题意

众所周知，汀老师是XDUACM实验室最优秀的人，无论是学习还是打游戏。今天他突然想到一个好玩的游戏。规则是这样的，在游戏中他要得到n个小国，初始的时候小国和小杰各有1个。经过了很久的修炼，汀老师学会了两种魔法，他每次可以动用自己的智慧来使用魔法。
第一个魔法：（小杰变小国）可以使用自己的智慧，复制和当前小杰一样数量的小国出来；
第二个魔法：（小国大爆发）可以将当前的小杰变成和小国的数量一样，然后小国的数量倍！
因为汀老师的智力是无限多的，他不关心花掉的智力大小。但是好学的汀老师想尽快得到n个小国，使得能有更多的时间去读paper和打比赛。他想问问你，最少需要使用多少次魔法可以恰好得到n个小国。

分析

dp[x]:生成x个小国需要的最少魔法数
生成x个小国的情况为：[p个小杰,2*p+c*p个小国]（2*p + c*p == x)，即由（p,2p)再使用c次魔法1达到
如果x为质数，那么p == 1 ，他只能由魔法1得到，dp[x] = x-1
否则 令（2+c) 为x的最小因数（一定为质数）
1. 如果2+c == 2 那么p = x/2 ,dp[x] 可以由dp[p] 使用一次魔法2得到 : dp[x] = dp[p] +1;
2. 如果2+c > 2 那么 p = x/ (c+2) dp[x] 可以由dp[2*p] 使用 c次魔法1得到 : dp[x] = dp[2*p] + c
看起来，我们需要预处理出一个素数表，这里可以用线性筛来处理

```
/*每个合数必有一个最小素因子。每个合数仅被它的最小素因子筛去正好一次*/
const int N = 1e6+5;
bool not_prime[N];
int prime[N];
int sss()
{
    int index = 0;
    memset(not_prime,0,sizeof(not_prime));
    for(int i = 2;i<=(int)1e6;i++)
    {
        if( !not_prime[i] ) prime[++index] = i;
        for(int j = 1;j <= index && prime[j]*i <= (int)1e6 ;j++ )
        {
            not_prime[i * prime[j]] = true;
```

```
            if(i % prime[j] == 0) break;
            /* prime数组 中的素数是递增的,当 i 能整除 prime[j]，那么 i*prime[j+1] 这个合数                肯定被 prime[j] 乘以某个数筛掉。
               因为i中含有prime[j], prime[j] 比 prime[j+1] 小。接下去的素数同理。所以不用筛                下去了。在满足i%prme[j]==0这个条件之前以及第一次满足改条件时,pr[j]必定是                        pr[j]*i的最小因子*/
        }
    }
    return index;
}
```

代码

```
#include <bits/stdc++.h>
using namespace std;
#define inc(i,l,r) for(int i = l;i<=r;i++)
#define dec(i,l,r) for(int i = l;i>=r;i--)
const int N = 1e6+5;
int dp[N];
bool not_prime[N];
int prime[N];
int sss()
{
    int index = 0;
    memset(not_prime,0,sizeof(not_prime));
    for(int i = 2;i<=(int)1e6;i++)
    {
        if( !not_prime[i] ) prime[++index] = i;
        for(int j = 1;j <= index && prime[j]*i <= (int)1e6 ;j++ )
        {
            not_prime[i * prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
    return index;
}
void ycl()
{
    int n = sss();
    dp[2] = 1;
    for(int i = 3;i <= (int)1e6;i++)
    {
        if(not_prime[i])
```

```
        {
            int c = 1;
            while(i%prime[c])c++;
            c = prime[c];
            int p = i/c;
            if(c == 2) dp[i] = dp[p] +1;
            else dp[i] = dp[2*p]+(c-2);
        }
        else dp[i] = i-1;
    }
}
int main()
{
    ycl();
    int _,x;scanf("%d",&_);
    while(_--)
    {
        scanf("%d",&x);
        printf("%d\n",dp[x]);
    }
}
```