SIEVE OF ERATOSTHENES

1. BASIC SIEVE:

**int n;**

**vector<char> is\_prime(n+1, true);**

**is\_prime[0] = is\_prime[1] = false;**

**for (int i = 2; i <= n; i++) {**

**if (is\_prime[i] && (long long)i \* i <= n) {**

**for (int j = i \* i; j <= n; j += i)**

**is\_prime[j] = false;**

**}**

**}**

1. SIEVING TILL ROOT:

**int n;**

**vector<char> is\_prime(n+1, true);**

**is\_prime[0] = is\_prime[1] = false;**

**for (int i = 2; i \* i <= n; i++) {**

**if (is\_prime[i]) {**

**for (int j = i \* i; j <= n; j += i)**

**is\_prime[j] = false;**

**}**

**}**

1. BLOCK SIEVING:

**int count\_primes(int n) {**

**const int S = 10000;**

**vector<int> primes;**

**int nsqrt = sqrt(n);**

**vector<char> is\_prime(nsqrt + 1, true);**

**for (int i = 2; i <= nsqrt; i++) {**

**if (is\_prime[i]) {**

**primes.push\_back(i);**

**for (int j = i \* i; j <= nsqrt; j += i)**

**is\_prime[j] = false;**

**}**

**}**

**int result = 0;**

**vector<char> block(S);**

**for (int k = 0; k \* S <= n; k++) {**

**fill(block.begin(), block.end(), true);**

**int start = k \* S;**

**for (int p : primes) {**

**int start\_idx = (start + p - 1) / p;**

**int j = max(start\_idx, p) \* p - start;**

**for (; j < S; j += p)**

**block[j] = false;**

**}**

**if (k == 0)**

**block[0] = block[1] = false;**

**for (int i = 0; i < S && start + i <= n; i++) {**

**if (block[i])**

**result++;**

**}**

**}**

**return result;**

**}**

1. PRIMES IN RANGE:

**vector<bool> segmentedSieve(long long L, long long R) {**

**// generate all primes up to sqrt(R)**

**long long lim = sqrt(R);**

**vector<bool> mark(lim + 1, false);**

**vector<long long> primes;**

**for (long long i = 2; i <= lim; ++i) {**

**if (!mark[i]) {**

**primes.emplace\_back(i);**

**for (long long j = i \* i; j <= lim; j += i)**

**mark[j] = true;**

**}**

**}**

**vector<bool> isPrime(R - L + 1, true);**

**for (long long i : primes)**

**for (long long j = max(i \* i, (L + i - 1) / i \* i); j <= R; j += i)**

**isPrime[j - L] = false;**

**if (L == 1)**

**isPrime[0] = false;**

**return isPrime;**

**}**

1. WITHOUT GENERATING PRIMES (PRIMES IN RANGE):

**vector<bool> segmentedSieveNoPreGen(long long L, long long R) {**

**vector<bool> isPrime(R - L + 1, true);**

**long long lim = sqrt(R);**

**for (long long i = 2; i <= lim; ++i)**

**for (long long j = max(i \* i, (L + i - 1) / i \* i); j <= R; j += i)**

**isPrime[j - L] = false;**

**if (L == 1)**

**isPrime[0] = false;**

**return isPrime;**

**}**