T(n) = T\lfloor \frac{n}{2} \rfloor + T\lfloor \frac{n}{\frac{4}{3}} \rfloor + n \geqslant 2 \cdot T(\frac{n}{2}) + n.$

$a=2, b=2. \quad n^{\log_b a} = n$，$f(n)=n \qquad f(n) = n^{\log_b a}$

$\therefore T(n) = \Omega(n \cdot \lg n)$

∴找到了T(n)的上界和下界.

$\therefore\ n\cdot\lg n < T(n) < n^{\log_{\frac{4}{3}}2}$

猜测 $T(n) = O(n^2)$，$T(n) \le cn^2$

$\therefore T(n) \le c\cdot\frac{n^2}{4} + c\cdot\frac{9}{16}n^2 + n$

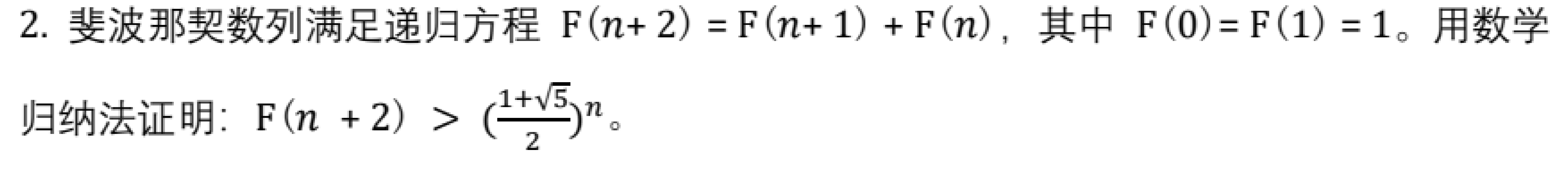
$= \frac{13c}{16}n^2 + n$

$= (c - \frac{3c}{16})\cdot n^2 + n$

$= cn^2 + n - \frac{3c}{16}n^2$

$\le cn^2$，只需要 $\frac{3c}{16}n^2 > n \Rightarrow c > \frac{16}{3n} \Rightarrow c > \frac{16}{3}$ 即可.

$\therefore T(n) = O(n^2)$



2. 斐波那契数列满足递归方程 $F(n+2) = F(n+1) + F(n)$，其中 $F(0) = F(1) = 1$。用数学归纳法证明：$F(n+2) > (\frac{1+\sqrt{5}}{2})^n$。

当 $n=0$ 时，$F(2) = F(1) + F(0) = 2 > (\frac{1+\sqrt{5}}{2})^0 = 1$

假设 $\begin{cases} n=m \text{ 时，满足 } F(m+2) > (\frac{1+\sqrt{5}}{2})^m, \\ n=m-1 \text{ 时，满足 } F(m+1) > (\frac{1+\sqrt{5}}{2})^{m-1}, \end{cases}$

$n \le m$ 时满足

当 $n = m+1$ 时，$F(m+3) = F(m+2) + F(m+1)$

$> (\frac{1+\sqrt{5}}{2})^m + (\frac{1+\sqrt{5}}{2})^{m-1}$

$= (\frac{1+\sqrt{5}}{2} + 1)(\frac{1+\sqrt{5}}{2})^{m-1}$

$= (\frac{3+\sqrt{5}}{2})\cdot(\frac{1+\sqrt{5}}{2})^{m-1}$

$= (\frac{1+\sqrt{5}}{2})^{m+1}$

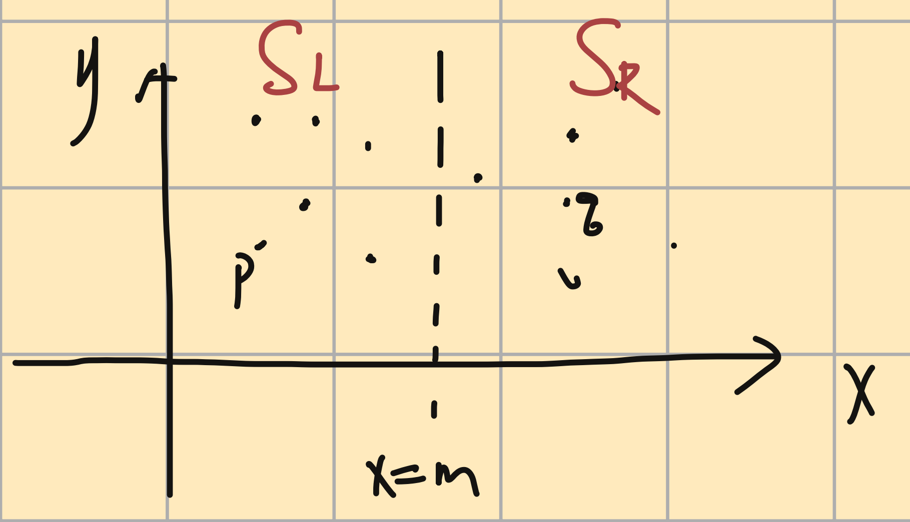
$(\frac{1+\sqrt{5}}{2})^2 = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2}$

