**國立臺南大學資訊工程學系**

**資工三「演算法」課程**

**第一次作業**

**題目: Primality Testing**

|  |  |  |
| --- | --- | --- |
| **班級** | **：** | **資工三** |
| **姓名** | **：** | **曾俊杰** |
| **學號** | **：** | **S11059003** |

**老師：陳宗禧**

**中華民國 112年9月23日**

**目錄**

[(一) 簡介及問題描述 4](#_Toc147065588)

[1. 簡介 4](#_Toc147065589)

[2. 問題 4](#_Toc147065590)

[(二) 理論分析 5](#_Toc147065591)

[1. Basic Prime Testing理論 5](#_Toc147065592)

[2. Sieve of Eratosthenes理論 5](#_Toc147065593)

[3. Fermat Primality Testing理論 5](#_Toc147065594)

[4. Miller-Rabin Primality Testing理論 5](#_Toc147065595)

[(三) 演算法則 6](#_Toc147065596)

[1. 第一個演算法(Algorithm) 6](#_Toc147065597)

[**i. 演算法時間複雜度(time complexity): O(√n) 6**](#_Toc147065598)

[**ii. 演算法空間複雜度(space complexity): O(1) 6**](#_Toc147065599)

[2. 第二個演算法(Algorithm) 7](#_Toc147065600)

[**i. 演算法時間複雜度(time complexity): O(n√n) 7**](#_Toc147065601)

[**ii. 演算法空間複雜度(space complexity): O(n) 8**](#_Toc147065602)

[3. 第三個演算法(Algorithm) 8](#_Toc147065603)

[**i. 演算法時間複雜度(time complexity): O(klog2(n)) 9**](#_Toc147065604)

[**ii. 演算法空間複雜度(space complexity): O(1) 9**](#_Toc147065605)

[4. 第四個演算法(Algorithm) 10](#_Toc147065606)

[**i. 演算法時間複雜度(time complexity): O(klog2(n)) 12**](#_Toc147065607)

[**ii. 演算法空間複雜度(space complexity): O(1) 12**](#_Toc147065608)

[(四) 程式設計環境架構 13](#_Toc147065609)

[1. 程式語言 13](#_Toc147065610)

[2. 程式開發工具 13](#_Toc147065611)

[3. 電腦硬體 13](#_Toc147065612)

[4. 作業系統 13](#_Toc147065613)

[(五) 程式 (含source code, input code, and output code) 14](#_Toc147065614)

[1. 主程式 14](#_Toc147065615)

[**(i) Basic Prime Testing 14**](#_Toc147065616)

[**(ii) Sieve of Eratosthenes 17**](#_Toc147065617)

[**(iii) Fermat Primality Testing 20**](#_Toc147065618)

[**(iv) Miller-Rabin Primality Testing 23**](#_Toc147065619)

[2. Input Code Format 28](#_Toc147065620)

[3. Output Code Format 28](#_Toc147065621)

[(五) 執行結果、討論與心得 30](#_Toc147065622)

[1. 執行結果 30](#_Toc147065623)

[**(i) Basic Prime Testing 30**](#_Toc147065624)

[**(ii) Sieve of Eratosthenes 30**](#_Toc147065625)

[**(iii) Fermat Primality Testing 31**](#_Toc147065626)

[**(iv) Miller-Rabin Primality Testing 31**](#_Toc147065627)

[2. 討論 32](#_Toc147065628)

[**(i) Running time & Problem size *n* 33**](#_Toc147065629)

[**(ii) Correct rate & Problem size *n* 40**](#_Toc147065630)

[3. 心得 41](#_Toc147065631)

[參考文獻 42](#_Toc147065632)

**(一) 簡介及問題描述**

1. 簡介

以不同理論函數，求Prime(*n*)。首先定義一正整數n，其中n大於1，由合理條件判斷n是質數或合成數，得出實驗數據，討論分析並且比對觀察之。

2. 問題

Prime(*n*)問題定義：

求Prime(*n*)，實作與分析

* + 1. Basic Prime Testing
    2. Sieve of Eratosthenes
    3. Fermat Primality Testing
    4. Miller-Rabin Primality Testing

