UCF ICPC Training Camp

Day I: Number Theory

Yongwhan Lim

Tuesday, March 21, 2022

Yongwhan Lim

MIT
Columbia University
icpc .foundation

Education

Stanford University
MIT

Part-time Jobs

Cornell Tech Jacobs Institute
MIT EECS
In lumine tuo videbimus lumen

Full-time Job

Google Research
2σ TWO SIGMA

Workshops

Stanford Engineering | Stanford Computer Forum
Carnegie Mellon University
Harvard
University of California
KAIST 1971
NUS National University of Singapore

Coach/Judge

icpc International Collegiate Programming Contest
icpc.foundation
advancing the art and sport of competitive programming

https://www.yongwhan.io

Yongwhan Lim

- Currently:
  - a **Co-Founder** in a Stealth Mode Startup;
  - ICPC **Internship Director**;
  - Columbia ICPC **Head Coach**;
  - ICPC **Judge** for NAQ and Regionals;
  - **Lecturer** at MIT;
  - **Adjunct** (Associate in CS) at Columbia;

https://www.yongwhan.io

Today’s Format

10:30am ET - 12pm ET	**Lecture & Exercises**
	Sorry for the late start due to flight issues!
12pm ET - 12:45pm ET	**Lunch**
12:45pm ET - 3:45pm ET	**Practice Contest**
	UCF ICPC Training Camp Day 1
4pm ET - 5:20pm ET	**Review**

Success Pathways

- Those who are just starting should focus on the **first half** of problems in Zealot Problem Set. Your main focus should be gaining some experiences with an explicit goal to enjoy the process of solving new problems and potentially making it to the ICPC North America Championship (NAC)!

- Those who are more serious should focus on the **second half** of problems in Zealot Problem Set. Your goal should be making into the World Finals and potentially winning a medal!

Practice Strategy

- If your goal is to get to a rating of **X**, you should practice on problems that are **X + 300** typically, with a spread of 100. So, picking problems within the range of:

  {X + 200, X + 300, X + 400}

  would be sensible!
- So, if you want to target becoming a **red**, which has a lower-bound of 2400, you should aim to solving {2600, 2700, 2800}.
- **(Eventual) Target**: You should focus on solving it for 30 minutes or less!

Practice Strategy

- You should focus on solving each problem for **30 minutes or less**; if you cannot solve any problem with this range, you should consider solving a problem with a lower rating.
- You should aim to solve **10 ~ 15 problems** each day within this range to expect a rank up within a quarter (3 months).
- If you cannot solve a problem, here is a sample recipe you can follow:
  - Look at editorial for hints, and try to solve the problem.
  - Look at editorial for full solutions, and try to solve the problem.
  - Look at accepted solutions, and try to solve the problem.
  - Make sure you look back after two weeks and see if you can solve it.

Request 1:1 Meeting, through Calendly

- Use calendly.com/yongwhan/one-on-one to request 1:1 meeting:
  - Mock Interview
  - Resume Critique
  - Career Planning
  - Practice Strategy
  - ...
- Always inspired by driven students like yourself!
- Since I'd feel honored/thrilled to talk to you, do not feel shy to sign up!!

Lecture

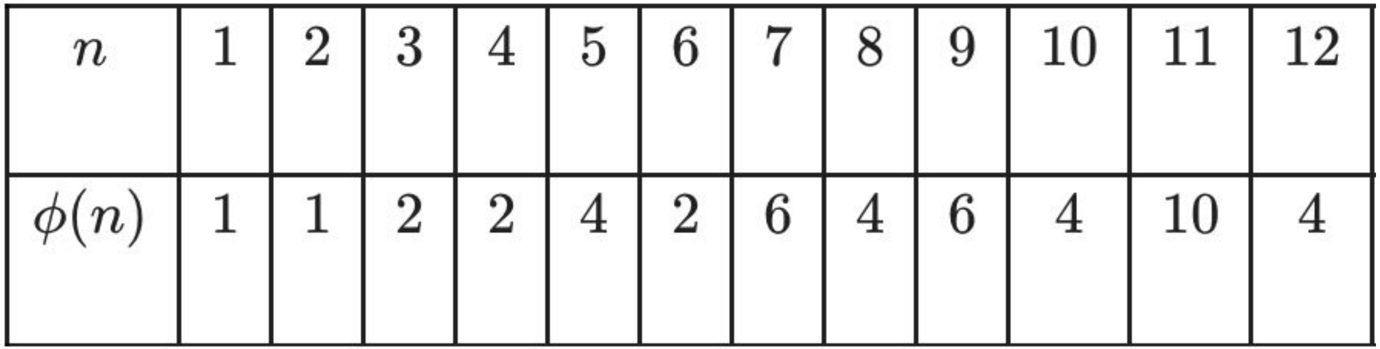
Now, let's dive right into Number Theory!

