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 $a\neq c^x$

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？

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]
```

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。

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。

一、递推式的渐进复杂度

定理1的证明：

无论 $f(n)=\begin{cases} d, & n=1 \\ af(\frac{n}{c})+bn^x, & n>1 \end{cases}$

其中a、c为非负整数，b、d、x为非负常数。

则 $f(n)=af(\frac{n}{c})+bn^x$

$=a(af(\frac{n}{c^2})+b(\frac{n}{c})^x)+bn^x$

$=a^2f(\frac{n}{c^2})+abn^x\cdot\frac{1}{c^x}+bn^x$

$=\cdots$

$=a^kf(\frac{n}{c^k})+bn^x\cdot\sum_{j=0}^{k-1}(\frac{a}{c^x})^j$

取其中 $A=\sum_{j=0}^{k-1}(\frac{a}{c^x})^j$，则 $a=c^x$ 时，$A=k$。

而 $a\neq c^x$ 时，$A=\frac{1-(\frac{a}{c^x})^k}{1-\frac{a}{c^x}}$。

取 $n=c^k$，则 $k=\log_c n$，且 $f(\frac{n}{c^k})=f(1)=d$。

则 $a=c^x$ 时，$f(n)=a^k\cdot d+bn^x\cdot k$

其中 $a=c^x$，$n=c^k$，则 $a^k=(c^x)^k=(c^k)^x=n^x$

则 $f(n)=n^x\cdot d+b\cdot n^x\cdot\log_c n=n^x(d+b\cdot\log_c n)$

而 $a\neq c^x$ 时，$f(n)=a^k\cdot d+\frac{bn^xc^x\cdot a^k}{(a-c^x)\cdot c^{kx}}-\frac{bn^xc^x}{a-c^x}$

$=(d+\frac{bc^x}{a-c^x})\cdot a^k$

其中 $k=\log_c n$，则 $a^k=a^{\log_c n}=n^{\log_c a}$

则 $f(n)=(d+\frac{bc^x}{a-c^x})\cdot n^{\log_c a}$

综上：$f(n)=\begin{cases}(d+b\log_c n)\cdot n^x, & a=c^x \\ (d+\frac{bc^x}{a-c^x})\cdot n^{\log_c a}, & a\neq c^x\end{cases}$

则对具体的 $C(n)=\begin{cases}1, & n=1 \\ 2C(n/2)+n-1, & n\geq 2\end{cases}$ 有：

计算得递归地：$C(n)=(n-1)\log_2 n+n$ ①

而直接代入定理1有：$a=2$，$b=1$，$c=2$，$d=1$，$x=1$。

则 $a=c^x$，即 $C(n)=(\log_2 n+1)\cdot n=n\log_2 n+n$ ②

由①②有递推计算和经过定理1代入有相同结果。

$C(n)$ 的渐进复杂度为 $O(n\log n)$。

二、Prim 与 Kruskal 算法的集成

1. 集成的方法

Prim 算法基于顶点进行搜索，所以适合顶点较少的稠密图；而 Kruskal 算法基于边进行搜索，所以适合边较少的稀疏图，因此对于具体问题可以根据图的稀疏稠密程度选择相应的算法进行处理。

2. 集成的意义

没有一个算法是万能的，任何算法都要适合具体面对的情况才能发挥出最优的效率，而这种针对具体的图问题选择合适算法的集成方法就体现了这种具体问题具体分析的哲学思想，有利于充分发挥算法的效率，从而更好地解决具体问题。

三、分析以下全排列算法的正确性与时间效率

1> 算法正确性分析：

首先分析模拟算法运行过程：

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，HeapPermute(2)将 a1a2 做完全排列输出，记为[a1a2]a3，并将 A 变为 a2a1a3，并交换 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

即全部输出完毕后数组 A 回到初始顺序

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

即全部输出完毕后数组 A 循环右移一位

由以上分析可得出结论：

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

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

所以由归纳法证明如下：

(1) (1)i=1 时，显然成立。

(2) (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) (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。

综上所述，该算法可以正确输出全排列。

2> 算法时间复杂度分析：

由递推式：$f(n)=\begin{cases}1, & n=1\\ n[f(n-1)+1], & n>1\end{cases}$ 则：

$$f(n)=nf(n-1)+n$$
$$=n(n-1)f(n-2)+n(n-1)+n$$
$$=n(n-1)(n-2)f(n-3)+n(n-1)(n-2)+n(n-1)+n$$
$$=\cdots$$
$$=n!f(1)+\underbrace{n+n(n-1)+n(n-1)(n-2)+\cdots+n!}_{S(n)}$$
$$=n!+S(n)$$

其中 $S(n)=\frac{n!}{(n-1)!}+\frac{n!}{(n-2)!}+\frac{n!}{(n-3)!}+\cdots\frac{n!}{1!}$

$$=n!\left(\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{(n-1)!}\right)>0.$$

而 $\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!}+\cdots$ 无穷收敛为自然对数e.