证毕.

3. 给定平面上$n$个点构成的集合$S$，设计分治算法输出$S$的三个点，使得以这三个点为顶点的三角形的周长达到最小值。(提示：模仿最邻近点对的分治过程)。

假设：S中的坐标已按x坐标、y坐标排好序

边界条件：如果S中仅有3个点，则返回三个点的距离之和；仅有2个点，则返回0



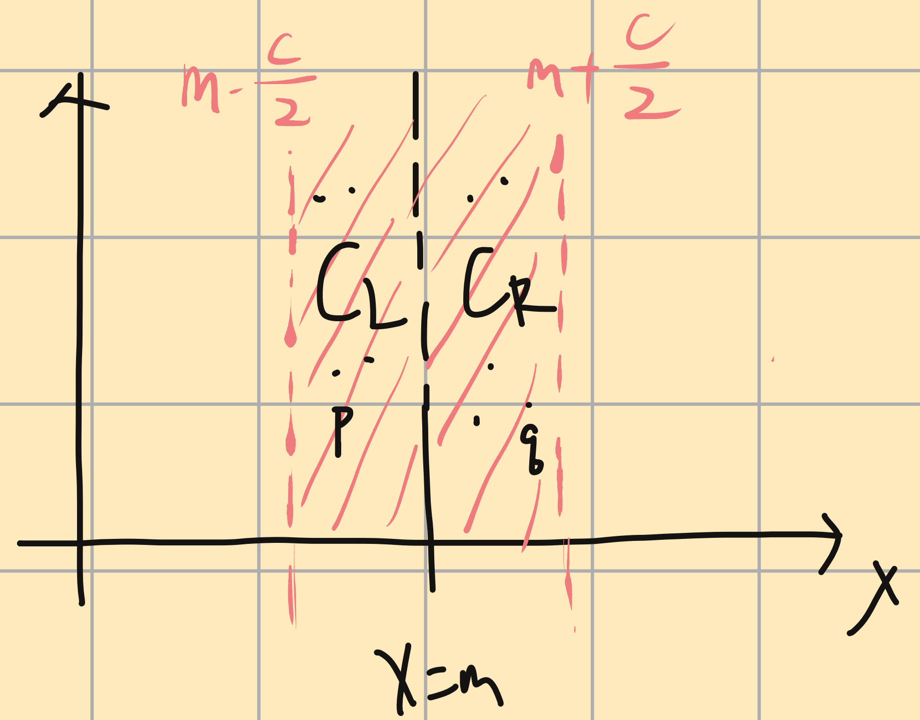
(1) 找到x的中位数m，用x=m将S划分为$S_L$和$S_R$

(2) 递归的在$S_L$、$S_R$中找出能构成三角形且周长最小的三个点.

$C(p_1, p_2, p_3) \in S_L$

$C(q_1, q_2, q_3) \in S_R$

(3) $C = \min\{C(p_1, p_2, p_3), C(q_1, q_2, q_3)\}$



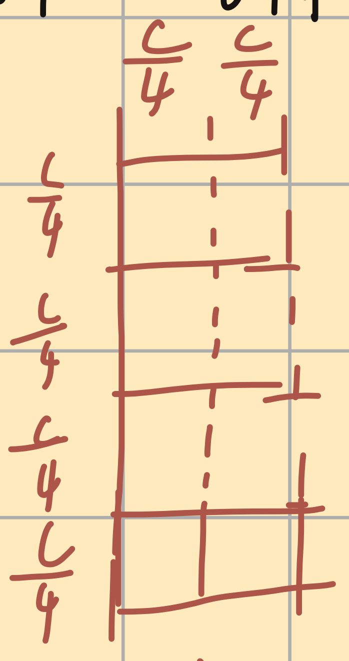
(4) 在临界区查找能构成三角形且不在同一子集且周长小于C的三个点

(5) 长度最小的就是答案.