I. Totient Function
II. Möbius Inversion Formula
III. Fast Fourier Transform

IV. Review: Binary Exponentiation
V. Review: Euclidean Algorithm
VI. Review: Sieve of Eratosthenes

I. Totient Function

- Euler's totient function, also known as ***ϕ*-function** ϕ(n), counts the number of integers between 1 and n inclusive, which are coprime to n.
- Two numbers are **coprime** if their greatest common divisor equals 1.



n	1	2	3	4	5	6	7	8	9	10	11	12
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4

I. Totient Function

$$\phi(p) = p - 1.$$

$$\phi(p^k) = p^k - p^{k-1}.$$

$$\phi(ab) = \phi(a) \cdot \phi(b).$$

$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$$

I. Totient Function

$$
\begin{aligned}
\phi(n) &= \phi({p_1}^{a_1}) \cdot \phi({p_2}^{a_2}) \cdots \phi({p_k}^{a_k}) \\
&= \left({p_1}^{a_1} - {p_1}^{a_1-1}\right) \cdot \left({p_2}^{a_2} - {p_2}^{a_2-1}\right) \cdots \left({p_k}^{a_k} - {p_k}^{a_k-1}\right) \\
&= p_1^{a_1} \cdot \left(1 - \frac{1}{p_1}\right) \cdot p_2^{a_2} \cdot \left(1 - \frac{1}{p_2}\right) \cdots p_k^{a_k} \cdot \left(1 - \frac{1}{p_k}\right) \\
&= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)
\end{aligned}
$$

I. Totient Function: Implementation ($n^{1/2}$)

```
int phi(int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      result -= result / i;
    }
  }
  if (n > 1) result -= result / n;
  return result;
}
```

I. Totient Function: Implementation (n log log n)

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

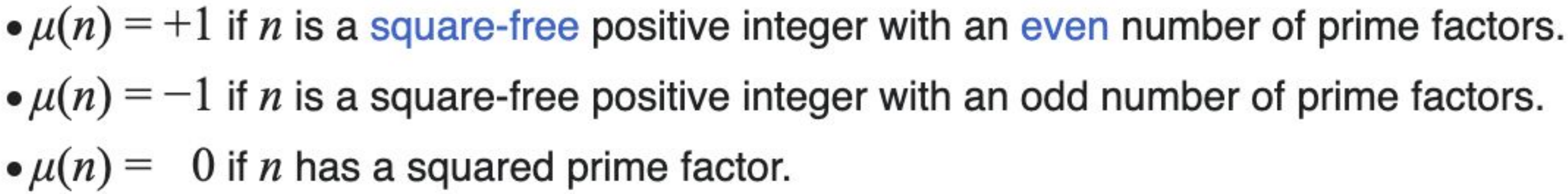
II. Möbius Inversion Formula

$$g(n) = \sum_{d|n} f(d) \quad \text{for every integer } n \geq 1$$

$$\Downarrow$$

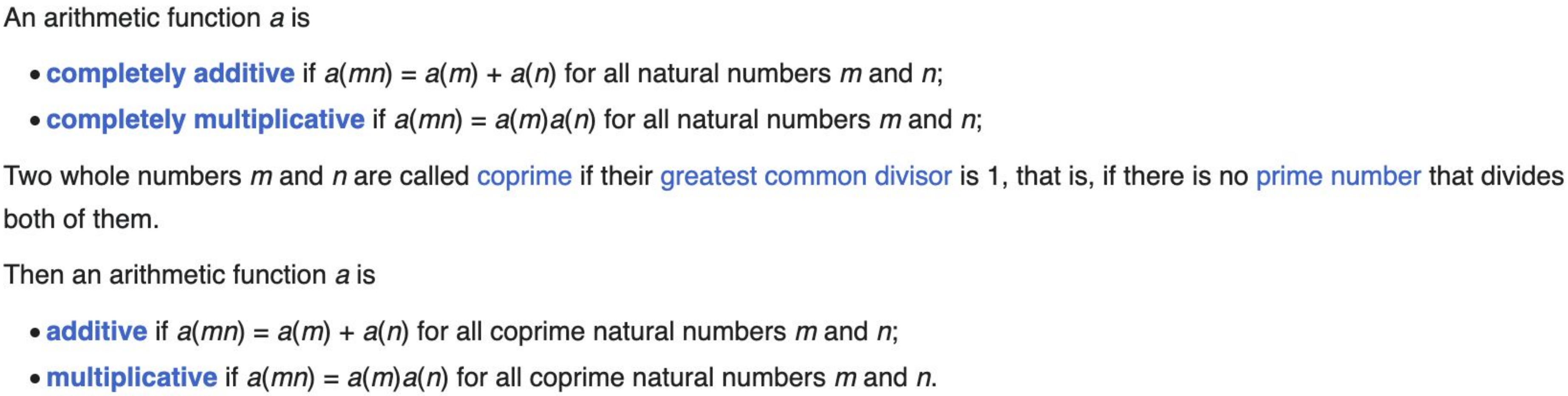
$$f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right) \quad \text{for every integer } n \geq 1$$