则 $S(n)=n!\left(e-\frac{1}{n!}-\frac{1}{(n+1)!}-\cdots\right)$

而 $n!\left(\frac{1}{(n+1)!}+\frac{1}{(n+2)!}+\frac{1}{(n+3)!}+\cdots+\right)$

$$=\frac{1}{n+1}+\frac{1}{(n+1)(n+2)}+\frac{1}{(n+1)(n+2)(n+3)}+\cdots$$
$$<\frac{1}{n+1}+\frac{1}{(n+1)2}+\frac{1}{(n+1)3}+\cdots$$
$$=\frac{1}{n+1}\left(\lim_{m\to\infty}\frac{1-\frac{1}{(n+1)^m}}{1-\frac{1}{n+1}}\right)$$
$$=\frac{1}{n+1}\left(1-\frac{1}{n}\right)\ (n>2)$$
$$=\frac{n-1}{n(n+1)}<1.$$

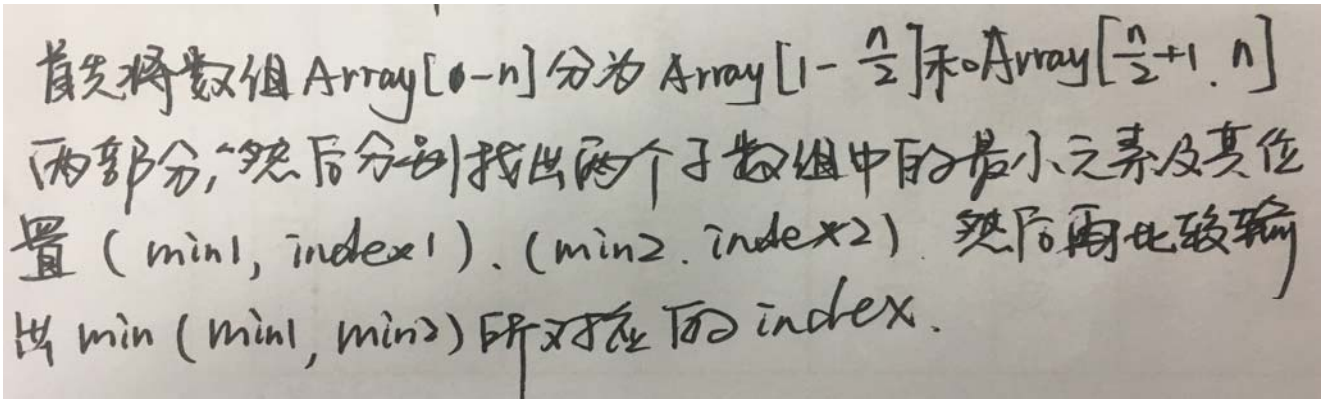
则 $n!e-2<S(n)<n!e-1$

~~则 $S(n)\in(0,1)$~~ 而 $S(n)$ 为整数，则 $S(n)=\lfloor n!e-1\rfloor$

则 $f(n)=n!+\lfloor n!e-1\rfloor$. 则时间复杂度为 $O(n!)$.（其中e为常量）

四、减治思想求数组中的最最小元素 index

(1) 算法设计思想：



(2) 伪代码：

(element, index) = FindLeastElement(array, low, high)

