CS302 – Analysis and Design of Algorithms

Algorithm Design

Content
Modulo Operation
Primes
Greatest Common Divisor
Exercises

Modulo

- The modulo operation is a mathematical operation that finds the remainder when one integer a is divided by another integer b.
  - $a \; mod \; b$
  - $a \: \% \: b$
- $a \; mod \; b$ equals the number r from the division of a by b

$$a = b \, * q + r \qquad \rightarrow \qquad r = a \, - b \times q$$

r is the remainder, q is the quotient

$$r = a \, - \, \left\lfloor \frac{a}{b} \right\rfloor * b$$

Modulo

Examples:

- $10\ mod\ 3 = 1$
  - $r = 10 - \left\lfloor \frac{10}{3} \right\rfloor * 3 = 10 - (3 * 3) = 1$
  - Because $10 = 3 * 3 + 1$
- $17 \% 5 = 2$
  - $r = 17 - \left\lfloor \frac{17}{5} \right\rfloor * 5 = 17 - 3 * 5 = 2$
  - Because $17 = 5 * 3 + 2$
- $14\ mod\ 7 = ?$

Modulo

Examples:

- $10 \; mod \; 3 = 1$
  - $r = 10 - \left\lfloor \frac{10}{3} \right\rfloor * 3 = 10 - (3 * 3) = 1$
  - Because $10 = 3 * 3 + 1$
- $17 \% 5 = 2$
  - $r = 17 - \left\lfloor \frac{17}{5} \right\rfloor * 5 = 17 - 3 * 5 = 2$
  - Because $17 = 5 * 3 + 2$
- $14 \; mod \; 7 = ?$
  - $r = 14 - \left\lfloor \frac{14}{7} \right\rfloor * 7 = 0$

Modulo

- Note that:
  - If a $mod\ b == 0$, then we say a is divisible by b, because the result is an integer.

  - If a $mod\ b \neq 0$, then we say a is NOT divisible by b, because the result includes a fractional part, e.g., 3.6.

  - The remainder, r, is always less than b. i.e., $0 \leq r < b$

Modulo

Checkpoint

I worked for 173 hours, how many days and hours did I work?

Modulo

Checkpoint

I worked for 173 hours, how many days and hours did I work?

Days = $\frac{173}{24} = 7$

Hours = $173 \;\%\; 24 = 5$

Modulo

Checkpoint

I am at work at 9’oclock and I have to work for 13 hours. At what time should I leave?

Modulo

Checkpoint

I am at work at 9 o’clock and I have to work for 13 hours. At what time should I leave?

(9 + 13) % 12 = 10 o’clock

Content
Modulo Operation
Primes
Greatest Common Divisor
Exercises

Primes

- Prime numbers are natural numbers that are divisible by only 1 and the number itself.
  - In other words, positive integers greater than 1 with exactly two factors, 1 and the number itself.
- Some of the prime numbers include 2, 3, 5, 7, 11, 13, etc.

- How to check if a number X is a prime number or not?

Trial Division

A naïve algorithm

Algorithm 1: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise

```
if X ≤ 1 then
    return FALSE;
end
if X ≤ 3 then
    return TRUE;
end
for n = 2 to X do
    if X % n == 0 then
        return FALSE;
    end
end
return TRUE;
```

Trial Division

Check 13

Algorithm 1: Check if a number is prime

```
Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise
if X ≤ 1 then
  | return FALSE;
end
if X ≤ 3 then
  | return TRUE;
end
for n = 2 to X do
  | if X % n == 0 then
  |   | return FALSE;
  | end
end
return TRUE;
```

$13 \leq 1$? No
$13 \leq 3$? No

Test[2, 3, 4, 5, 6, 7, 8,9,10,11,12]
13 % 2 == 0? No
13 % 3 == 0? No
13 % 4 == 0? No
13 % 5 == 0? No
13 % 6 == 0? No
13 % 7 == 0? No
13 % 8 == 0? No
13 % 9 == 0? No
13 % 10 == 0? No
13 % 11 == 0? No
13 % 12 == 0? No

Return True
∴ 13 is prime

Trial Division

Check 43

Algorithm 1: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise

```
if X ≤ 1 then
    return FALSE;
end
if X ≤ 3 then
    return TRUE;
end
for n = 2 to X do
    if X % n == 0 then
        return FALSE;
    end
end
return TRUE;
```

Trial Division

Check 43 Prime

Algorithm 1: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise

if $X \leq 1$ **then**
    return FALSE;
end
if $X \leq 3$ **then**
    return TRUE;
end
for $n = 2$ **to** X **do**
    if $X \ \% \ n == 0$ **then**
        return FALSE;
    end
end
return TRUE;

Trial Division

Can we have a better algorithm?