tips：如果有一条边大于$\frac{c}{2}$，∵两边之和大于第三边，∴周长必定大于C，∴临界区x的范围：$[m-\frac{c}{2}, m+\frac{c}{2}]$

(6)：对于临界区的搜索，对于$\forall p \in C_L$，在$C_R$中找到两个能与p构成三角形且周长小于C的点.

对于$\forall q \in C_R$，在$C_L$中找到两个能与q构成三角形且周长小于C的点.



鸽巢原理.

分成8个区域，每个区域两个点

最多有16个点. ∴还是O(n)

如果一个区域有3个点，则周长小于C 矛盾.

```
get_min(S, n)
    if n ≤ 2    return 0;
    if n = 3  and  is △      return d;
    计算x的中位数m，用x=m划分为$S_L$、$S_R$
    $C_1$ = get_min($S_L$, $\frac{n}{2}$)
    $C_2$ = get_min($S_R$, $\frac{n}{2}$)
    C = min($C_1$, $C_2$)
    for p in $C_L$:
        if ($q_i$, $q_j$) 使得 is △ 且 $C_p$ < C
        C = $C_p$
    for q in $C_R$:
        if ($p_i$, $p_j$) 使得 is △ 且 $C_q$ < C
        C = $C_q$
    return C;
```

$T(n) = 2T(\frac{n}{2}) + O(n)$

根据master定理：

$T(n) = O(n)$

4. 给定一个整数序列$a_1, \dots, a_n$。相邻两个整数可以合并，合并两个整数的代价是这两个整数之和。通过不断合并最终可以将整个序列合并成一个整数，整个过程的总代价是每次合并操作代价之和。试设计一个动态规划算法给出$a_1, \dots, a_n$的一个合并方案使得该方案的总代价最大。设计动态规划算法求解此问题并分析算法的时间复杂性。

优化子结构：找到一个 $a_k$ 使得 $m[i,k-1]+m[k,j]$ 最大，此时 $m[i,k-1]$ 和 $m[k,j]$ 也一定最大，具有优化子结构

重叠子问题：具有重叠子问题

递推公式：

$$m[i,j] = \begin{cases} 0, & i==j \\ \max\limits_{\underset{2}{i+1} \le k \le \underset{3}{j}} \{ m[i,k-1] + m[k,j] + s[i,j] \}, & i<j \end{cases}$$

