1. **假定我们对一个数据结构执行一个由 n 个操作组成的操作序列，当 i 严格为 2 的幂时，第 i 个操作的代价为 i，否则代价为 1.**

   （1）使用聚合分析确定每个操作的摊还代价。

   （2）使用核算法确定每个 操作的摊还代价。

   （3）使用势能法确定每个操作的摊还代价。

   (1) 对于n个操作，其中为2的幂的操作共有$\lfloor lgn \rfloor + 1$个，故总摊还代价为

$$\begin{aligned}
&\sum_{i=0}^{\lfloor lgn \rfloor} 2^i + \sum_{i=1}^{n-\lfloor lgn \rfloor -1} 1 \\
&= \frac{1 \times (1 - 2^{\lfloor lgn \rfloor +1})}{1-2} + n - \lfloor lgn \rfloor - 1 \\
&= 2^{\lfloor lgn \rfloor +1} - 1 + n - \lfloor lgn \rfloor - 1 \\
&\le 2n + n - 2 - lgn \\
&= 3n - lgn \\
&= O(n)
\end{aligned}$$

   故每个操作的摊还代价为O(1)

   (2) 赋予每个操作的摊还代价为3，则对于第2^{k-1}+1到第2^k-1个操作来说，这些操作的所多付的信用为

   (3-1)(2^k-1-(2^{k-1}+1)+1)=2(2^{k-1}-1)=2^k-2

   对于第2^k个操作，已经付了3个信用，加上之前多付的信用，总共付的信用为2^k-2+3=2^k+1>2^k，大于第2^k次操作本来应该付的代价。由于故每个操作的摊还代价为O(1)

   (3) 定义势函数 $\Phi(D_i) = 2i - 2^{\lfloor \lg i \rfloor +1}, \Phi(D_0) = 0$

   对于任意 i ，有$\Phi(D_i) = 2i - 2^{\lfloor \lg i \rfloor +1} = 2(i - 2^{\lfloor \lg i \rfloor}) \ge 2(i - 2^{\lg i}) = 0 = \Phi(D_0)$

   则对于 i 不是2的幂，有$\lfloor \lg i \rfloor = \lfloor \lg i - 1 \rfloor$，故摊还代价

$$\begin{aligned}
&\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= 1 + 2i - 2^{\lfloor \lg i \rfloor +1} - (2(i-1) - 2^{\lfloor \lg i - 1 \rfloor +1}) \\
&= 1 + 2 = 3
\end{aligned}$$

   对于 i 是2的幂，有$\lg i = \lfloor \lg i \rfloor = \lfloor \lg i - 1 \rfloor + 1, 2^{\lfloor \lg i \rfloor} = i$，故摊还代价

$$\begin{aligned}
&\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= i + 2i - 2^{\lfloor \lg i \rfloor +1} - (2(i-1) - 2^{\lfloor \lg i - 1 \rfloor +1}) \\
&= i + 2 - 2^{\lfloor \lg i \rfloor +1} + 2^{\lfloor \lg i - 1 \rfloor +1} \\
&= i + 2 - 2i + i = 2
\end{aligned}$$

   故每个操作的摊还代价为O(1)

2. **V.Pan 发现一种方法，可以用 132464 次乘法操作完成 68×68 的矩阵相乘，发现另一种方法，可以用 143664 次乘法操作完成 70×70 的矩阵相乘，还发现一种方法，可以用 155424 次乘法操作完成 72×72 的矩阵乘法。当用于矩阵乘法的分治算法时，上述哪种方法会得到最佳的渐近运行时间？与 Strassen 算法相比，性能如何？**

对于用 132464 次乘法操作完成 68×68 的矩阵相乘，有 $\lg_{68} 132464 = 2.795128$

对于用 143664 次乘法操作完成 70×70 的矩阵相乘，有 $\lg_{70} 143664 = 2.795123$

对于用 155424 次乘法操作完成 72×72 的矩阵乘法，有 $\lg_{72} 155424 = 2.795147$

而Strassen算法的时间复杂度为$O(n^{\lg 7}) = O(n^{2.807355})$，故用 143664 次乘法操作完成 70×70 的矩阵相乘会得到最佳的渐进运行时间，与 Strassen 算法相比，这些算法性能更好一些。