Algorithm 1: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise

```
if X ≤ 1 then
    return FALSE;
end
if X ≤ 3 then
    return TRUE;
end
for n = 2 to X do
    if X % n == 0 then
        return FALSE;
    end
end
return TRUE;
```

Trial Division

Yes. Iterate from 2 to $\sqrt{X}$. It saves time.

Algorithm 2: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise

```
if X ≤ 1 then
    return FALSE;
end
if X ≤ 3 then
    return TRUE;
end
for n = 2 to √X do
    if X % n == 0 then
        return FALSE;
    end
end
return TRUE;
```

Trial Division

Check 63

Algorithm 2: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise
if $X \leq 1$ **then**
  return FALSE;
end
if $X \leq 3$ **then**
  return TRUE;
end
for $n = 2$ **to** $\sqrt{X}$ **do**
  if $X \ \% \ n == 0$ **then**
    return FALSE;
  end
end
return TRUE;

$63 \leq 1$? No
$63 \leq 3$? No

$\sqrt{63} = 7.9$

Test[2, 3, 4,5,6,7,8]
63 % 4 == 0? No
63 % 5 == 0? No
63 % 6 == 0? No
63 % 7 == 0? YES (63 = 7*9)

Return False
∴ 63 is NOT prime

Trial Division

Check 243

Algorithm 2: Check if a number is prime

Input: Integer X
Output: Boolean TRUE if X is prime, FALSE otherwise
if $X \leq 1$ **then**
  return FALSE;
end
if $X \leq 3$ **then**
  return TRUE;
end
for $n = 2$ **to** $\sqrt{X}$ **do**
  if $X \ \% \ n == 0$ **then**
    return FALSE;
  end
end
return TRUE;

Trial Division

Check 243 Not prime: 243 = 281 * 3

Algorithm 2: Check if a number is prime

Input: Integer X

Output: Boolean TRUE if X is prime, FALSE otherwise

if $X \leq 1$ **then**
    return FALSE;
end
if $X \leq 3$ **then**
    return TRUE;
end
for $n = 2$ **to** $\sqrt{X}$ **do**
    if $X \ \% \ n == 0$ **then**
        return FALSE;
    end
end
return TRUE;

Fermat's Little Theorem

- Given an integer X, select a random number a such that $1 < a < X$ then compute:

$$a^X \; mod \; X = result$$

If $result = a$, then X MAY BE prime. Otherwise, it is not prime.

- Alternatively, compute

$$a^{X-1} \; mod \; X = result$$

If $result = 1$, the X MAY BE prime. Otherwise, it is not prime.

Which one is better? Why?

Fermat's Little Theorem

- Given an integer X, select a random number a such that $1 < a < X$ then compute:

$$a^X \; mod \; X = result$$

If $result = a$, then X MAY BE prime. Otherwise, it is not prime.

- Alternatively, compute

$$a^{X-1} \; mod \; X = result$$

If $result = 1$, the X MAY BE prime. Otherwise, it is not prime.

Which one is better? Why? – The second eqn, it saves time to compute, as it performs $X - 1$ multiplications of a instead of X multiplications.

Fermat's Little Theorem

- Check 17:
  - let's select the base $a = 2$

$$2^{17}\ mod\ 17 = 2$$

Or

$$2^{16}\ mod\ 17 = 1$$

$\therefore$ 17 is (probably) prime

- Check 33:

$$2^{33}\ mod\ 33 = 8$$

Or

$$2^{32}\ mod\ 33 = 4$$

$\therefore$ 33 is not prime

Fermat's Little Theorem

Algorithm 3: Fermat's Little Theorem Primality Test

Input: Integer X, Integer k
Output: Boolean TRUE if X is prime, FALSE otherwise
if $X \leq 1$ **then**
    return FALSE;
end
if $X \leq 3$ **then**
    return TRUE;
end
for $i = 1$ **to** k **do**
    Choose a random integer a such that $1 < a < X$;
    $result \leftarrow a^{X-1} \mod X$;
    if $result \neq 1$ **then**
        return FALSE;
    end
end
return TRUE;

Fermat’s Little Theorem

- Counterexamples:

$$2^{340}\ mod\ 341 = 1\text{, but } 341 = 11*31$$

$$5^{560}\ mod\ 561 = 1\text{, but } 561 = 3 * 11 * 17$$

Sieve of Eratosthenes

- Efficiently find all prime numbers up to a specified integer n.
- Steps:

  1. Create a list of numbers from 2 to n.

  2. Starting from the first prime (2), mark all of its multiples as non-prime.

  3. Move to the next unmarked number and repeat until all numbers are processed.

Sieve of Eratosthenes

- Find primes numbers less than 100.

