```
#include<stdio.h>
#include<math.h>


// at i=28, first_sqrt = 1 by rounding, meaning second_sqrt will equal 0

double original_formula() {
    double previous = 2*sqrt(2);
    for (int i = 2; i < 35; i++){
        double internal_sq = pow(previous/pow(2,i),2);
        double first_sqrt = sqrt(1-internal_sq);
        double second_sqrt = sqrt(2*(1-first_sqrt));
        double last_calc = pow(2,i) * second_sqrt;
        previous = last_calc;
        // int current = i+1;
        // printf("i = %i, an = %f \n", current, previous);
    }
    return previous;
}

double better_formula() {
    double previous = 2*sqrt(2);
    for (int i = 2; i < 35; i++) {
        double top = 2*pow(previous/pow(2,i),2);
        double bottom = 1 + sqrt(1-pow(previous/pow(2,i),2));
        double overall = pow(2,i)*sqrt(top/bottom);
        previous = overall;
        // int current = i+1;
        // printf("i = %i, an = %f \n", current, previous);
    }
    return previous;
}


void main(){
    printf("%f \n", original_formula());
    printf("%f \n", better_formula());
}
```

# Explanation:

3.a: The issue is that at i =28, a rounding error will occur, where the internal square root (double first_sqrt) will round to 1, meaning at the next calculation (double second_sqrt), we will see a catastrophic cancellation, meaning that second_sqrt will equal 0, which is not authentic to the calculation

3.b: To fix the catastrophic cancellation, we need to remove the subtraction fro the function, which we do by rationalizing the equation, multiplying the top and bottom by

$$1+\sqrt{1-\left(\frac{x}{2^n}\right)^2}$$

which removed the subtraction.