一、已知下列递推式：

C(n) = 1 若 n =1

= 2C（n/2） + n - 1 若 n ≥ 2

请由定理 1 导出 C(n)的非递归表达式并指出其渐进复杂性。

定理 1：设 a, c 为非负整数，b, d, x 为非负常数，并对于某个非负整数 k, 令 $n=c^k$, 则以下递推式

f(n) =d 若 n=1

=af(n/c)+bn^x 若 n>=2

的解是

f(n)= $bn^x\log_c n + dn^x$ 若 $a=c^x$

f(n)= $\left(d+\frac{bc^x}{a-c^x}\right)n^{\log_c a}-\left(\frac{bc^x}{a-c^x}\right)n^x$ 若 $a\neq c^x$

解：令 F(n) = C(n)-1

则 F(n) = 0 n = 1

F(n) = 2C(n/2) + n -2 n ≥ 2

= 2[F(n/2) + 1] + n – 2

=2F(n/2) + n

所以利用定理 1，我们可以得到：

d = 0,a=2,c=2,b=1,x=1,且 $a=c^x$

所以 $F(n)=n\log_2 n$

$C(n)=F(n) + 1 = n\log_2 n + 1$

C(n)的渐进复杂性为 $O(n\log_2 n)$。

二、由于 Prim 算法和 Kruskal 算法设计思路的不同，导致了其对不同问题实例的效率对比关系的不同。请简要论述：

1、如何将两种算法集成，以适应问题的不同实例输入；

2、你如何评价这一集成的意义？

答：

1. **Prim 算法是基于顶点进行搜索的，适合顶点少边多的稠密图。而 Kruskal 算法基于边集合中进行搜索，所以适合边较少的稀疏图。**
   根据输入的图中顶点和边的情况，边少的选用 Kruskal 算法，顶点少的选用 Prim 算法。
   以下是考虑能不能把两个算法融合，根据当下图的情况随时切换选择使用 Kruskal 算法还是 Prim 算法？
   我们可以通过把临时的最小生成树看做是一个点，把生成过程中的最小生成树里面点集的互相连接的边从边集合中去除，维护一个点集和边集。
   假设一开始是顶点少边多的稠密图时候，一开始使用 Prim 算法去生成最小生成树，当顶点数和边数到达某一阈值时候，稠密图变成边较少的稀疏图，我们可以换成 Kruskal 算法进行运算。而相反稀疏图到稠密图也可以这样做，通过还未加入最小生成树的顶点数和边数进行判断，从而决定当下使用 Prim

算法还是 **Kruskal** 算法。

2. 评价意义：没有一个算法的万能的，任何算法都需要面对具体的情况进行对应的设计，才能发挥出最优的效率。所以设计算法需要具体问题具体分析，要充分了解问题的本质，不能胡乱套用算法框架，要知道再好的算法也有运行效率最差的情况出现。

三、分析以下生成排列算法的正确性和时间效率：

```
HeapPermute(n)
//实现生成排列的 Heap 算法
//输入：一个正正整数 n 和一个全局数组 A[1..n]
//输出：A 中元素的全排列
if n = 1
    write A
else
    for i ←1 to n do
        HeapPermute(n-1)
        if n is odd
            swap A[1]and A[n]
        else swap A[i]and A[n]
```

答：

n=1 时，输出 a1

n=2 时，输出 a1a2,a2a1

n=3 时

（1） 第一次循环 **i=1** 时，进入 **HeapPermute (2)**将 **a1a2** 做全排列输出，输出 **a1a2a3**，**a2a1a3**，记为**[a1a2]a3**,然后在把 **A** 变为 **a2a1a3** 之后，交换 **1,3** 位，得 **a3a1a2**。

（2） 第二次循环 **i=2** 时，**HeapPermute(2)**输出**[a3a1]a2**，并将 **A** 变为 **a1a3a2**，交换 **1,3** 位，得 **a2a3a1**。

（3） 第三次循环 **i=3** 时，**HeapPermute(2)**输出**[a2a3]a1**，并将 **A** 变为 **a3a2a1**，交换 **1,3** 位，得 **a1a2a3**。即输出完毕后数组 **A** 回到初始的序列。

n=4 时

（1） **i=1** 时，**HeapPermute(3)**输出**[a1a2a3]a4**，并且 **a1a2a3** 顺序不变，交换 **1,4** 位，得 **a4a2a3a1**。