Sieve of Eratosthenes

- Find primes numbers less than 100.

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	89	90	91	92
93	94	95	96	97	98	99	100	

1. Create a list of numbers from 2 to n.

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11,

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11, 17

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11, 17, 19

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11, 17, 19, 23

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11, 17, 19, 23, 29

Sieve of Eratosthenes

- Find primes numbers less than 100.

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~~

1. Create a list of numbers from 2 to n.
2. Starting 2, mark all of its multiples as non-prime.
3. Move to the next unmarked number and repeat until all numbers are processed.

Primes: 2, 3, 5, 7, 11, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 and 97.

Sieve of Eratosthenes

Checkpoint

Find primes numbers between 100 and 200.

Sieve of Eratosthenes

Checkpoint

Find primes numbers between 100 and 200.

101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199

Sieve of Eratosthenes

Pseudocode

Algorithm 5: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n

```
Create a list is_prime of size n + 1 and initialize all entries as TRUE;
is_prime[0] ← FALSE;
is_prime[1] ← FALSE;
for p = 2 to n do
    if is_prime[p] then
        for i = p × 2 to n do
            is_prime[i] ← FALSE;
        end
    end
end
return the indices i such that is_prime[i] is TRUE;
```

Sieve of Eratosthenes

Pseudocode

Algorithm 5: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n
Create a list is_prime of size $n + 1$ and initialize all entries as TRUE;
$is\_prime[0] \leftarrow FALSE$;
$is\_prime[1] \leftarrow FALSE$;
for $p = 2$ **to** n **do**
  if $is\_prime[p]$ **then**
    for $i = p \times 2$ **to** n **do**
      $is\_prime[i] \leftarrow FALSE$;
    end
  end
end
return the indices i such that $is\_prime[i]$ is TRUE;

This is incorrect! Why?

Sieve of Eratosthenes

Pseudocode

Algorithm 5: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n
Create a list is_prime of size $n+1$ and initialize all entries as TRUE;
$is\_prime[0] \leftarrow FALSE$;
$is\_prime[1] \leftarrow FALSE$;
for $p = 2$ **to** n **do**
    if $is\_prime[p]$ **then**
        for $i = p \times 2$ **to** n **do**
            $is\_prime[i] \leftarrow FALSE$;
        end
    end
end
return the indices i such that $is\_prime[i]$ is TRUE;

Step by p.
So, the loop should be:
$P * 2,$
$p * 2 + p,$
$p * 2 + 2p,$
…

Sieve of Eratosthenes

Pseudocode – Can we do better?

Algorithm 6: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n

```
Create a list is_prime of size n + 1 and initialize all entries as TRUE;
is_prime[0] ← FALSE;
is_prime[1] ← FALSE;
for p = 2 to n do
    if is_prime[p] then
        for i = p × 2 to n by p do
            is_prime[i] ← FALSE;
        end
    end
end
return the indices i such that is_prime[i] is TRUE;
```

Sieve of Eratosthenes

Pseudocode – Can we do better? Yes, instead of n, iterate until $\sqrt{n}$

Algorithm 7: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n

```
Create a list is_prime of size n + 1 and initialize all entries as TRUE;
is_prime[0] ← FALSE;
is_prime[1] ← FALSE;
for p = 2 to √n do
    if is_prime[p] then
        for i = p × 2 to n by p do
            is_prime[i] ← FALSE;
        end
    end
end
return the indices i such that is_prime[i] is TRUE;
```

Sieve of Eratosthenes

Pseudocode – Can we do better?

Algorithm 7: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n
Create a list is_prime of size $n + 1$ and initialize all entries as TRUE;
$is\_prime[0] \leftarrow FALSE$;
$is\_prime[1] \leftarrow FALSE$;
for $p = 2$ **to** $\sqrt{n}$ **do**
  if $is\_prime[p]$ **then**
    for $i = p \times 2$ **to** n **by** p **do**
      $is\_prime[i] \leftarrow FALSE$;
    end
  end
end
return the indices i such that $is\_prime[i]$ is TRUE;

Sieve of Eratosthenes

Pseudocode – Can we do better? Yes, instead of $i = p * 2$, start from $i = p^2$

Algorithm 8: Sieve of Eratosthenes

Input: Integer n (the upper limit to find all prime numbers)
Output: List of prime numbers up to n

```
Create a list is_prime of size n + 1 and initialize all entries as TRUE;
is_prime[0] ← FALSE;
is_prime[1] ← FALSE;
for p = 2 to √n do
    if is_prime[p] then
        for i = p^2 to n by p do
            is_prime[i] ← FALSE;
        end
    end
end
return the indices i such that is_prime[i] is TRUE;
```

Content
Modulo Operation
Primes
Greatest Common Divisor
Exercises

Greatest Common Divisor

- GCD is a function that takes two numbers and computes the greatest that divides both numbers.
- For example, GCD(10, 15) is 5.
  - $\frac{10}{5} = 2$
  - $\frac{15}{5} = 3$
  - 5 divides both numbers, then 5 is the greatest common divisor.

