一、已知下列递推式：

$$C(n) = \begin{cases} 1 & 若\ n=1 \\ 2C(n/2) + n - 1 & 若\ n \geqslant 2 \end{cases}$$

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

$$f(n) = \begin{cases} d & 若\ n=1 \\ af(n/c)+bn^x & 若\ n>=2 \end{cases}$$

的解是

$$f(n) = bn^x \log_c n + dn^x \quad 若\ a=c^x$$

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

答：

设 F(n) = C(n) - 1

则有：

$F(n) = 0 \qquad n = 1$

$F(n) = 2C(n/2) + n - 2 \qquad n >= 2$

$= 2[F(n/2) + 1] + n - 2$

$= 2F(n/2) + n$

由定理一可知，a=2, c=2, b=1, x=1, d=0 因为 $a=c^x$，则有：

$F(n) = n\log_2 n$

所以 $C(n) = n\log_2 n + 1$，其渐进时间复杂度为 $O(n\log_2 n)$

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：

Prim 算法通过依次加入未访问的点，并计算新加入的点与其他未加入的点的距离，如果小，则更新，之后再次加入此时距离最近的点，不断循环，直至加入完所有的点，可见该算法是基于顶点来进行搜索的，因此适合顶点较少而变多的稠密图；而 Kruskal 算法则将所有的边先进行排序，按照从小到大的顺序加入改边两端的顶点，并将它们合并为一个集合，可见该算法是基于边来进行搜索的，因此适合顶点多而变少的稀疏图。

二者的集成，可以通过不断更新边与顶点数量之比，大于某一阈值判定为稠密图，小于该阈值为稀疏图。具体做法是，将已经合并的子图当作一个顶点处理，此时剩余顶点个数为 m-n（m 为所有点个数，n 为已经合并的点个数），同理，边的个数也要不断减去已经合并的边数量，通过不断更新剩余点数和变数，并计算比值，来决定是否切换算法。

我的评价是，几乎没有任意一种算法可以完全适应某一问题的全部输入，应当具体问题具体分析，比较不同输入的特点，结合不同算法特性来选取合适算法，亦或者将不同算法进行整合。

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：

正确性分析：

设数组 A 长度为 k，*HeapPermute(n)*算法表示生成数组 A 前 n 位的全排列与后 k 位的组合，当 n==k 时，算法输出了数组 A 完整的全排列。且当 n 为奇数时，输出完一组全排列后，前 n 位数组复原，若 n 为偶数时，输出完一组全排列后，前 n 位数组循环右移一位。

现使用归纳法进行证明。

1、 当 k==1 时，算法输出 A 的全排列，即[a1]，满足推论。
2、 当 k==2 时，算法输出 A 的全排列，即[a1,a2]，[a2,a1]，发现数组循环右移了 1 位，满足推论。
3、 当 k==3 时，算法输出 A 的全排列，即[a1,a2,a3]，[a2,a1,a3]，[a3,a1,a2]，[a1,a3,a2]，[a2,a3,a1]，[a3,a2,a1]，且查看原数组，发现顺序没有发生变化，也满足推论。
4、 当 n==2p-1,n<k 时，n 为奇数，输出 A 的前 n 位全排列并与 A 的后(k-(2p-1))组合，当输出完一轮全排列时，前 n 位数组复原，此时程序返回到 **for** *i* ←1 **to** *n* **do** 处执行，由于此时 n=2p，故执行 swap *A*[i]and *A*[*n*]，即前(2p-1)位的元素依次与第 2p 位元素进行置换，并且每轮生成该轮的全排列。
5、 当 n==2p，n<k 时，n 为偶数，继第 3 个推导，输出完前 n 位的全排列后，前 n 位元素循环右移一位，满足推论。

时间复杂度分析：

根据交换操作次数进行分析，有如下公式：

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

推导可以得出 T(n)= n! + (n!+ … + n(n-1)(n-2) + n(n-1) + n)

故总体时间复杂度为 $O(n^{n-1})$。

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：

算法思想：将 n 均分为两部分（若不能均分则取整，一边比另一边多一个），分别求两边最小元素及其位置，通过比较二者元素的大小，得到最小元素的位置。

伪代码：

```
(x,i) = FindLeastElement(A[], a, b)
//从数组 A[1…n]中寻找最小元素及其位置
//输入：数组 A[1…n]，左下标 a，又下标 b，
//输出最小元素及其位置
if a=b
    return (A[a],a)
else
    (x1,i) = FindLeastElement(A, a, (a+b)/2)
    (x2,j) = FindLeastElement(A, (a+b)/2+1, b)
    if(x1 < x2)
        return (x1,i)
    else
        return (x2,j)
    endif
endif
```

时间复杂度分析：

$$
\begin{aligned}
F(n) &= 1 && n = 1\\
&= 2F(n/2) + 1 && n > 1\\
&= 2[2F(n/4) + 1] + 1\\
&= \ldots\\
&= 2^kF(n/2^k) + 1 + 2 + 4 + \ldots + 2^{k-1}
\end{aligned}
$$

因为 $n=2^k$，根据等比数列求和的最终结果 $F(n) = 2n-1$

而蛮力算法时间复杂度为 n，由于减治算法每个子问题的时间复杂度依然与 n 成线性关系，故该减治思想并没有“捡到便宜”，反而因为每次调用结束后还要返回来继续比较一次，所以比较次数比蛮力算法更多。

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：

设问题开始时有 n 个人，报数为第 m 的人将被杀掉，求最后的幸存者编号。推导过程如下：

若有 2n 个人，即人数为偶数，且 m=2，则第一轮报数为偶数的人将被杀掉，则：

1	2	3	4	5	6	7	8	9	10
1		2		3		4		5	

第一行为旧的编号，第二行为新的编号。观察可得旧编号=新编号*2-1，即

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

若有 2n+1 个人，即人数为奇数，且 m=2，则第一轮报数为偶数的人将被杀掉，则：

1	2	3	4	5	6	7	8	9
		1		2		3		4

第一行为旧的编号，第二行为新的编号。观察可得旧编号=新编号*2+1，即

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

综上，约瑟夫斯问题的递推公式如下：

$J(1) = 1$

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

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

将上述推导整理结果可得

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

表中数据以 2 的幂次为一组，即每一组数据都比上一组数据多 2 倍，则可将 n 表示为 2^m+b，m 是使得 2^m 不超过 n 的最大次幂的数，i 表示分组的次序。则此时非递推公式可表示为：

令 $n = 2^m+b$, $J(n) = 2*b+1$, $(m>=0 , 0<=b<2^m)$

证明如下：

1） 若 i=1，则 m=0,b=0,J(1) = 2*0+1=1，成立

2） 若 i>1，当 i 为偶数时，设 n= i/2 成立，即 $n=2^m + b$，则 J(n)=2b+1，此时 $i=2n=2^{m+1}+2b$，J(2n)=2J(n)-1=2(2b+1)-1=4b+1=2(2b)+1,成立。

3） 若 i>1，当 i 为奇数时，设 n=(i-1)/2 成立，即 $n=2^m + b$，则 J(n)=2b+1, 此时 $i=2n+1=2^{m+1}+2b+1$，J(2n+1)=2J(n)+1=2(2b+1)+1=4b+3=2(2b+1)+1,成立。