Prime Numbers

and stuff

Factors

- A **factor** of some number *N* is just another number that, when divided with N, leaves no remainder.

6 / 4 = 1 (with remainder of 2) 4 is not a FACTOR of 6

6 / 2 = 3 (with no remainder) 2 is then a FACTOR of 6

Prime Numbers

- A **prime number** is a number *N* which has no positive factors other than 1 and itself.
  - 1 is not a prime number (it's just defined as not prime)

+ve factors of 6: {1, 2, 3, 6} — 6 is not prime

+ve factors of 7: {1, 7} — 7 is prime

+ve factors of 2: {1, 2} — 2 is prime

How many primes are there?

- Not really a surprise, but there are an infinite number of primes.
- BUT, how many primes are there that are less than or equal to some number *x?*
  - $\pi (x)$ = the number of primes $\leq x$
  - $\pi (2) = 1, \pi (3) = 2, \pi (7) = 4$
- It turns out,

$\pi (x)$ is approximately equal to $x / \ln(x)$

x	$\pi(x)$	$\pi(x) - x / \ln x$
10	4	−0.3
10^2	25	3.3
10^3	168	23
10^4	1,229	143
10^5	9,592	906
10^6	78,498	6,116
10^7	664,579	44,158
10^8	5,761,455	332,774
10^9	50,847,534	2,592,592
10^{10}	455,052,511	20,758,029
10^{11}	4,118,054,813	169,923,159
10^{12}	37,607,912,018	1,416,705,193
10^{13}	346,065,536,839	11,992,858,452
10^{14}	3,204,941,750,802	102,838,308,636
10^{15}	29,844,570,422,669	891,604,962,452
10^{16}	279,238,341,033,925	7,804,289,844,393
10^{17}	2,623,557,157,654,233	68,883,734,693,281
10^{18}	24,739,954,287,740,860	612,483,070,893,536
10^{19}	234,057,667,276,344,607	5,481,624,169,369,960
10^{20}	2,220,819,602,560,918,840	49,347,193,044,659,701
10^{21}	21,127,269,486,018,731,928	446,579,871,578,168,707
10^{22}	201,467,286,689,315,906,290	4,060,704,006,019,620,994
10^{23}	1,925,320,391,606,803,968,923	37,083,513,766,578,631,309
10^{24}	18,435,599,767,349,200,867,866	339,996,354,713,708,049,069
10^{25}	176,846,309,399,143,769,411,680	3,128,516,637,843,038,351,228
10^{26}	1,699,246,750,872,437,141,327,603	28,883,358,936,853,188,823,261

Primality Testing

- We're given some arbitrary number *N* and we want to determine if it's prime or not. This is called the primality testing problem.

```
boolean is_prime(int N) {
  for (int i = 2; i < N; i++) {
    if (N % i == 0) return false;
  }
  return true;
}
```

Can we do better? Suggestions.

```
boolean is_prime(int N) {
  for (int i = 2; i < N; i++) {
    if (N % i == 0) return false;
  }
  return true;
}
```

- Our goal is to reduce the number of iterations of the for loop (less iterations means faster code!)

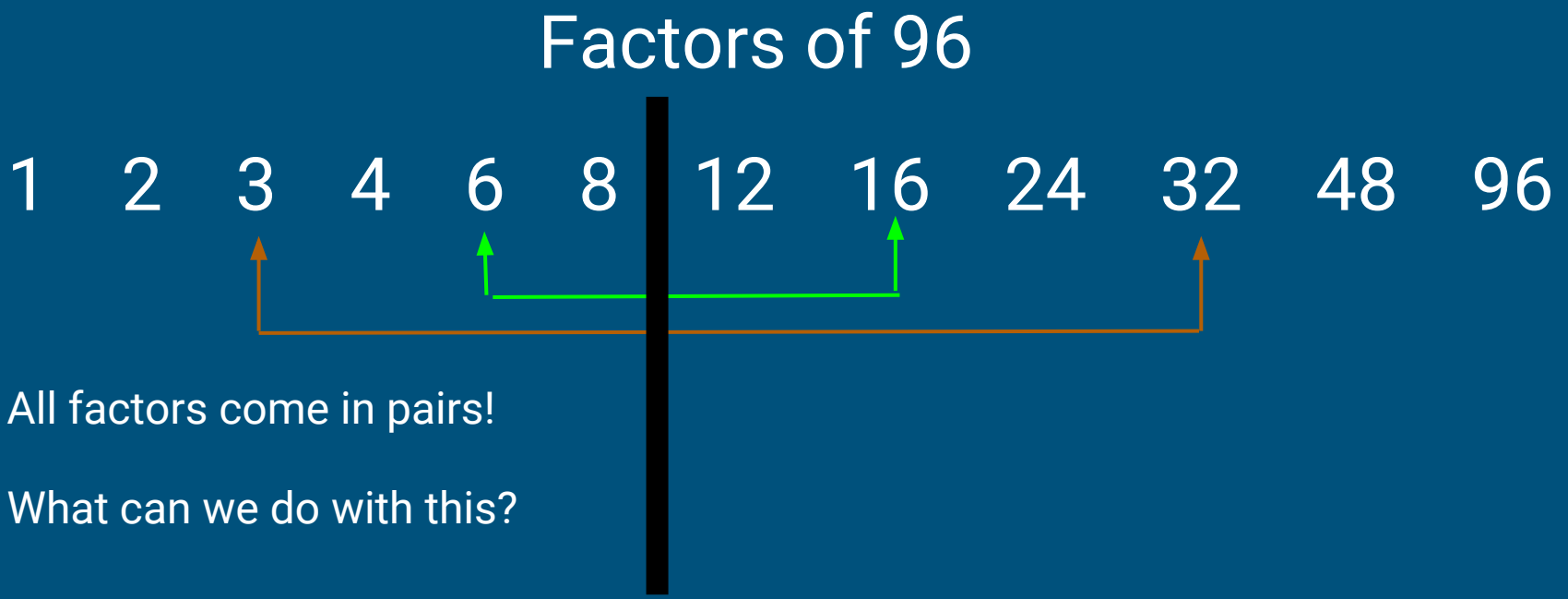
Can we do better? Suggestions.

```
boolean is_prime(int N) {
  if (N == 2) return true;
  if (N % 2 == 0) return false;
  for (int i = 3; i < N; i += 2) {
    if (N % i == 0) return false;
  }
  return true;
}
```

