Answer 1

A1

1
-103 原码为 01100111，反码为 10011000，补码为 10011001（负数补码为原码的反码 + 1）
+76 原码与补码均为 01001100（正数补码与原码一致）

2
54
-19

A2

1. 最小的补码为 1000 0000，即 -128；最大的补码为 0111 1111，即 127.
2. 位 2 进制补码的范围为 $-2^{N-1} \sim 2^{N-1} - 1.$

A3

-64。（原码为 1100 0000，反码为 1011 1111，补码为 1100 0000）

A4

1. int 型的范围为 $-2^{31} \sim 2^{31} - 1$

当两个正整数 a 和 b 的和超过了 int 类型的最大值时，发生有符号整数溢出，导致结果在二进制补码表示中变成一个负数，因此程序会错误地输出 "Sum is negative"，虽然从数学上看 a + b 实际是正数。

2. 如果将 a 和 b 改为 unsigned int 类型，由于无符号整数的值永远是非负的，即使溢出也只会按模 2^{32} 回绕成另一个正数，所以条件 (a + b < 0) 永远不成立，程序始终输出 "Sum is non-negative"

A5

这个 IEEE 单精度浮点数 0 10000010 10100000000000000000000 的符号位为 0，表示正数；阶码为 130，减去偏移量 127 得到实际指数 3；尾数部分为 1.625。综合计算得到数值为 +1.625 × 2^3 = 13.0，因此它的十进制等价形式是 **13.0**。

A6

最小可以表示的浮点数为 $-(2 - 2^{-23}) \times 2^{127}$，也就是
1 11111110 11111111111111111111111

最小可以表示的正数为 $2^{-23} \times 2^{-126}$，也就是
0 00000000 00000000000000000000001

A7

```c
#include <limits.h>
#include <stdio.h>

union my_union {
    int a;
    float b;
};

int main(void) {
    union my_union t;
    for (int i = INT_MIN; i < INT_MAX; i++) {
        t.b = i;
        if (t.a == i) {
            printf("%d\n", i);
        }
    }
    return 0;
}
```

运行程序，得到结果-834214802，0，1318926965.

A8

1. 答案如下

```c
void swap(int *a, int *b) {
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;
}
```

2. 如果在 sort 函数中调用 swap(a + i, a + min); 时，i == min，也就是说传入了**同一个地址**，这时就会出现问题：

因为当 a 和 b 指向同一个变量时：

```
*a = *a + *a;  // 值被改变成原来的两倍
*b = *a - *b;  // 结果变成 0
*a = *a - *b;  // 结果仍然是 0
```

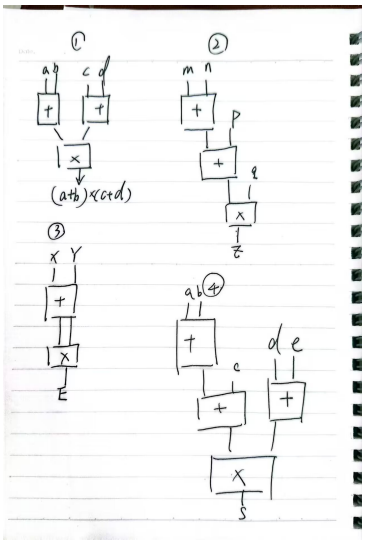
最终导致该元素变成 **0**，数据被破坏。

修正方法：

在调用前加上判断，确保不会对同一个元素进行交换：

```
if (i != min)
    swap(a + i, a + min);
```

A9



A10

1. 共有 64 个字符，因此需要 6 位二进制数来表示。
2. 6N
3. 001000 110110 101100 111110 110110 110100 110110 111001 111111