計算(i)、(ii)、(iii)、(iv)的時間複雜度和空間複雜度

觀察(i)、(ii)、(iii) 在不同次數測試下與測試時間的關係

統計(iii)、(iv)在不同次數測試下與測試結果正確率的關係

**(二) 理論分析**

以一個大於1的正整數n為欲求的數為前提，進行下列四種理論分析。

1. Basic Prime Testing理論

首先第一個Basic Prime Test，令i為逐漸遞增的變數，從最小的質數2開始判斷，在n能拆成i\*i的前提下(因為i必大於1，所以迭代到i時，若i大於√n，則i的乘數須小於√n，但因i之前的數都沒有通過判斷，因此該乘數不存在)，測試n是否能整除i，如果可以，則代表n是i的倍數，是為一合數，如果能通過測試則是一質數。

2. Sieve of Eratosthenes理論

其次第二個Sieve of Eratosthens，建立一個具有n個元素的陣列，從值域從1開始到n，將所有的元素標記為0(表示是質數)，該陣列index+1為實際數值，將index0設為1(因為1不屬於質數或合數)，從index1開始演算法，如果該index的值為0，則使用內迴圈做倍數為合數的值的標記，在不大於n的前提下，對所有該值1倍以上的數標記為1(表示是合數)，直到完成n開根號取整數的值時(因為做到此時，若該數大於√n，則它的乘數需小於√n，但該小於√n的乘數若是存在，則已經檢查過了)，即完成實際值質數為0，其餘(不包含1)為1(合數)的已標記陣列。

3. Fermat Primality Testing理論

再者第三個Fermat Primality Test，隨機選定一個正整數k，作為測試次數，藉由費馬小定理a^(n-1) = 1對n取餘數(%n)，其中a為隨機整數，在範圍1到n-1之間，每次取一個a做測試，如果有任一組不符合條件式，則n為合數，然而，即使是通過k次測試後，也不一定保證n為質數，仍有可能為合數。

4. Miller-Rabin Primality Testing理論

最後第四個Miller-Rabin Primality Test，此時n須為大於2的正整數，隨機選定一個正整數k，作為測試次數，由n-1 = d \* 2^r紀錄正整數d、r，d須為奇數，在範圍2到n-1之間，隨機選定一個值a，做模除x0 = (a^d) %n，如果x0為+1，則n為合數；如果為-1，則n可能為質數；若兩者皆非，則需以bi = ((b(i-1))^2) %n進行下一步判斷，首先選定r作為測試次數，然後得出x1 = ((x0)^2) % n，再次判斷x1是否為+1或-1，若還是無法判斷，則迭代進行運算，直到判斷成功為止。但是有可能直到r次測試後，仍然無法判斷，此時就只能判定n為合數，甚至，即使n通過k次一系列的測試後，也只能被判定為「可能是質數」而並非絕對。

**(三) 演算法則**

1. 第一個演算法(Algorithm)

**一張含有 圖表, 寫生, 行, 文字 的圖片

自動產生的描述**

1 func Basic\_prime\_testing**(**n**):** result**,** result is bool

2 i **<-** 0

3 **for** i **(**2 to sqrt**(**n**))** **do**

4 **if** n **%** i **=** 0 then

5 **return** **false**

6 end **if**

7 end **for**

8 **return** **true**

1. 演算法時間複雜度(time complexity): O(√n)

**Line 2: 1**

**Line 3: (√n – 2 + 1) + 1**

**Line 4: (√n – 2 + 1)**

**Total: 2√n -> √n**

1. 演算法空間複雜度(space complexity): O(1)

Because no matter how large n is, it does not cost extra memory in any case. I would

answer the space complexity is O(1).

2. 第二個演算法(Algorithm)

**一張含有 圖表, 寫生, 行, 方案 的圖片

自動產生的描述**

1 func fill\_all\_multiple\_to\_1**(**data**[**n**],** start**,** n**)**

2 i **<-** 2

3 **for** i **(**2 to n**/**start**)** **do**

4 result**[**start **\*** i **-** 1**]** **<-** 1

5 end **for**

6

7 func Sieve\_of\_eratosthenes**(**data**[**n**],** n**):** result**,** result is array

8 i **<-** 0

9 result**[**0**]** **<-** 1

10 **for** i **(**1 to sqrt**(**n**))** **do**

11 **if** result**[**i**]** **=** 0 then

12 fill\_all\_multiple\_to\_1**(**data**[**n**],** i**+**1**,** n**)**

13 end **if**

14 end **for**

15 **return** data**[**n**]**

1. 演算法時間複雜度(time complexity): O(n√n)

Line 2: 1

Line 3: (n/start – 2 + 1) + 1

Line 4: n/start – 1

Total of fill\_all\_multiple\_to\_1: 2n/start -> maximum = 50 times(%2)

Line 9: √n + 1

Line 10: √n

Line 11: √n \* (2n/start)

Total: + 2√n+ 3 -> n√n

1. 演算法空間複雜度(space complexity): O(n)

Because I use an array to store n elements, so I would answer the space

complexity is O(n).

