1) (10 pts) DSN (Recursive Coding)

Consider writing a recursive method that raises a polynomial to an exponent, calculating each of its coefficients mod a given integer. One way this method could work is checking to see if the exponent is even. If so, raise the polynomial to half of the original power. Then, take that result (a polynomial) and multiply it by itself for the result. If the original exponent, *exp*, was odd, then we could simply first raise the polynomial to *exp*-1, and then take that result and multiply it by the original polynomial. Implement this algorithm recursively below. You are given the code for the multiply function and should call it accordingly. A polynomial is stored as an array of integers, where poly[i] is the coefficient to x^i. In the function signature, len is the length of the array poly, so poly is of degree len-1, exp is the exponent to which we are raising the polynomial and mod is the modulus by which we are calculating each coefficient.

```
int* power(int* poly, int len, int exp, int mod) {
    if (exp == 0) {
        int* res = malloc(sizeof(int));
        res[0] = 1;
        return res;
    }
    if (exp == 1) {
        int* res = malloc(len*sizeof(int));
        int i;
        for (i=0; i<len; i++) res[i] = poly[i]%mod;
        return res;
    }
    if (exp%2 == 0) {
        int* tmp = power(poly, _______, _______, mod);

        int* prod = multiply(___________, _____________,

                             ___________, ____________, mod)
        free(tmp);
        return prod;
    }
    int* tmp = power(poly, ________, __________, mod);

    int* prod = multiply(_____, ___________________________________,
                         poly, len, mod);
    free(tmp);
    return prod;
}

int* multiply(int* poly1, int len1, int* poly2, int len2, int mod) {
    int* res = calloc(len1+len2-1, sizeof(int));
    int i, j;
    for (i=0; i<len1; i++)
        for (j=0; j<len2; j++)
            res[i+j] = (res[i+j] + poly1[i]*poly2[j])%mod;
    return res;
}
```