Problem 1: Integer Representation

(a) 0110 1101
(b) -54
(c) 1110 0001
(d) (any)

Problem 2: Pointers and Arrays

(a) Leave blank or assert(nelems > 0). Although sort of harmless, assert(nelems >= 0)
   is not helpful because its size is unsigned anyway. Code should not be mallocing temp
   space.
(b) nelems * sizeof(int)
(c) copy[i] = *(arr[i]);
   free(arr[i]);
   arr[i] = copy + i;
(d) Leave blank.

Problem 3: Memory Diagram

[Memory diagram diagram showing stack, heap, and read-only data regions with variable aaron, the_other, eliza, and array structures]
Problem 4: Generics and Function Pointers
(a) void remove_less (void *arr, size_t *nelems, size_t width,  
    int (*cmp)(const void *p, const void *q))
{
    // this guards against nelems = 0
    for (size_t i = (*nelems) ? (*nelems - 1) : 0; i > 0; i--)
    {
        void *ith = (char*)arr + i * width;
        int res = cmp(ith, arr);
        if (res < 0)
        {
            memmove(ith, (char*)ith + width, (*nelems - 1 - i) * width);
            *nelems = *nelems - 1; // *nelems--; doesn’t work due to op precedence
        }
    }
}

(b) int farm_compare(const void *p, const void *q)
{
    const struct farm *farm_p = (const struct farm *)p;
    const struct farm *farm_q = (const struct farm *)q;
    return farm_p->count + strlen(farm_p->species) -
           farm_q->count + strlen(farm_q->species);
}

Problem 5: Bitwise Operations
(a) bool zeros_detector_loop(unsigned int n)
{
    unsigned int mask = 0x3; // 0b000....00011
    for (int i = 0; i < 31; i++)
    {
        if (!((n & mask)) return true;
        mask <<= 1;
    }
    return false;
}

(b) // one elegant solution
bool zeros_detector(unsigned int n)
{
    return (~n) & (~n << 1);
}

// a alternate mask-based solution
bool zeros_detector(unsigned int n)
{
    return ((n | (n >> 1)) & 0x7FFFFFFF) != 0x7FFFFFFF;
}