II. Möbius Function



- $\mu(n) = +1$ if n is a square-free positive integer with an even number of prime factors.
- $\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors.
- $\mu(n) = \quad 0$ if n has a squared prime factor.

II. Arithmetic Function



An arithmetic function a is

- **completely additive** if $a(mn) = a(m) + a(n)$ for all natural numbers m and n;
- **completely multiplicative** if $a(mn) = a(m)a(n)$ for all natural numbers m and n;

Two whole numbers m and n are called coprime if their greatest common divisor is 1, that is, if there is no prime number that divides both of them.

Then an arithmetic function a is

- **additive** if $a(mn) = a(m) + a(n)$ for all coprime natural numbers m and n;
- **multiplicative** if $a(mn) = a(m)a(n)$ for all coprime natural numbers m and n.

III. Fast Fourier Transform [Cooley and Tukey, 1965]

- **Multiply two polynomials of length `n`** in `O(n log n)` time, which is better than the trivial multiplication which takes O(n^2) time.

III. Discrete Fourier Transform (DFT)

$$w_{n,k} = e^{\frac{2k\pi i}{n}} \qquad\qquad w_{n,k} = (w_n)^k.$$

$$\begin{aligned}
\mathrm{DFT}(a_0, a_1, \ldots, a_{n-1}) &= (y_0, y_1, \ldots, y_{n-1}) \\
&= (A(w_{n,0}), A(w_{n,1}), \ldots, A(w_{n,n-1})) \\
&= (A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1}))
\end{aligned}$$

III. Inverse Discrete Fourier Transform (Inverse DFT)

$$\mathrm{InverseDFT}(y_0, y_1, \ldots, y_{n-1}) = (a_0, a_1, \ldots, a_{n-1})$$

III. Discrete Fourier Transform (DFT)

$$(A \cdot B)(x) = A(x) \cdot B(x).$$

$$\mathrm{DFT}(A \cdot B) = \mathrm{DFT}(A) \cdot \mathrm{DFT}(B)$$

$$A \cdot B = \mathrm{InverseDFT}(\mathrm{DFT}(A) \cdot \mathrm{DFT}(B))$$

III. Fast Fourier Transform (FFT)

$$A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$$

$$A_0(x) = a_0x^0 + a_2x^1 + \cdots + a_{n-2}x^{\frac{n}{2}-1}$$
$$A_1(x) = a_1x^0 + a_3x^1 + \cdots + a_{n-1}x^{\frac{n}{2}-1}$$

$$A(x) = A_0(x^2) + xA_1(x^2).$$

III. Fast Fourier Transform (FFT)

$$\left(y_k^0\right)_{k=0}^{n/2-1} = \mathrm{DFT}(A_0) \qquad \left(y_k^1\right)_{k=0}^{n/2-1} = \mathrm{DFT}(A_1)$$

$$y_k = y_k^0 + w_n^k y_k^1, \quad k = 0 \ldots \frac{n}{2} - 1.$$

III. Fast Fourier Transform (FFT)

$$\begin{aligned} y_{k+n/2} &= A\left(w_n^{k+n/2}\right) \\ &= A_0\left(w_n^{2k+n}\right) + w_n^{k+n/2} A_1\left(w_n^{2k+n}\right) \\ &= A_0\left(w_n^{2k} w_n^{n}\right) + w_n^{k} w_n^{n/2} A_1\left(w_n^{2k} w_n^{n}\right) \\ &= A_0\left(w_n^{2k}\right) - w_n^{k} A_1\left(w_n^{2k}\right) \\ &= y_k^0 - w_n^k y_k^1 \end{aligned}$$

III. Fast Fourier Transform (FFT)

$$\begin{aligned} y_k &= y_k^0 + w_n^k y_k^1, & k &= 0 \ldots \frac{n}{2} - 1, \\ y_{k+n/2} &= y_k^0 - w_n^k y_k^1, & k &= 0 \ldots \frac{n}{2} - 1. \end{aligned}$$

