**2-1.证明Hanoi塔问题的递归算法与非递归算法等价：**

假设塔座a，b，c排列成一个三角形，且按abca的顺序构成一顺时针循环，现要将a座上的圆盘移动到b座上。在非递归算法中，可将移动步骤分为两种：①将最小圆盘移动到下一塔座上（当圆盘个数n为奇数时顺时针移动，记为S；当n为偶数时逆时针移动，记为N）；②保持最小圆盘不动，在其它两个塔座之间，将较小圆盘移动到另一塔座上（可以是空塔座），记为F。上述两种步骤交替执行，就构成了非递归算法。

Hanoi塔问题的递归算法如下:

void hanoi(int n, int a, int b, int c) {

if (n > 0) {

hanoi(n - 1, a, c, b);

move(a, b);

hanoi(n - 1, c, b, a);

}

}

显然，当n=1或n=2时，递归算法与非递归算法的执行步骤一样。

假设n=k-1时，递归算法与非递归算法的执行步骤一样。当n=k时：

若k为奇数，则非递归算法的执行步骤为：S-F-S-F-S-F-……-S。递归算法的执行步骤为：hanoi( k-1, a, c, b)-F- hanoi( k-1, c, b, a)。而hanoi( k-1, a, c, b)表示将k-1个圆盘从a座移动到c座，根据假设的条件，其执行步骤与非递归算法一致，为S-F- S-F-……-S。同理，hanoi( k-1, c, b, a)的执行步骤为S-F- S-F-……-S。故k为奇数时，递归算法与非递归算法的执行步骤一样。

同理可证，k为偶数时，递归算法与非递归算法的执行步骤一样。

综上，由数学归纳法可知，Hanoi塔问题的递归算法与非递归算法等价。

**2.7求解d次多项式P(x)：**

由题目条件可知：

因此，可用分治法将d次多项式分解成两个d/2次多项式的乘积。

故算法时间复杂度的递归表达式为：

解得。

部分代码如下：

#include <iostream>

#define d /\*常数\*/

typedef struct {

int\* coefficient; //多项式系数

int n; //最高项次数

} polynomial;

int zero\_point[d] = {/\*零点集\*/ };

//初始化一次多项式P(x)=x-nd

polynomial\* initialize(int\* zero\_point) {

polynomial\* p = new polynomial[d];

for (int i = 0; i < d; i++) {

p[i].coefficient = new int[2];

p[i].coefficient[0] = -zero\_point[i];

p[i].coefficient[1] = 1;

p[i].n = 1;

}

return p;

}

//求解多项式

polynomial\* poly(polynomial\* p, int low, int high) {

//Mul1()可以在O(i)时间内计算一个i次多项式与一个1次多项式的乘积

//Mul2()可以在O(ilogi)时间内计算两个i次多项式的乘积

polynomial\* ans = NULL;

if (low == high) {

ans = &(p[low]);

}

else if (high - low == 1) {

ans = Mul1(&(p[low]), &(p[high]));

}

else {

int mid = (low + high) / 2;

if ((high - low) % 2 == 0) {

polynomial\* p1 = poly(p, low, mid - 1);

polynomial\* p2 = poly(p, mid, high - 1);

polynomial\* p3 = &(p[high]);

ans = Mul2(p1, p2);

ans = Mul1(ans, p3);

}

else {

polynomial\* p1 = poly(p, low, mid);

polynomial\* p2 = poly(p, mid + 1, high);

ans = Mul2(p1, p2);

}

}

return ans;

}

int main(void) {

polynomial\* p = initialize(zero\_point);

polynomial\* ans = poly(p, 0, d - 1);

/\* …… \*/

return 0;

}

**2.8下标值与元素值相等问题：**

假设n个不同的整数按升序排好后存放在数组中。每次检查数组下标中位数k与其对应的元素值是否相等，若下标大于元素值，则对于,都有,因此只需递归调用算法继续在后半段数组中寻找即可。同理，若下标小于元素值，只需递归调用算法继续在前半段数组中寻找即可。

故算法时间复杂度的递归表达式为：

解得。

代码如下：

#include <iostream>

int index\_select(int\* a, int low, int high) {

int mid = (low + high) / 2;

if (low == high && low != a[low]) {

std::cerr << "non-existent!" << std::endl;

exit(0);

}

else {

if (mid < a[mid])

return index\_select(a, low, mid - 1);

else if (mid > a[mid])

return index\_select(a, mid + 1, high);

else

return a[mid];

}

}

//测试程序

int main(void) {

//按升序排好的整数

int a[10] = { -1, 0, 1, 2, 3, 4, 5, 7, 17, 18 };

std::cout << index\_select(a, 0, 9) << std::endl;

return 0;

}