3. **3.** 我们可以将一维离散傅里叶变换 (DFT) 推广到 d 维上。这时输入是一个 d 维的数组 $A=(a_{j_1,j_2,\ldots,j_d})$，维数分别为 $n_1,n_2,\ldots,n_d$，其中 $n_1n_2\ldots n_d = n$。定义 d 维离散傅里叶变换如下：

$$y_{k_1,k_2,\ldots,k_d} = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\cdots\sum_{j_d=0}^{n_d-1} a_{j_1,j_2,\ldots,j_d}\omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}\ldots\omega_{n_d}^{j_dk_d}$$

其中 $0\le k_1<n_1, 0\le k_2<n_2,\ldots,0\le k_d<n_d$。

a. 证明：我们可以依次在每个维度上计算一维的 DFT 来计算一个 d 维的 DFT。也就是说，首先沿着第 1 维计算 n/n_1 个独立的一维 DFT。然后，把沿着第 1 维的 DFT 结果作为输入，我们计算沿着第 2 维的 n/n_2 个独立的一维 DFT。利用这个结果作为输入，我们计算沿着第三维的 n/n_3 个独立的一维 DFT，如此下去，直到第 d 维。

b. 证明：维度的次序并无影响，于是可以通过在 d 个维度的任意顺序中计算一维 DFT 来计算一个 d 维的 DFT。

c. 证明：如果采用计算快速傅里叶变换计算每个一维的 DFT，那么计算一个 d 维的 DFT 的总时间是 $O(n\lg n)$，与 d 无关。

a.

$$\begin{aligned}
&y_{k_1,k_2,\ldots,k_d}\\
&=\sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\cdots\sum_{j_d=0}^{n_d-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1k_1} w_{n_2}^{j_2k_2}\ldots w_{n_d}^{j_dk_d}\\
&=\sum_{j_d=0}^{n_d-1}\cdots\sum_{j_2=0}^{n_2-1}\sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1k_1} w_{n_2}^{j_2k_2}\ldots w_{n_d}^{j_dk_d}\\
&=\sum_{j_d=0}^{n_d-1}\cdots\sum_{j_2=0}^{n_2-1}(\sum_{j_1=0}^{n_1-1} a_{j_1,j_2,\ldots,j_d} w_{n_1}^{j_1k_1}) w_{n_2}^{j_2k_2}\ldots w_{n_d}^{j_dk_d}\\
&=\sum_{j_d=0}^{n_d-1}\cdots\sum_{j_2=0}^{n_2-1} a_{j_2,\ldots,j_d} w_{n_2}^{j_2k_2}\ldots w_{n_d}^{j_dk_d}\\
&=\sum_{j_d=0}^{n_d-1}\cdots\sum_{j_3=0}^{n_3-1}(\sum_{j_1=0}^{n_2-1} a_{j_2,\ldots,j_d} w_{n_2}^{j_2k_2}) w_{n_3}^{j_3k_3}\ldots w_{n_d}^{j_dk_d}\\
&=\ \ldots\ldots
\end{aligned}$$

先计算括号里面的，总共需计算$n_2\cdot n_3\cdot\ldots\cdot n_d = n/n_1$次，也就是沿着第一维计算$n/n_1$个独立的一维DFT，再把这个结果作为输入，继续计算第二维的DFT，对于每个第一维的结果要计算n/n_1n_2次，故总共要计算n/n_2个。每次计算出结果，都可以减小一维，直到问题解决，也就是到第n维。

b.对于上述等式，由于求和号之间相互独立，故可以互换次序，故可以交换先后计算维度的次序。

c.

计算每一个维度的时间都是$O(n_i\lg n_i)$，故计算一个d维的DFT的总时间为

$$total = \sum_{i=1}^{d} O(n_i\lg n_i) = O(\sum_{i=1}^{d} n_i\lg n_i) \le O(\lg n\sum_{i=1}^{d} n_i) = O(n\lg n)$$

故总时间为O(nlgn)，与d无关