Fast Sieve

```cpp
const int N = 1e8;
const int prime_count_estimate = 6e6; // ~N/ln(N) for N=1e8
vector<int> primes(prime_count_estimate);
vector<char> is_prime(N / 2 + 1, true);  // +1 to avoid bounds issues

void fast_sieve() {
    int now = 0;
    primes[now++] = 2;

    for (int i = 3; i <= N; i += 2) {
        if (is_prime[i >> 1]) {
            primes[now++] = i;
            if ((long long)i * i <= N) {
                // Increment by 2*i to skip even multiples
                for (int j = i * i; j <= N; j += 2 * i) {
                    is_prime[j >> 1] = false;
                }
            }
        }
    }
    primes.resize(now); // Trim to actual size
}
```

Segmented Sieve

```cpp
vector<ll> segmented_sieve(ll L, ll R) {
    vector<char> is_prime_range(R - L + 1, 1); // Assume all numbers in range are prime

    if (L == 1) is_prime_range[0] = 0; // 1 is not a prime

    for (int p : primes) {
        if (1LL * p * p > R) break;

        // First multiple of p >= L
        ll start = max(p * p, ((L + p - 1) / p) * p);

        for (ll j = start; j <= R; j += p)
            is_prime_range[j - L] = 0;
    }

    vector<ll> seg_primes;
    for (ll i = L; i <= R; ++i)
        if (is_prime_range[i - L])
            seg_primes.push_back(i);

    return seg_primes;
}
```