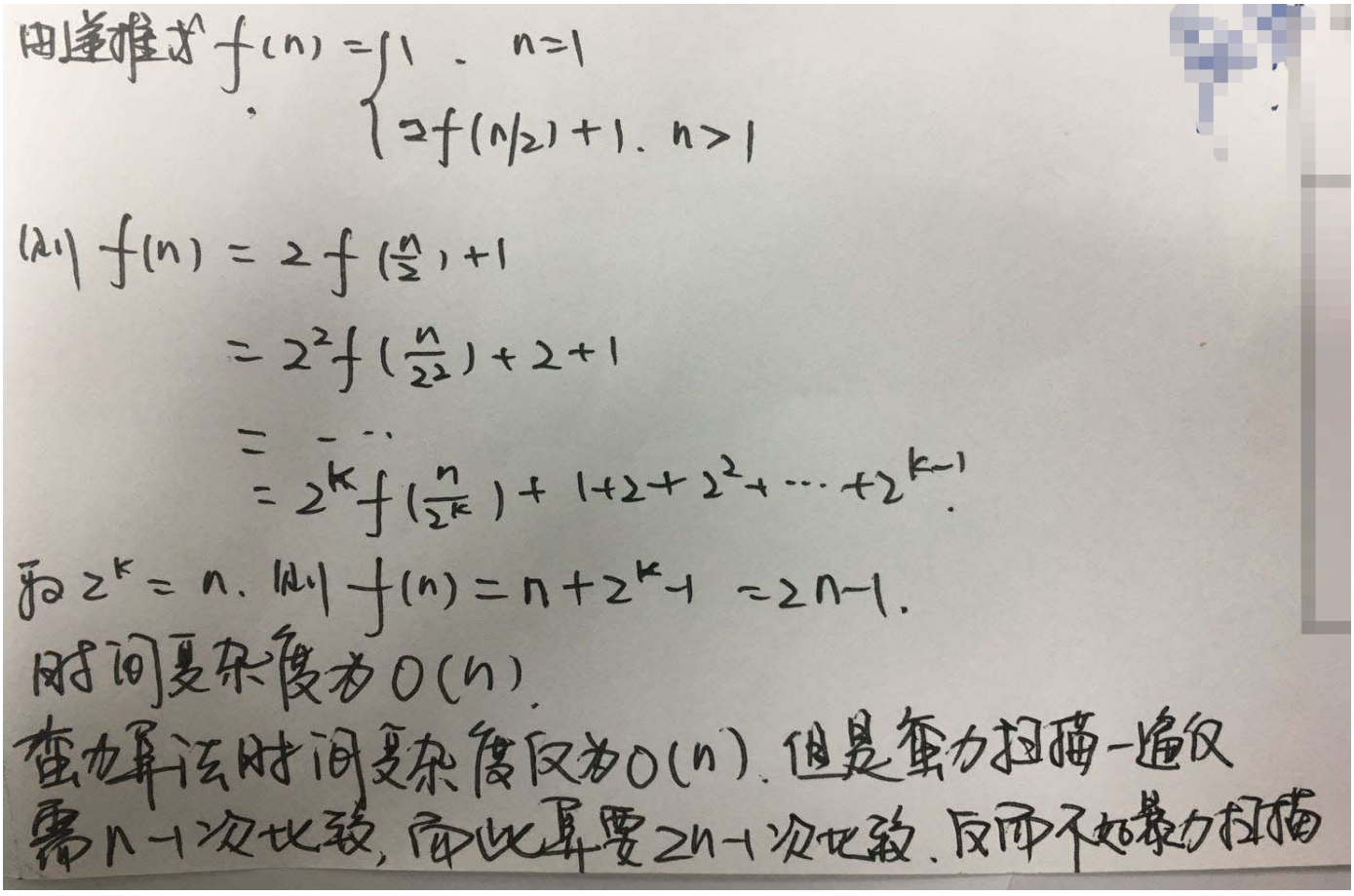
Params: 从数组 array[low, …, high]中找出最小元 element，及其位置 index

Input: 实数数组 A array[low, …, high]，数组起始下标 low，终止下标 high

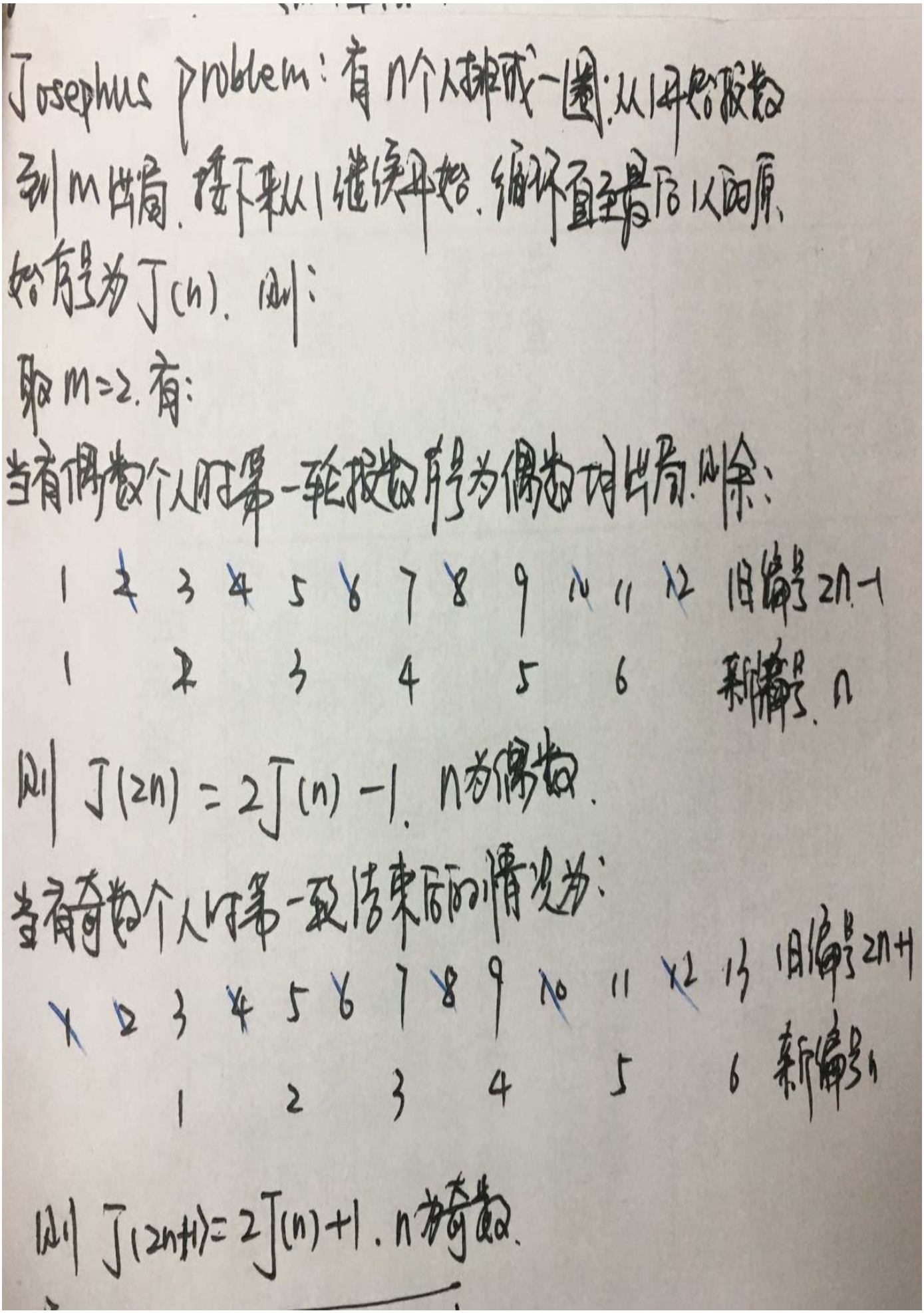
Output: 最小元素 element 及其位置 index

```
If(low==high)
    return （array[low], low）
else
    (element1, index1) = FindLeastElement(array, low, (high+low)/2);
    (element2, index2) = FindLeastElement(array, (high+low)/2+1, high);
    If(element1<element2)
        return (element1, index1)
    else
        return (element2, index2)
```

(3) 时间复杂度分析：



五、证明约瑟夫问题的非递推公式 J(n)



综上有：

$$\begin{cases} J(1)=1 \\ J(2n)=2J(n)-1,\ n\geq 1 \\ J(2n+1)=2J(n)+1,\ n\geq 1 \end{cases}$$

Q1可以快速列出 $J(1)-J(20)$ 有：

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