（2） **i=2** 时，**HeapPermute(3)**输出**[a4a2a3]a1**，并且 **a4a2a3** 顺序不变，交换 **2,4** 位，得 **a4a1a3a2**。

（3） **i=3** 时，**HeapPermute(3)**输出**[a4a1a3]a2**，并且 **a4a1a3** 顺序不变，交换 **3,4** 位，得 **a4a1a2a3**。

（4） **i=4** 时，**HeapPermute(3)**输出**[a4a1a2]a3**，并且 **a4a1a2** 顺序不变，交换 **4,4** 位，得 **a4a1a2a3**。即输出完毕后数组 **A** 循环右移一位。

因此有上面总结分析可以得到：

当 **n** 为偶数时候，**HeapPermute(n)**输出全排列后数组元素循环右移一位。

当 **n** 为奇数时候，**HeapPermute(n)**输出全排列后数组元素顺序保持不变。

因此由数学归纳法可得：

(1) 当 **i=1** 时，显然成立。

(2) i=k 为偶数时，假设输出的是全排列，则 **i=k+1**(奇数)时，**k+1** 次循环中，每次前 **k** 个元素做全排列输出后循环右移一位，所以对换 **swap A[1]and A[n]**可以保证每次将前 **k** 个元素中的一个换到 **k+1** 的位置，所以 **k+1** 次循环后输出的是 **A[1…k+1]**的全排列。

(3) i=k 为奇数时，假设输出的是全排列,则 **i=k+1**(偶数)时，**k+1** 次循环中，每次前 **k** 个元素做全排列输出后顺序保持不变，所以对换 **swap A[i]and A[n]**可以保证每次将前 **k** 个元素中的一个换到 **k+1** 的位置，所以 **k+1** 次循环后输出的是 **A[1…k+1]**的全排列。

证毕。

时间复杂度递推公式为

$$T(n) = \begin{cases} 1 & \text{若 } n=1 \\ n[T(n-1)+2] & \text{若 } n>=2 \end{cases}$$

$$\begin{aligned} T(n) &= n[T(n-1)+2] \\ &= n(n-1)T(n-2) + 2n(n-1) + 2n \\ &= n(n-1)(n-2)T(n-3) + 2n(n-1)(n-2) + 2n(n-1) + 2n \\ &\cdots \\ &= n! + 2(n! + \ldots + n(n-1)(n-2) + n(n-1) + n) \\ &= n! + 2(1^2 + 2^2 + 3^2 + \ldots + n^2) \\ &= n! + n(n+1)(2n+1)/3 \end{aligned}$$

所以时间复杂度为 $T(n) = O(n!)+O(n^3)$

四、对于求 n 个实数构成的数组中最小元素的位置问题，写出你设计的具有减治思想算法的伪代码，确定其时间效率，并与该问题的蛮力算法相比较。

答：

（1）算法思想：将 n 个实数划分为[n/2](向下取整),n-[n/2]两个部分，分别找出其中最小的元素及其位置，比较这两个元素的大小，得出总的最小元素的位置。

（2）伪代码：

```
(element,index) = FindLeastElement(A,low,high)
//从数组 A[low…high]中找到最小元 element，及其位置 index
//input:实数数组 A[low…high]，数组起始下标 low,数组终止下标 high
//output:最小元素 element 及其位置 index
if(low==high)
    return (A[a],a)
else
    (x1,i) = FindLeastElement(A,low,(low+high)/2)
    (x2,j) = FindLeastElement(A, (low+high)/2+1,high)
if(x1<x2)
    return x1,i
else
    return x2,j
```

(3)算法复杂度递推公式为：F(n) = 1　　n=1

=2F(n/2)+1　n>1

化简为：F(n) = 2F(n/2) + 1

=2[2F(n/4)+1]+1

=2^3F(n/2^3)+4+2+1

…

=2^kF(n/2^k)+1+2+4+…+2^{k-1}

因为 n=2^k　=2n-1

所以算法复杂度为 O(2n-1)