Greatest Common Divisor

- Find the GCD of 100 and 10.
- Try all the numbers starting from 2 until 10.
  - $100/2 = 50, \quad 10/2 = 5$
  - $100/3 = 33.33, \quad 10/3 = 3.33$
  - $100/4 = 25.0, \quad 10/4 = 2.5$
  - $100/5 = 20.0, \quad 10/5 = 2.0$
  - $100/6 = 16.6, \quad 10/6 = 1.6$
  - $100/7 = 14.28, \quad 10/7 = 1.428$
  - $100/8 = 12.5, \quad 10/8 = 1.25$
  - $100/9 = 11.11, \quad 10/9 = 1.11$
  - $100/10 = 10.0, \quad 10/10 = 1.0$

Greatest Common Divisor

- Find the GCD of 100 and 10.
- Try all the numbers starting from 2 until 10.
  - $100/\,2 \;=\; 50, \qquad 10/\,2 \;=\; 5$
  - $100/\,3 \;=\; 33.33, \quad 10/\,3 \;=\; 3.33$
  - $100/\,4 \;=\; 25.0, \qquad 10/\,4 \;=\; 2.5$
  - $100/\,5 \;=\; 20.0, \qquad 10/\,5 \;=\; 2.0$
  - $100/\,6 \;=\; 16.6, \qquad 10/\,6 \;=\; 1.6$
  - $100/\,7 \;=\; 14.28, \quad 10/\,7 \;=\; 1.428$
  - $100/\,8 \;=\; 12.5, \qquad 10/\,8 \;=\; 1.25$
  - $100/\,9 \;=\; 11.11, \quad 10/\,9 \;=\; 1.11$
  - $100/\,10 \;=\; 10.0, \quad 10/\,10 \;=\; 1.0$

$\therefore GCD(100, 10) = 10$

Can we do better?

Greatest Common Divisor

- Find the GCD of 100 and 10.
- Try all the numbers starting from 2 until 10.
  - $100/\ 2\ =\ 50,\quad 10/\ 2\ =\ 5$
  - $100/\ 3\ =\ 33.33,\quad 10/\ 3\ =\ 3.33$
  - $100/\ 4\ =\ 25.0,\quad 10/\ 4\ =\ 2.5$
  - $100/\ 5\ =\ 20.0,\quad 10/\ 5\ =\ 2.0$
  - $100/\ 6\ =\ 16.6,\quad 10/\ 6\ =\ 1.6$
  - $100/\ 7\ =\ 14.28,\quad 10/\ 7\ =\ 1.428$
  - $100/\ 8\ =\ 12.5,\quad 10/\ 8\ =\ 1.25$
  - $100/\ 9\ =\ 11.11,\quad 10/\ 9\ =\ 1.11$
  - $100/\ 10\ =\ 10.0,\quad 10/\ 10\ =\ 1.0$

$\therefore GCD(100, 10) = 10$

Can we do better?
Yes, instead starting from 2, start from the smallest of a and b

Greatest Common Divisor

- Checkpoint

Find the highest common factor of 78 and 20

Greatest Common Divisor

- Checkpoint

Find the highest common factor of 78 and 20

GCD(78, 20) = 2

- Maybe it would have been better to start from 2 instead of 20!
  - Let’s see a better algorithm.

Greatest Common Divisor

- Euclidean algorithm is an efficient method for computing the greatest common divisor (GCD) of two integers.
- Basic principle: The GCD of two numbers does not change if the larger number is replaced by its difference with the smaller number.