J(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1	3	5	7	9

观察规律，以2的幂为界是奇数的1. 则可猜想：

$$J(2^m+k)=2k+1,\ m\geq 0 \text{ 且 } 0\leq k<2^m.$$

由数学归纳法证明：

① $m=0$ 时，$J(1)=1$，成立.

② 设 $m>0$ 时，则由归纳假设 $m-1$ 成立.

即 $J(2^{m-1}+k)=2k+1,\ m\geq 0 \text{ 且 } 0\leq k<2^m$.

则 m 时：

1. k 为偶数，则

$$J(2^m+k)=2J\left(\frac{2^m+k}{2}\right)-1=2\times\left(2\cdot\frac{k}{2}+1\right)-1=2k+1.$$

2. k 为奇数，则：

$$J(2^m+k)=2J\left(\frac{2^m+k-1}{2}\right)+1=2\cdot\left(\frac{k-1}{2}\cdot 2\right)+1=2k+1.$$

即 m 时仍成立，则证毕.

$$J(2^m+k)=2k+1,\ m\geq 0 \text{ 且 } 0\leq k<2^m \text{ 成立.}$$

由此可知 $n=2^m+k$. $m\geq 0$ 且 $0\leq k<2^m$ 时.

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

即 n 的二进制表示为: $n=(b_m b_{m-1}\cdots b_1 b_0)_2$.

