looking for all primes p with distance 1 from a power of two, i.e., numbers of the form

$$p = 2^k \pm 1, \quad \text{with } 2 \le p \le n$$

These include Mersenne candidates $(2^k - 1)$ and Fermat-like candidates $(2^k + 1)$.

(a) Representative test example

Let $n = 300$.
Powers of two up to 301: $2,4,8,16,32,64,128,256$.

Candidates $2^k \pm 1$ within $[2, n]$:

$$3,5,7,15,17,31,33,63,65,127,129,255,257$$

Primes among these (by inspection or primality test):

$$\boxed{[3,5,7,17,31,127,257]}$$

Since $257 \le 300$, it's included.

also $n = 100$ → $\boxed{[3,5,7,17,31]}$.

(b) Intuitive algorithm (idea)

Instead of checking all numbers up to n, exploit the structure:

1. Generate powers of two: $2,4,8, \dots$ up to slightly above n.
2. For each power $P = 2^k$, consider candidates $P - 1$ and $P + 1$.
3. If a candidate is within $[2, n]$ and is **prime**, add it to the answer.
4. Avoid duplicates (e.g., $3 = 2^2 - 1 = 2^1 + 1$).
5. Return the sorted resulting list.

This is efficient because there are only $O(\log n)$ powers of two ≤ n.

(c) Pseudocode (language-agnostic)

```
sqlCopy codeFUNCTION IS-PRIME(x):
    IF x < 2: RETURN FALSE
    IF x = 2 OR x = 3: RETURN TRUE
    IF x MOD 2 = 0: RETURN FALSE
    d ← 3
    WHILE d * d ≤ x:
        IF x MOD d = 0: RETURN FALSE
        d ← d + 2
    RETURN TRUE

FUNCTION PRIMES_NEAR_POWERS_OF_TWO(n):
    RESULT ← empty list
    SEEN ← empty set   // to avoid duplicates like 3

    P ← 2              // start at 2^1
    WHILE (P - 1 ≤ n) OR (P + 1 ≤ n):
        FOR EACH C IN [P - 1, P + 1]:
            IF 2 ≤ C ≤ n AND IS-PRIME(C) AND (C NOT IN SEEN):
                APPEND C TO RESULT
                ADD C TO SEEN
        P ← 2 * P      // next power of two

    SORT RESULT IN ASCENDING ORDER
    RETURN RESULT
```

If you want to include $2 = 2^0 + 1$, initialize `P ← 1` instead.

(d) Asymptotic worst-case running time

- There are $\lfloor \log_2 n \rfloor$ powers of two up to n. We test **two** candidates per power: $\approx 2\log_2 n$ candidates.
- Using trial division primality testing up to $\sqrt{p}$ gives cost $O(\sqrt{p})$ per candidate.
- Total time:

  $\sum_{k=1}^{\lfloor \log_2 n \rfloor} O\left(\sqrt{2^k}\right) = O\left(\sum 2^{k/2}\right) = O\left(\sqrt{n}\right)$
- Space: $O(1)$ auxiliary (besides output).

Comparison: A sieve of Eratosthenes up to n would be $O(n \log \log n)$ time and $O(n)$ space—overkill here since we only need $O(\log n)$ candidates.

(e) Proof sketch of correctness

- **Soundness:** The algorithm only includes numbers that (i) equal $2^k \pm 1$ for some k and (ii) pass a correct primality test—therefore every returned number is a prime at distance 1 from a power of two and lies in $[2, n]$.
- **Completeness:** Suppose p is any prime with $|p - 2^k| = 1$ and $2 \leq p \leq n$. For that k, the loop considers $2^k \pm 1$, so it will test p. Since p is prime, it will be included. Hence all valid primes are discovered.
- **No duplicates:** Using SEEN prevents returning the same prime twice (notably $3 = 2^2 - 1 = 2^1 + 1$).

(f) Python implementation (readable & pythonic)

```
pythonCopy codeimport math
from typing import List

def is_prime(x: int) -> bool:
    """Deterministic primality test by trial division."""
    if x < 2:
        return False
    if x in (2, 3):
        return True
    if x % 2 == 0:
        return False
    limit = int(math.isqrt(x))
    d = 3
    while d <= limit:
        if x % d == 0:
            return False
        d += 2
    return True

def primes_near_powers_of_two(n: int, include_two_power_zero: bool = False) -
> List[int]:
    """
    Return primes p with |p - 2^k| = 1 and 2 ≤ p ≤ n.
    By default uses k ≥ 1 (i.e., powers 2,4,8,...).
    Set include_two_power_zero=True to also allow k = 0 (which can include p
= 2).
    """
    result = []
    seen = set()

    # Choose starting power of two
    power = 1 if include_two_power_zero else 2
```

```
    while (power - 1) <= n or (power + 1) <= n:
        for cand in (power - 1, power + 1):
            if 2 <= cand <= n and cand not in seen and is_prime(cand):
                result.append(cand)
                seen.add(cand)
        power *= 2

    result.sort()
    return result

# --- quick demo ---
if __name__ == "__main__":
    for N in (100, 300):
        print(N, "->", primes_near_powers_of_two(N))
```

Readability tweaks (docstrings, clear names, helper function, `math.isqrt`) do **not** change the asymptotic running time: it remains $O(\sqrt{n})$ time and $O(1)$ extra space (plus output). They only improve constants and clarity.

Optional alternative: Sieve-based (only if you already need many primes)

If you already have a sieve or need many prime queries, you could:

1. Sieve all primes up to n in $O(n \log \log n)$.
2. For each k, test whether $2^k \pm 1$ is in the prime set.

This uses more time & memory for this specific task but is handy in broader contexts.

If you tell me whether your instructor allows $2^0 = 1$ as a “power of two”, I’ll tailor the final answer/output format accordingly.