3. 第三個演算法(Algorithm)

**一張含有 文字, 圖表, 行, 字型 的圖片

自動產生的描述**

1 func modulo**(**base**,** exponential**,** n**):** mod**,** mod is int

2 a **<-** 1

3 b **<-** base

4 **while** exponential **>** 0 **do**

5 **if** exponential **%** 2 **=** 1 then

6 a **<-** **(**a **\*** b**)** **%** n

7 end **if**

8 b **<-** **(**b **\*** b**)** **%** n

9 exponential **<-**exponential **/** 2

10 end **while**

11 **return** a **%** n

12

13 func Fermat\_primality\_testing**(**n**):** result**,** result is bool

14 pick a random integer k

15 i **<-** 1

16 **for** i **(**1 to k**)** **do**

17 pick a random integer tester in **(**1**,** n**-**1**)**

18 **if** modulo**(**tester**,** n**-**1**,** n**)** **!=** 1 then

19 **return** **false**

20 end **if**

21 end **for**

22 **return** **true**

1. 演算法時間複雜度(time complexity): O(klog2(n))

Line 2: 1

Line 3: 1

Line 4: log2(n)

Line 5: log2(n)

Line 6: log2(n) / 2

Line 7: log2(n)

Line 8: log2(n)

Total of modulo: (9/2)log2(n) + 1

Line 13: 1

Line 14: 1

Line 15: k + 1

Line 16: 1

Link 17: k \* ((9/2)log2(n) + 1) = (9k/2)log2(n) + k

Total: (9k/2)log2(n) + 2k + 4 -> klog2(n)

1. 演算法空間複雜度(space complexity): O(1)

Because no matter how large n is, it does not cost extra memory in any

case. I would answer the space complexity is O(1).

4. 第四個演算法(Algorithm)

**一張含有 圖表, 方案, 圖畫, 寫生 的圖片

自動產生的描述**

1 func modulo**(**base**,** exponential**,** n**):** mod**,** mod is int

2 a **<-** 1

3 b **<-** base

4 **while** exponential **>** 0 **do**

5 **if** exponential **%** 2 **=** 1 then

6 a **<-** **(**a **\*** b**)** **%** n

7 end **if**

8 b **<-** **(**b **\*** b**)** **%** n

9 exponential **<-** exponential **/** 2

10 end **while**

11 **return** a **%** n

12 func check\_negative\_or\_positive\_1**(**n**,** times**,** input**):** flag**,** flag is bool

13 **for** i **(**1 to times**)** **do**

14 input **<-** **(**input **\*** input**)** **%** n

15 **if** input **=** 1

16 **return** **false**

17 elsif input **=** n **-** 1

18 **return** **true**

19 end **if**

20 end **for**

21 **return** **false**

23 func MillerRabin\_primality\_testing**(**n**):** result**,** result is bool

24 **if** n **%** 2 **=** 0

25 **return** **false**

26 end **if**

27

28 write n **-** 1 as d **\*** 2**^**r

29 pick a random integer k

30

31 **for** i **(**1 to k**)** **do**

32 pick a random integer tester in **(**2**,** n**-**2**)**

33 x **<-** modulo**(**tester**,** d**,** n**)**

34 **if** x **=** 1 **or** x **=** n**-**1 then

35 **continue** loop

36 end **if**

37 flag **=** check\_negative\_or\_positive\_1\_in\_r**-**1\_times**(**n**,** r**-**1**,** x**)**

38 **if** **not(**flag**)** then

39 **return** **false**

40 end **if**

41 end **for**

42 **return** **true**

1. 演算法時間複雜度(time complexity): O(klog2(n))

Total of modulo: (9/2)log2(n) + 1

Line 13: (r – 1) + 1

Line 14: r - 1

Line 15: r – 1

Total of check\_negative\_or\_positive\_1: 3r - 2

Line 24: 1

Line 28: log2(n) + 1

Line 29: 1

Line 31: k + 1

Line 32: k

Line 33: k \* ((9/2)log2(n) + 1) = (9k/2)log2(n) + k

Line 34: k

Line 37: k \* (3r – 2) = 3kr – 2k

Line 38: k

Total: 4 + (9k/2 + 1)log2(n) + 3k + 3kr -> klog2(n)

1. 演算法空間複雜度(space complexity): O(1)

Because no matter how large n is, it does not cost extra memory in any case. I

would answer the space complexity is O(1).

