2.1

- **Baby-Step Giant-Step (BSGS).** The idea is to set $m \approx \sqrt{n}$ (where n is the group order) and precompute the "baby steps" g^j for $j = 0,1,\dots,m-1$. These are stored in a hash/dictionary keyed by the value of g^j. Then one computes "giant steps" of the form $h \cdot g^{-m\,i}$ for $i = 0,1,\dots,m-1$ and checks if each one appears in the baby-step table. A match tells you that

$$g^{m\,i+j} = h,$$

so the discrete log is $x = m\,i + j$. The algorithm runs in time $O(\sqrt{n})$ and uses $O(\sqrt{n})$ memory.

- **Pohlig–Hellman.** This algorithm takes advantage of the factorization of the group order

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k},$$

where each p_i is (typically small) prime. One first solves the discrete-log problem modulo each prime-power $p_i^{e_i}$, which is cheaper because each factor $p_i^{e_i}$ is smaller than n. Finally, all these "partial" solutions are recombined into a single solution modulo n using the Chinese Remainder Theorem. When all prime factors p_i are small, the running time can be much faster than a direct $\sqrt{n}$ method.

2.2
Let the group have order 1031^4. In "number of loop-iterations" (counting each exponentiation or table lookup as constant time):

1. **Brute Force:**
   One would try $x = 0,1,2,\dots$ until finding $g^x = h$.
   - Worst-case steps: $1031^4 \approx 1.13 \times 10^{12}$.
2. **Baby-Step Giant-Step:**
   - Worst-case steps: on the order of $\sqrt{1031^4} = 1031^2 \approx 1.06 \times 10^6$.
3. **Pohlig–Hellman:**
   - Since 1031 is prime and the order is 1031^4, there is effectively just one "prime-power" factor. The Pohlig–Hellman method then also costs on the order of $\sqrt{1031^4} = 1031^2 \approx 1.06 \times 10^6$ steps in the worst case.

2.3.
Even though $\mathrm{p} - 1$ has very large prime factors, our ElGamal-attack code does not need to work in the full group Z_p^*. Instead, we exploit the fact that $\mathrm{p} - 1$ also has a big enough product of small prime-power factors—namely, at least 2^{128}. Concretely, we gather only those prime-powers among the factorization of $\mathrm{p} - 1$ whose product is $\geq 2^{128}$. By restricting all exponentiation into that large "smooth" subgroup, we can apply Pohlig–Hellman (which works quickly on smooth-order groups) to compute the discrete log of $\mathrm{pk} = \mathrm{g}^{\mathrm{sk}}$.

Once the subgroup has order at least 2^{128}, the discrete log in that subgroup for an exponent $< 2^{128}$ must be the true secret key sk. Thus the overall “ElGamal Attack” function from Problem 1.5 effectively calls Pohlig–Hellman as a subroutine on the subgroup of size at least 2^{128} (instead of the entire $p - 1$) to recover the 128-bit key.

2.4.
If p is a large safe prime, then $p - 1 = 2q$ where q is itself prime (and large). That means the group Z_p^* is of prime order q (up to the small factor 2). Since q is large—typically hundreds of bits—there is no sizeable smooth factor you can peel off to form a subgroup of order $\geq 2^{128}$. In other words, if you want a subgroup of order at least 2^{128}, you are forced to use nearly all of q, and hence to solve a discrete-log problem in a prime-order group of size $\approx$ q.

But Pohlig–Hellman is not efficient when the group order is dominated by a single large prime: you would end up needing generic $\sqrt{q}$ steps to solve the discrete log. Therefore, the Problem 1.5 attack—relying on finding a big smooth subgroup—is no longer feasible. It will not efficiently recover the secret key if p is a large safe prime.