即 $n=b_m\cdot 2^m+b_{m-1}\cdot 2^{m-1}+\cdots+b_1\cdot 2+b_0$

则有 $b_m=1$ ($n=2^m+k$. $m\geq 0$ 且 $0\leq k<2^m$).

则 $n=(1\ b_{m-1}\ b_{m-2}\cdots b_1 b_0)_2$

$k=(0\ b_{m-1}\ b_{m-2}\cdots b_1 b_0)_2$

$2k=(b_{m-1}b_{m-2}\cdots b_1 b_0\ 0)_2$

$2k+1=(b_{m-1}b_{m-2}\cdots b_1 b_0\ 1)_2$

$J(n)=(b_{m-1}b_{m-2}\cdots b_1 b_0\ 1)_2$

$=(b_{m-1}b_{m-2}\cdots b_1 b_0 b_m)_2$

因为 $b_m=1$.

即 J(n) 的值为 n 的二进制表示循环左移1位.

推广到经数学证明方式也:

n个人编号 1—n. 从1开始报号到 m 出局. 则

J(n.m) 的值为 n 的 m 进制表示循环左移1位.

其递归表示为:

$$\begin{cases} J(n.m)=(J(n-1,m)+m)\%n & (n>1\text{为循环次数}) \\ J(1.m)=0 \end{cases}$$

则 n 个人经 n-1 次循环 ~~[illegible]~~ 报仅余1人.

即问题复杂度从 O(n.m) 归为到了 O(n).

成为一个进制转换问题. 即:

$J(n.m)=J(m^p+k)=m\cdot k+1$.

其中 $m^p\geq 0$ 且 $0\leq k<m^p$.