Problem 1.
We have a recursive solution to Hanoi tower problem with 3 pegs:
void Hanoi3(peg A, peg B, peg C, int n)
{ if (n == 0)
    return;
    Hanoi3(A,C,B,n-1);
    move a disc from A to B;
    Hanoi3(C,B,A,n-1);
}
and the exact count how many disc movements are made: $T_3(n) = 2^n - 1$.

a) Assuming that each move takes 1 second, what is the number of years needed to execute Hanoi3(0,1,2,64)? Round the answer to the nearest power of 2, e.g “approximately $2^{20}$ years”.

b) Now suppose that we have 4 pegs, and otherwise the same rules. We may have a function $f(n)$ that allows to formulate the following recursive algorithm:
void Hanoi4(peg A, peg B, peg C, peg D, int n)
{ if (n < 3) {
    Hanoi3(A,B,C,n);
    return;
}
    Hanoi4(A,D,B,C,n-f(n));
    Hanoi3(A,B,C,f(n));
    Hanoi4(D,B,A,C,n-f(n));
}
b) Show that if for every $n > 2$ we have $0 < f(n) < n$ then this is a correct algorithm.
c) For $2 < n \leq 20$, find values for $f(n)$ that result in the minimum number of moves by Hanoi4.
d) Suggest a formula for $f(n)$ that results in a low (as low as possible) number of moves needed to move 64 discs with 4 pegs.
e) Find the order of growth of the logarithm of the number of moves that results from your formula. Usually, people use notation like $2^{\Theta(f(n))}$.

Problem 5, page 128.