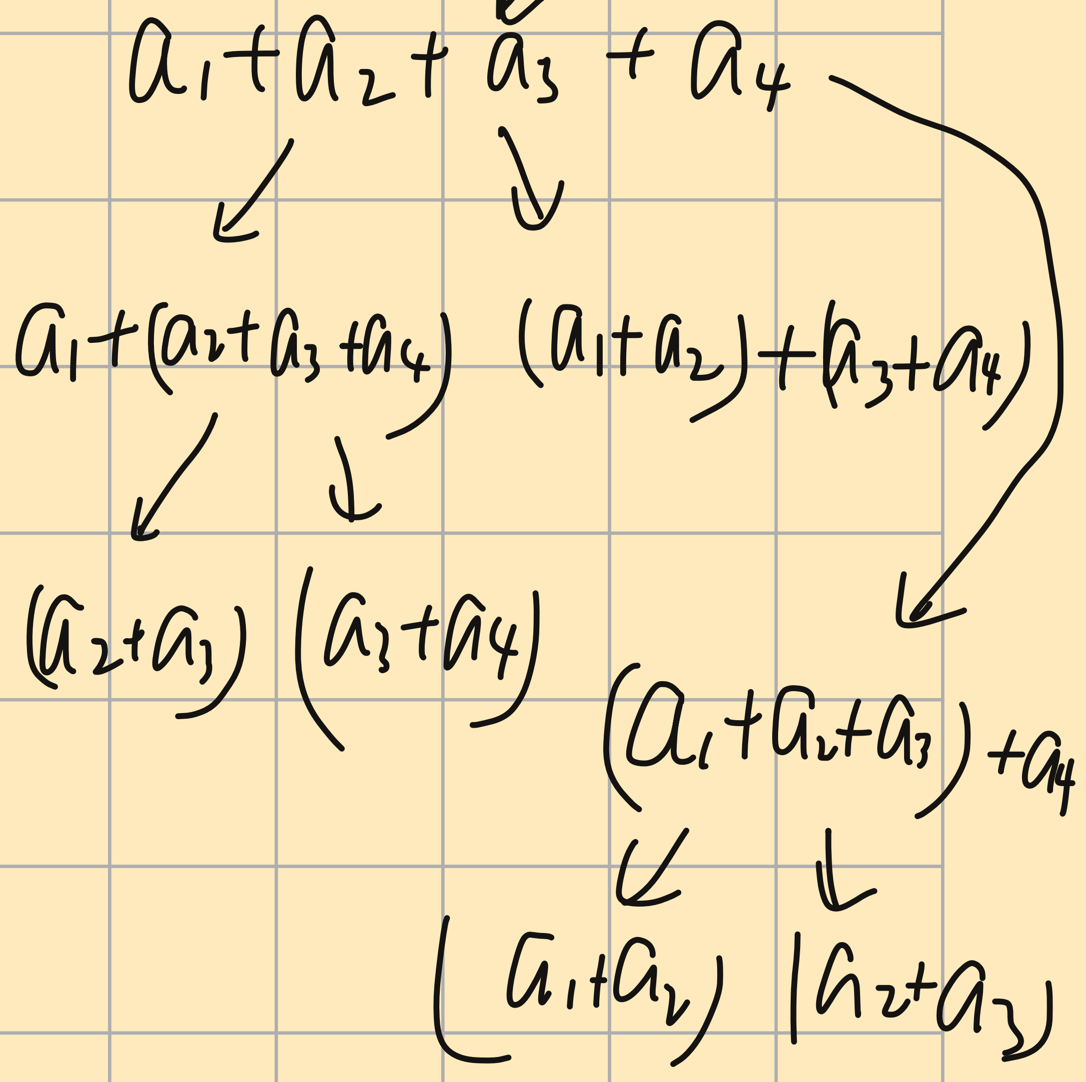
$s[i,j]$ 表示 $a_i + \dots + a_j$

$$\begin{cases} s[i,j] = a_i, & i==j \\ s[i,j] = s[i,j-1] + a_j, & i<j \end{cases}$$

计算二维数组m：

例：整数序列：1, 2, 3, 4, 5.

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 11 & 26 & 50 \\ 2 & & 0 & 5 & 16 & 35 \\ 3 & & & 0 & 7 & 21 \\ 4 & & & & 0 & 9 \\ 5 & & & & & 0 \end{array}$$



伪代码：用 $D[i,j]=k$ 表示合并 $a_i \cdots a_j$ 在 $a_k$ 处断开

$m[i,j]$ 表示代价，$s[i,j]$ 表示 $a_i \cdots a_j$ 之和.

```
get_max_m(a_1,...a_n)
    for(int i=1; i≤n, i++)
        m[i,i]=0, s[i,i]=a_i;
    for(int l=2, l≤n, l++)
        for(int i=1; i≤n-l+1; i++)
            j=i+l-1;
            m[i,j)=-∞; s[i,j]=s[i,j-1]+a_j;
            for(int k=i+1; k≤j, k++)
```