- Example:
  - GCD(252, 105) = 21
  - Replace 252 by 252 – 105 = 147
  - GCD(147, 105) = 21
  - We reduced the larger number to a smaller number → easier, faster

Greatest Common Divisor

- Math expression:

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

- Example: find GCD(178, 46)
  - $GCD(178, 46) \quad \rightarrow \quad r = 178 \,\%\, 46 = 40, \quad a = 46, b = 40$
  - $GCD(46, 40) \quad \rightarrow \quad r = 46 \,\%\, 40 = 6, \quad a = 40, b = 6$
  - $GCD(40, 6) \quad \rightarrow \quad r = 40 \,\%\, 6 = 4, \quad a = 6, b = 4$
  - $GCD(6, 4) \quad \rightarrow \quad r = 6 \,\%\, 4 = 2, \quad a = 4, b = 2$
  - $GCD(4, 2) \quad \rightarrow \quad r = 4 \,\%\, 2 = 0 \quad a = 2, b = 0$

Greatest Common Divisor

- GCD using $while$ loop

Algorithm 9: Euclidean Algorithm

Input: Two non-negative integers a and b (assume $a \geq b$)
Output: The greatest common divisor (GCD) of a and b
while $b \neq 0$ **do**
  $r \leftarrow a \mod b$;
  $a \leftarrow b$;
  $b \leftarrow r$;
end
return a;

Greatest Common Divisor

- GCD using recursion

Algorithm 10: Recursive Euclidean Algorithm

Input: Two non-negative integers a and b

Output: The greatest common divisor (GCD) of a and b

Function gcd(a, b)**:**

```
    if b = 0 then
        return a;
    end
    return gcd(b, a mod b);
```

Greatest Common Divisor

Checkpoint:

Compute the GCD(321, 122)

Algorithm 10: Recursive Euclidean Algorithm

Input: Two non-negative integers a and b
Output: The greatest common divisor (GCD) of a and b
Function `gcd`(a, b)**:**
  if $b = 0$ **then**
    return a;
  end
  return `gcd`$(b, a \mod b)$;

Algorithm 9: Euclidean Algorithm

Input: Two non-negative integers a and b (assume $a \geq b$)
Output: The greatest common divisor (GCD) of a and b
while $b \neq 0$ **do**
  $r \leftarrow a \mod b$;
  $a \leftarrow b$;
  $b \leftarrow r$;
end
return a;

Greatest Common Divisor

Checkpoint:

Compute the GCD(321, 122)

GCD(321, 122) = 1

- If two numbers a and b have their GCD =1, then we say that a and b are co-prime.
  - i.e., a is prime with respect to b, and b is prime with respect to a

Greatest Common Divisor

Checkpoint

Find the GCD(18, 30, 66)

Greatest Common Divisor

Checkpoint: Find the GCD(18, 30, 66)

$GCD(18,30)$:

1. $30 \; mod \; 18 = 12$
2. $18 \; mod \; 12 = 6$
3. $12 \; mod \; 6 = 0$

$\therefore GCD(18, 30) = 6$

$GCD(6, 66)$:

1. $66 \; mod \; 6 = 0$

$\therefore GCD(6,66) = 6$

Therefore, the GCD of 18, 30, and 66 is 6.

Content
Modulo Operation
Primes
Greatest Common Divisor
Exercises

Exercise

- Write an algorithm to check if a given number is even or odd.

Exercise

- Write an algorithm to check if a given number is even or odd.

Algorithm 11: Even or Odd Check Without Bitwise Operators

Input: An integer n
Output: A string indicating if n is "even" or "odd"
if $n \mod 2 == 0$ **then**
  return *"even"*;
end
else
  return *"odd"*;
end

Algorithm 12: Even or Odd Check Using Bitwise Operators

Input: An integer n
Output: A string indicating if n is "even" or "odd"
if $n\&1 == 0$ **then**
  return *"even"*;
end
else
  return *"odd"*;
end

Exercise

- (Maximum consecutive increasingly ordered substring) Write an algorithm that takes a string and displays the maximum consecutive increasingly ordered substring.
- For example, for a string “abcabcdgabxy”, the max consecutive increasingly ordered substring is “abcdg”
- “abcabcdgabmnsxy” → “abmnsxy”

Exercise

- Maximum consecutive increasingly ordered substring.

Algorithm 13: Find Maximum Consecutive Increasingly Ordered Substring

```
Input: A string s
Output: The longest increasing substring in s
if length(s) == 0 then
    return "";
end
max_length ← 1;
current_length ← 1;
start_index ← 0;
max_start_index ← 0;
for i ← 1 to length(s) - 1 do
    if s[i] > s[i - 1] then
        current_length ← current_length + 1;
    end
    else
        if current_length > max_length then
            max_length ← current_length;
            max_start_index ← start_index;
        end
        current_length ← 1;
        start_index ← i;
    end
end
if current_length > max_length then
    max_length ← current_length;
    max_start_index ← start_index;
end
return s[max_start_index : max_start_index + max_length];
```

Task

- Write an algorithm to calculate the dot product of two vectors.