蛮力法的复杂度为 O(n)，所以这个方法还没有蛮力法效率高，并且空间占用也比蛮力法多，因为减治处理后会增加比较次数。

五、请给出约瑟夫斯问题的非递推公式 J（n），并证明之。其中，n 为最初总人数，J(n) 为最后幸存者的最初编号。

答：

约瑟夫问题是个著名的问题：N 个人围成一圈，第一个人从 1 开始报数，报 M 的将被杀掉，下一个人接着从 1 开始报。如此反复，最后剩下一个，求最后的胜利者。

递推式证明如下：

假设一开始有 2n 个人，且 m=2 有：

当有 2n 偶数个人时候第一轮报数序号为偶数的出局，则有

1　~~2~~　3　4　5　~~6~~　7　~~8~~　9　~~10~~　11　~~12~~　旧编号 2n-1

1　2　3　4　5　6　新编号 n

则有 J(2n)=2J(n)-1　n>=1

当有 2n+1 奇数个人时候第一轮报数序号为偶数的出局，则有

~~1~~　~~2~~　3　4　5　~~6~~　7　~~8~~　9　~~10~~　11　~~12~~　13　旧编号 2n+1

1　2　3　4　5　6　新编号 n

注意这里 1 为标号为 2n 的人之后被删除，我们类比上面情况，并以此作为一轮

则有 J(2n)=2J(n)+1　n>=1

因此将上面内容结合起来，有:

J(1)=1

J(2n) = 2J(n) – 1　n>=1

J(2n+1) = 2J(n) + 1　n>=1

我们可以快速列出前面几个数,N 为 J(2n)和 J(2n+1)里面的 2n 与 2n+1

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
J(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1

我们发现表中的数据以 2 的幂将表分组(1,2,4,8…),并且每一组中的数据都是在递增 2。所以可以将 N 表示成 N = 2^m + b , m 是使 2^m 不超过 N 的最大幂次，l 表示在每一个分组中所占的位置，此时的递归式的解可以表示为:

J(2^m +b)=2*b+1 , m>=0 , 0<=b<2^m.

证明如下（数学归纳法）：

（1）　i=1 时，m=0，b=0，J(1) = 2*0+1=1，成立。

（2） i>1 时，

当 i 为偶数时，设 N=i/2 时成立，即 $N=2^m+b$，则 J(N)=2b+1,

此时，$i=2N=2^{m+1}+2b$

$J(i)=J(2N)=2J(N)-1=2(2b+1)-1=4b+1=2(2b)+1$,即 k=i 时成立。

当 i 为奇数时，设 N=(i-1)/2 时成立，即 $N=2^m+b$，则 J(N)=2b+1,

此时，$i=2N+1=2^{m+1}+2b+1$

$J(i)=J(2N+1)=2J(N)+1=2(2b+1)+1=4b+3=2(2b+1)+1$,即 k=i 时成立。

综上所述，递推式证明成立。

当 m=2 时的非递推式为：

$$J(n) = 1 + 2n - 2^{1+\lfloor \log_2 n \rfloor}$$

证明如下：

（1） 当 n=1，则 J(n) = 1 + 2 − 2 = 1，成立

（2） 当 n=2k，k = 1，2，3，…时，假设 n=k 时等式成立，则有

$$J(2k) = 2J(k) - 1$$

$$= 2(1 + 2k - 2^{1+\lfloor \log_2 k \rfloor}) - 1$$

$$= 2 + 4k - 2^{2+\lfloor \log_2 k \rfloor} - 1$$

$$= 1 + 4k - 2^{1+\log_2 2+\lfloor \log_2 k \rfloor}$$

$$= 1 + 4k - 2^{1+\lfloor \log_2 2k \rfloor}$$

$$= 1 + 2(2k) - 2^{1+\lfloor \log_2 2k \rfloor}$$

等式成立。

（3） 当 n=2k+1，k = 1，2，3，…时，假设 n=k 时等式成立，则有

$J(2k) = 2J(k) + 1$

$$J(2k + 1) = 2J(k) + 1$$

$$= 2(1 + 2k - 2^{1+\lfloor \log_2 k \rfloor}) + 1$$

$$= 3 + 4k - 2^{2+\lfloor \log_2 k \rfloor}$$

$$= 1 + 2 + 4k - 2^{1+1+\lfloor \log_2 k \rfloor}$$

$$= 1 + 2(2k + 1) - 2^{1+\lfloor \log_2 2k+1 \rfloor}$$

等式成立。

综上所述，非递推式成立。