Prime factorization

- get the prime factors of a number
  main idea
- every composite number(not-prime) have at least 1 prime below sqrt(n) so we check until sqrt(n)
- if the remaining is not 1 then the number is prime

time complexity -- > $Sqrt(n)$

code

```
vector<int> prime_factorization(int num) {
    vector<int> factors;
    if (num < 2) return factors;
    while (!(num % 2)) {
        factors.push_back(2);
        num /= 2;
    }
    for (ll i = 3; i * i <= num; i += 2) {
        while (!(num % i)) {
            factors.push_back(i);
            num /= i;
        }
    }
    if (num > 1) factors.push_back(num);
    return factors;
}
```

Count_divisors

- count the number of divisors
  we just iterate until sqrt(n)
  time complexity --> Sqrt(n)

code

```
int count_divisors(int num) {
    int i, counter = 0;
    for (i = 1; i * i < num; i++) {
        if (!(num % i)) counter += 2;
    }
    if ((i * i) == num) counter++;
    return counter;
}
```

by prime fact

- number of divisors = pow1+1 *pow2+1* ..
  ex : 12 = 2^2 *3^1 -> no div = 2+1* 1+1 = 6

---

get_divisors

- get the divisors of a number
- we just iterate until sqrt(n)
  time complexity --> Sqrt(n)

code

```
vector<int> get_divisors(int num) {
    vector<int> divisors;
    ll i;
    for (i = 1; i * i < num; i++) {
        if (!(num % i)) {
            divisors.push_back(i);
            divisors.push_back(num / i);
        }
    }
    if((i * i) == num) divisors.push_back(i);
    return divisors;
}
```

bonus

- number of divisors = pow1+1 *pow2+1* ..
  ex : 12 = 2^2 *3^1 -> no div = 2+1* 1+1 = 6

multiples loop - get divisors

```
const int N = 1000005;
vector<int> dv[N];
void get_divisors() {
    for(int i=1;i<N;++i)
    {
        for(int j=i;j<N;j+=i)
        {
            dv[j].push_back(i);
        }
    }
}
```