递推公式里的 $\max\limits_{i+1 \le k \le j}$

这是一层层循环

用l表示第几层，从2→n

i: n-l+1，随l而↓

j=i+l-1，随l而↑

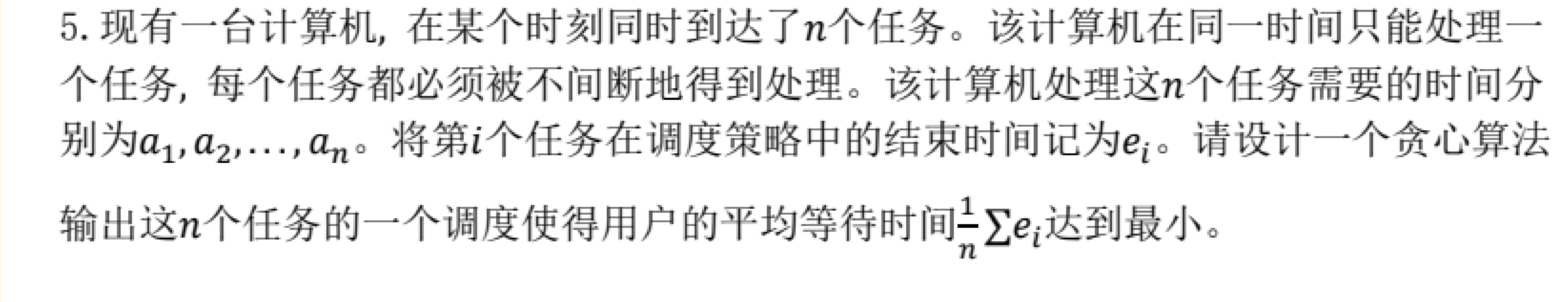
```
            q = m[i, k-1] + m[k, j]
            if q > m[i, j]     m[i, j] = q;  D[i, j] = k;
        m[i, j] = m[i, j] + s[i, j];

Pr (D, i, j)
  if i==j    print a_i.
  else  print("(");
        Pr (D, i, D(i,j) -1);
        print("+");
        pr (D, D(i,j), j);
        print(")");
```

时间复杂度：计算中有三层循环。∴ $O(n^3)$

构造最优解的时间：$O(n)$

∴总时间复杂度为 $O(n^3)$.



5. 现有一台计算机，在某个时刻同时到达了$n$个任务。该计算机在同一时间只能处理一个任务，每个任务都必须被不间断地得到处理。该计算机处理这$n$个任务需要的时间分别为$a_1, a_2, \ldots, a_n$。将第$i$个任务在调度策略中的结束时间记为$e_i$。请设计一个贪心算法输出这$n$个任务的一个调度使得用户的平均等待时间$\frac{1}{n}\sum e_i$达到最小。

想法：依次处理所需时间最短的任务

贪心选择性：设 $A=\{a_1, a_2, \cdots a_n\}$ 是n个任务且处理时间 $a_1 \le a_2 \le \cdots \le a_n$.

则存在一个最优解，将时间最短的 $a_1$ 排在第一个处理.

设 $a_i a_j \cdots a_k$ 是问题的最优解.

若 $a_i = a_1$，则贪心选择性成立.

若 $a_i \neq a_1$，假设顺序为 $a_i a_j \cdots a_1 \cdots a_k$.

此时将$a_i$与$a_1$调换位置为：$a_1 a_j \cdots a_i \cdots a_k$.

对于$a_i$后面的任务结束时间不变.

对于$a_i$前面的任务结束时间提前

因为$a_1$所需处理时间少于$a_i$.

∴调换后的$\frac{1}{n}\sum e_i$变小，也是一个最优解且优先处理任务一.

优化子结构：设$A=\{a_1, a_2, \cdots a_n\}$是n个任务，且处理时间从小到大.

由贪心选择性知，一定先处理第一个任务；

设一个最优解为：$a_1 a_i a_j \cdots a_k$，

则$a_i a_j \cdots a_k$一定是$A'=\{a_2, \cdots a_n\}$的一个最优解.

证明：假设$a_i a_j \cdots a_k$不是$A'$的最优解.

则存在$a_{i'} a_{j'} \cdots a_{k'}$是$A'$的最优解

满足$\frac{1}{n-1}\sum e_{i'} < \frac{1}{n-1}\sum e_i$

此时比较：$a_1 a_i a_j \cdots a_k$与$a_1 a_{i'} a_{j'} \cdots a_{k'}$的$\frac{1}{n}\sum e_i$.

因为$a_1$的处理时间相同.

所以$\frac{1}{n}\sum e_i > \frac{1}{n}\sum e_{i'}$

与$a_1 a_i a_j \cdots a_k$是原任务的最优解矛盾.

∴$a_i a_j \cdots a_k$是$A'$的最优解.

因此，若$A=\{a_1, a_2, \cdots a_n\}$已按处理时间排序.

则$a_1 \to a_2 \to \cdots \to a_n$这个顺序是问题的最优解.

时间复杂性：若$a_i$已排序：$O(n)$

若$a_i$未排序：$O(n)+O(n\log n)=O(n\log n)$.