- Suggestion #1: 2 is the only even prime, so why not just handle them as a special case.

  Then, we only have to test odd numbers! This reduces the amount of iterations by half!!!!!

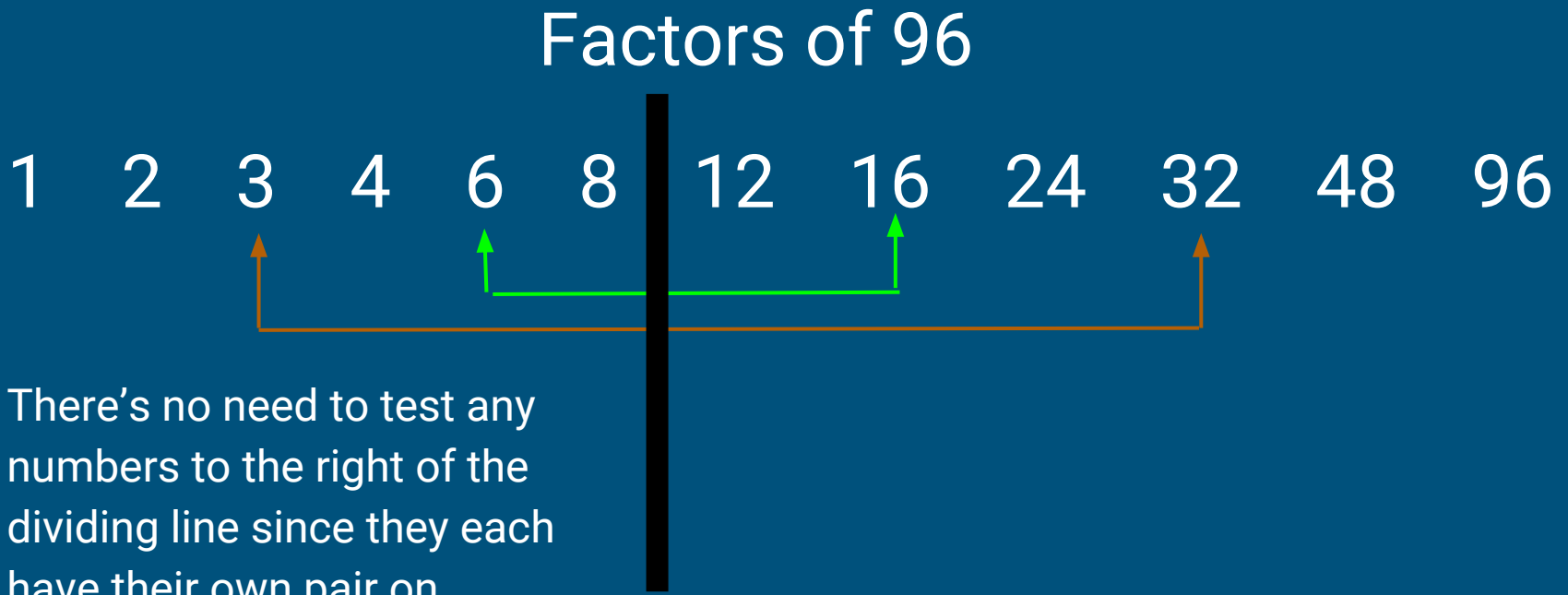
Cool Idea



All factors come in pairs!

What can we do with this?

Cool Idea



There’s no need to test any numbers to the right of the dividing line since they each have their own pair on the left side.

Cool Idea

Factors of x

What is the largest number on the left side?

x/2? x/10? x/ln(x)?

Cool Idea

Factors of x

What is the largest number on the left side?