III. Inverse Fast Fourier Transform (Inverse FFT)

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Vandermonde matrix

III. Inverse Fast Fourier Transform (Inverse FFT)

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

III. Inverse Fast Fourier Transform (Inverse FFT)

$$\frac{1}{n}\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^{-1} & w_n^{-2} & w_n^{-3} & \cdots & w_n^{-(n-1)} \\ w_n^0 & w_n^{-2} & w_n^{-4} & w_n^{-6} & \cdots & w_n^{-2(n-1)} \\ w_n^0 & w_n^{-3} & w_n^{-6} & w_n^{-9} & \cdots & w_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{-(n-1)} & w_n^{-2(n-1)} & w_n^{-3(n-1)} & \cdots & w_n^{-(n-1)(n-1)} \end{pmatrix}$$

III. Inverse Fast Fourier Transform (Inverse FFT)

$$a_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j w_n^{-kj}$$

$$y_k = \sum_{j=0}^{n-1} a_j w_n^{kj}$$

III. Fast Fourier Transform: Implementation

```
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1) return;
    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++)
        a0[i] = a[2*i], a1[i] = a[2*i+1];
    fft(a0, invert), fft(a1, invert);
```

III. Fast Fourier Transform: Implementation

```
  double ang = 2 * PI / n * (invert ? -1 : 1);
  cd w(1), wn(cos(ang), sin(ang));
  for (int i = 0; 2 * i < n; i++) {
    a[i] = a0[i] + w * a1[i];
    a[i + n/2] = a0[i] - w * a1[i];
    if (invert)
      a[i] /= 2, a[i + n/2] /= 2;
    w *= wn;
  }
}
```

III. Fast Fourier Transform: Multiplying Two Polynomials

```
vector<int> multiply(vector<int> const& a,
                     vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()),
               fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size()) n <<= 1;
    fa.resize(n), fb.resize(n);
    fft(fa, false), fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, true);
```

III. Fast Fourier Transform: Multiplying Two Polynomials

```
    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

IV. Review: Binary Exponentiation: Main Idea

$$a^n = \begin{cases} 1 & \text{if } n == 0 \\ \left(a^{\frac{n}{2}}\right)^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ \left(a^{\frac{n-1}{2}}\right)^2 \cdot a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$$

IV. Binary Exponentiation: Implementation (Recursive)

```
long long binpow(long long a, long long b) {
    if (b == 0)
        return 1;
    long long res = binpow(a, b / 2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

IV. Binary Exponentiation: Implementation (Iterative)

```
long long binpow(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

V. Review: Euclidean Algorithm

- Given two non-negative integers a and b, we have to find their GCD (greatest common divisor), i.e. the largest number which is a divisor of both a and `b`. It's commonly denoted by `gcd(a,b)`.

$$\gcd(a,b) = \begin{cases} a, & \text{if } b = 0 \\ \gcd(b, a \bmod b), & \text{otherwise.} \end{cases}$$

V. Euclidean Algorithm: Implementation (Recursive)

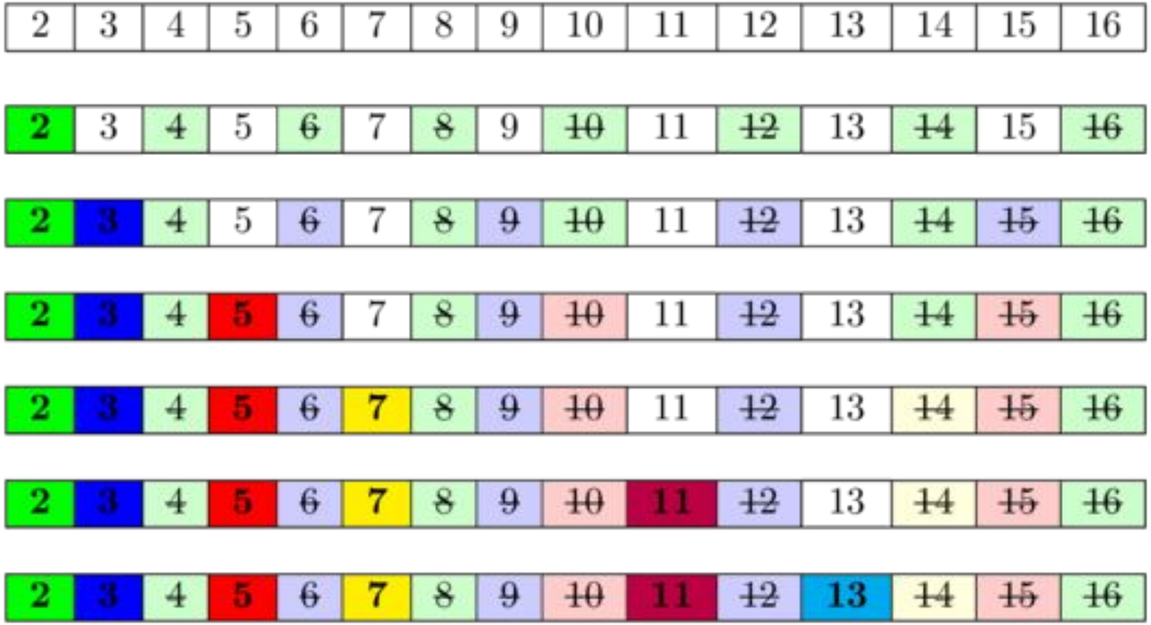
```
int gcd (int a, int b) {
    return b ? gcd (b, a % b) : a;
}
```

V. Euclidean Algorithm: Implementation (Iterative)

- Since C++17, you may use gcd as a standard function in C++.

