问题 1：求 n 至少为多大时，n 个 1 组成的整数能被 2013 整除

解答：逐个 n 进行搜索即可。需要注意数的大小，C++的 long long 类型最大支持约 10^18 的整数，超出该范围即溢出，因此我在程序中模拟竖式计算，每次将余数×10+1，从而避免溢出并减少运算量。

最终答案：n = 60

111111111111111111111111111111111111111111111111111111111111 = 2013*551967765082519180879836617541535574322459568361207705 47

求解代码：

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <iostream>

using namespace std;

// res 是商，ans 是 1 的数量，div 是余数
int sample = 1;
int target = 2013;
int cur = 0, ans = 0;              //被除数、1 的数量
int result[10000000], length = 0;  //商、商的长度
int main() {
    // 首先找到第一个比 target 大的整数作为被除数
    while (cur < target) {
        cur = cur * 10 + sample;
        ans++;
    }
    // 每次用被除数对 target 取余，若余数不为 0 再补 1，从而避免溢出
    while (cur % target != 0) {
        result[length++] = cur / target;
        cur = cur % target * 10 + sample;
        ans++;
    }
    result[length++] = cur / target;  //补充商的最后一位
    //输出结果
    cout << "答案为:" << ans << endl;
    for (int i = 0; i < ans; i++) cout << sample;
    cout << "=" << target << "*";
    for (int i = 0; i < length; i++) cout << result[i];
    return 0;
}
```

问题 2：减法版欧几里得算法的伪代码：

解答：

```
// 减法版欧几里得算法，寻找 a,b 的最大公因数
// GCD(a,b) = GCD(a-b,b)
Integer: Gcd(Integer:a, Integer:b)
    while(a != b)
        // 判断大小，用大数-小数
        if(a > b){
            a = a - b;
        }
        else{
            b = b - a;
        }
    End while
    // 返回答案
    Return a
End Gcd
```

问题 3：给定一个排好序的单调递增数组 A[1]…A[n]，元素两两不相等，查找满足 A[i]=i 的下标个数。

解答：对于元素均为整数的情形，可以采用二分法，首先在数组中找到第一个满足 A[i]=i 的下标，记为 L；如果找不到，则 L=n+1。之后在数组中找到第一个满足 A[i]<i 的下标，记为 R，若找不到，则 R=n+1。最终答案为 R-L。

时间复杂度分析：

两次二分查找的时间复杂度均为 O(log(n))，因此算法的时间复杂度为 O(log(n))

求解代码：

```
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
#define random(x) rand() % x
using namespace std;

// 指定数组 A 的成员，n 为搜索长度
int A[1000] = {0, -1, 0, 3, 4, 5, 6, 7, 10, 30, 31}, n = 10;
// 找到第一个 A[i]=i，返回 i;如果没有找到，返回 n+1
int find_first_equal() {
    int a = 1, b = n, mid, ans = n + 1;
    while (a <= b) {
        mid = (a + b) / 2;
        if (A[mid] == mid) {
            ans = mid;
            b = mid - 1;
        } else if (A[mid] > mid) {  // A[i]>i，向左查找
```

```
            b = mid - 1;
        } else {
            a = mid + 1;  // A[i]<i，向右查找
        }
    }
    return ans;
}
// 找到第一个 A[i]>i，返回 i;如果没有找到，返回 n+1
int find_first_bigger() {
    int a = 1, b = n, mid, ans = n + 1;
    while (a <= b) {
        mid = (a + b) / 2;
        if (A[mid] > mid) {  // A[i]>i，向左查找
            ans = mid;
            b = mid - 1;
        } else {  // A[i]<=i，向右查找
            a = mid + 1;
        }
    }
    return ans;
}
int main() {
    int l = find_first_equal(), r = find_first_bigger();
    cout << r - l << endl;
    return 0;
}
```

小数情况的讨论：

如果存在小数，则需要逐个元素进行比对，若遇到 A[i]>i，则可以直接跳转到下标为 ceil(A[i])的位置进行查找，从而略过一些元素并提高算法效率。但是在最坏的情况下，如 A[i]<i 总成立，仍然需要遍历数组进行查找。因此算法的时间复杂度为 O(n)。