A：核心的计算逻辑类似之前的加法计算器（数组按位相乘），但直接使用 for 循环每次乘 2 计算 2^100000 会导致运行时间爆炸，所以必须想办法减少循环次数和乘法计算次数。思路是每次乘一个更大的数 x 来减少次数。unsigned long long int 型存储的最大数值是 2^64，而在按位计算中，x 最大的可能要一次性乘 9（加进位），因此 x 最大只能为 2^60，以免数值溢出。把 n 拆解为 x，y，n=60x+y，执行两个 for 循环来计算。

计算过程第一个循环先将数组的第一个值初始化为 1，每次乘 2，进位最大为 1，可以只用一个循环来完成相乘和进位过程。第二循环每次 for 循环按位乘 2^60，然后再用 for 循环从低位到高位进位，得到结果。可以通过估计位数来决定每个循环执行的次数（即使执行次数过多也不会影响结果，因为数组后面的位都被初始化为 0，但这样会增加时间）。

最后计算出结果的精确位数，倒序打印出来就是结果。

B：此题思路基本固定，所以不做过多解释。整体思路是把数字每三位断开输出。重点是把需要输出的单词存储到二维字符串中，这样可以省去 if 或 switch 判定的冗长代码（使用 for 循环，printf %s, str[i]），注意空格的输出，输出每一个单词后判定之后是否还有内容，如果没有就不再输出空格。

C：核心思路是利用 for 循环依次判定数组中两个相邻的两个元素是否前一个元素大于零且后一个元素小于零，如果符合条件就继续判定这两个相邻元素的绝对值大小，将绝对值较小的元素设为 0（如果相等一起设为 0）。在循环过程中如果碰到 0（已经被删除掉的数字）就把后一个元素继续向后移动，借此跳过已经被删除掉的元素。每次执行完删除操作后跳出这个循环并重新执行，如果遍历整个数组后依然没有可删除的元素则表示战斗已经完成，结束整个进程，打印数组（跳过其中的 0）。

由于判定 0 的过程较为复杂，且时间复杂度高，我使用了链表来实现这个过程（链表的大小根据元素数量决定，且在其中删除元素会直接让整个链表重新连接），避免掉了判定 0 的过

程，降低了时间复杂度。

D：如果一个字符串的所有元素可以排列为连续的，设它的长度为 len，将字符串中的元素两两做差取绝对值，最大的差一定为 len-1，最小的差一定为 1。