```
int gcd (int a, int b) {
  while (b) {
    a %= b;
    swap(a, b);
  }
  return a;
}
```

VI. Review: Sieve of Eratosthenes: Main Idea



2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	**11**	~~12~~	13	~~14~~	~~15~~	~~16~~
2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~	**11**	~~12~~	**13**	~~14~~	~~15~~	~~16~~

VI. Sieve of Eratosthenes: Implementation

```
int n;
vector<bool> 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;
```

Further Topics

- Chinese Remainder Theorem
- Discrete Log
- Primitive Root
- Continued Fraction
- ...

Further Readings

- https://cp-algorithms.com/
- https://usaco.guide/
- Terse Guides

- Math (contest) textbooks (for Putnam, etc):
    - *Putnam and Beyond*, Răzvan Gelca and Titu Andreescu
    - *Number Theory*, Georgia E. Andrews
    - *An Introduction to the Theory of Numbers*, Ivan Niven, et. al.
    - *An Introduction to the Theory of Numbers*, G. H. Hardy, et. al.

Discord Server: ICPC CodeForces Zealots

- Join the following discord servers, if you have not already!!!

  [ICPC CodeForces Zealots] https://discord.gg/7bvMnMyF6G

Universal Cup Registration

- Ask Arup Guha to register you in a batch!

https://open.kattis.com/

- Kattis is a good site to train!
- If you have not already, try registering and doing some problems!

Methods to Solve (Steven Halim's CP 4)

- You can get some guided list from: https://cpbook.net/methodstosolve?oj=kattis&topic=all&quality=all

https://cses.fi/problemset/ (Antti Laaksonen's)

- You can get some guided list from CSES!

Practice makes PERFECT!

- Do as **many practice contests** as you can!
  - CodeForces **live contests**
  - **ICPC NA**: **Sundays** from 1pm ET to 6pm ET
  - **Zealot Problem Sets**: **Daily**!

A Terse Guide on ICPC Contest Strategies

- Please take a look at:
  - A Terse Guide on ICPC Contest Strategies for Columbia team.
  - In addition, we have Google Drive to Terse Guides, of course!

- These documents will be frequently expanded upon later.

Lecture Exercises

Lecture Exercise #1

- https://codeforces.com/contest/26/problem/A

Solution

```
int main() {
   bool flag[MAX];
   for (int i=0; i<MAX; i++) flag[i]=true;
   flag[0]=flag[1]=false;
   for (int i=0; i*i<MAX; i++)
      if(flag[i])
      for (int j=i; j*i<MAX; j++) flag[j*i]=false;
   vector<int> p;
   for (int i=0; i<MAX; i++) if(flag[i])
p.push_back(i);
```

Solution (con’t)

```
int sz=p.size(), dp[MAX];
dp[0]=0;
for (int i=1; i<MAX; i++) {
    int cur=0;
    for (int j=0; j<sz; j++) {
        int d=p[j];
        if(d>i) break;
        if(i%d==0) cur++;
    }
    dp[i]=dp[i-1];
    if(cur==2) dp[i]++;
}
```

Solution (con’t)

```
    int n; cin>>n; cout << dp[n] << endl;
    return 0;
}
```

Lecture Exercise #2

- https://codeforces.com/contest/776/problem/B

Solution

- https://codeforces.com/contest/776/submission/24948195

Lecture Exercise #3

- https://codeforces.com/problemset/problem/954/I

Solution

- https://codeforces.com/contest/954/submission/78258286

LUNCH BREAK

THANK YOU