It turns out, the numbers on the left are all ≤ square-root(x)

Imagine we had some pair (a, b) where a is on the left side and b is on the right. Since they're a pair, we know that

N = a*b

But if we know that a > sqrt(N) and b > sqrt(N),

then

a*b > sqrt(N)*sqrt(N)

or

a*b > N which makes no sense lol

so one of them must be <= sqrt(N)!

Can we do better? Suggestions.

```
boolean is_prime(int N) {
  if (N == 2) return true;
  if (N % 2 == 0) return false;
  for (int i = 3; i <= Math.sqrt(N); i += 2) {
    if (N % i == 0) return false;
  }
  return true;
}
```

- Suggestion #2: Only test numbers less than or equal to square-root(x)

How much better did we do?

Let’s say we wanted to test if 1,000,000,007 was prime.

```
boolean is_prime(int N) {
  for (int i = 2; i < N; i++) {
    if (N % i == 0) return false;
  }
  return true;
}
```

Number of iterations:
1,000,000,006

```
boolean is_prime(int N) {
  for (int i = 2; i <= Math.sqrt(N);
        i++) {
    if (N % i == 0) return false;
  }
  return true;
}
```

Number of iterations:
31621

How much better did we do?

Let's say we wanted t

```
boolean is_prime(int
  for (int i = 2; i
    if (N % i == 0)
  }
  return true;
}
```

```
ime(int N) {
= 2; i <= Math.sqrt(N);
{
 == 0) return false;
;
}
```

Number of iterations:
1,000,000,006

Number of iterations:
31621

Generating primes

- Now, we're given some number *N* and we want to determine the primality of EVERY number less than or equal to N.

INPUT: N

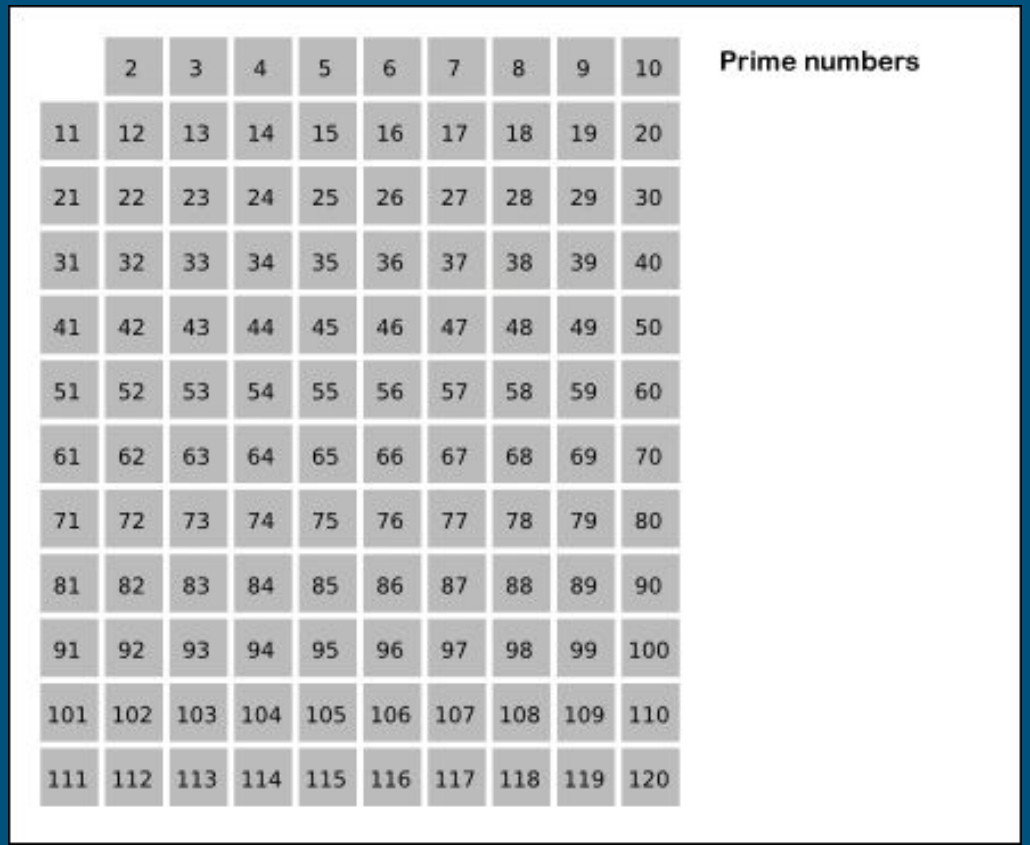
OUTPUT:

boolean[] isPrime, where isPrime[x] = true if x is prime, otherwise it's false

Easy Algorithm

```
boolean[] isPrime = new boolean[N+1];
for (int i = 2; i <= N; i++) {
  isPrime[i] = is_prime(i); // using the function we wrote
}
```

Faster Algorithm - Sieve of Eratosthenes



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Problems to Solve

http://spoj.com/problems/PRIME1/

then

http://codeforces.com/gym/100579/ (Problem A - Homework)

then

http://spoj.com/problems/ANARC09C/