**(四) 程式設計環境架構**

1. 程式語言

C++ in Microsoft Windows10 64-bits operating system

2. 程式開發工具

Microsoft Visual Studio 2019

3. 電腦硬體

CPU: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

Main Memory: 16.0 GB

顯示卡1: Intel® UHD Graphics

顯示卡2: NVIDIA GeForce RTX 3050 Ti Laptop GPU

4. 作業系統

Windows 10 家用版

版本: 22H2

**(五) 程式 (含source code, input code, and output code)**

1. 主程式

**(i) Basic Prime Testing**

#include <iostream>

#include <random>

#include <ctime>

**using** **namespace** std**;**

#define test\_size 100

bool Basic\_prime\_testing**(**int**);**

void Is\_valid\_input**(**int**\*);**

void print\_result**(**int**);**

int main**(**void**)** **{**

int n**,** i**;**

double start**,** end**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist**(**2**,** INT\_MAX**);**

start **=** clock**();**

**do** **{**

cout **<<** "Enter a integer n which larger than 2 for basic prime testing(enter -1 to quit, 0 to create 100 random n): "**;**

cin **>>** n**;**

Is\_valid\_input**(&**n**);**

**if** **(**n **==** **-**1**)** **{**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms"**;**

**return** 0**;**

**}**

**else** **if** **(**n **==** 0**)** **{**

**for** **(**i **=** 0**;** i **<** test\_size**;** i**++)** **{**

n **=** dist**(**rd**);**

print\_result**(**n**);**

cout **<<** "\n"**;**

**}**

**}**

print\_result**(**n**);**

cout **<<** "\n\n"**;**

**}** **while** **(true);**

**}**

bool Basic\_prime\_testing**(**int n**)** **{**

int i**;**

**for** **(**i **=** 2**;** i **<=** sqrt**(**n**);** i**++)** **{**

**if** **(**n **%** i **==** 0**)** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

void Is\_valid\_input**(**int**\*** n**)** **{**

**if** **(\***n **==** **-**1 **||** **\***n **==** 0**)** **{**

**return;**

**}**

**else** **if** **(\***n **<=** 1**)** **{**

cout **<<** "Invalid input, please enter n which is larger than 1: "**;**

cin **>>** **\***n**;**

Is\_valid\_input**(**n**);**

**}**

**}**

void print\_result**(**int n**)** **{**

**if** **(**Basic\_prime\_testing**(**n**))** **{**

cout **<<** n **<<** " is a prime number."**;**

**}**

**else** **{**

cout **<<** n **<<** " is not a prime number."**;**

**}**

**}**

### (ii) Sieve of Eratosthenes

#include <iostream>

#include <vector>

#include <random>

**using** **namespace** std**;**

void fill\_all\_multiple\_to\_1**(**vector**<**int**>&,** int**,** int**);**

vector**<**int**>** Sieve\_of\_eratosthenes**(**int**);**

void Is\_valid\_input**(**int**\*);**

void print\_result**(**vector**<**int**>,** int**);**

int main**(**void**)** **{**

int n**;**

vector**<**int**>** result**;**

double start**,** end**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist**(**2**,** sqrt**(**INT\_MAX**));**

start **=** clock**();**

**do** **{**

cout **<<** "Enter a integer n which larger than 1 for Sieve of Eratosthenes(enter -1 to quit, 0 to create a random n): "**;**

cin **>>** n**;**

Is\_valid\_input**(&**n**);**

**if** **(**n **==** **-**1**)** **{**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms"**;**

**return** 0**;**

**}**

**else** **if** **(**n **==** 0**)** **{**

n **=** dist**(**rd**);**

result **=** Sieve\_of\_eratosthenes**(**n**);**

print\_result**(**result**,** n**);**

**}**

result **=** Sieve\_of\_eratosthenes**(**n**);**

print\_result**(**result**,** n**);**

**}** **while** **(true);**

**}**

void fill\_all\_multiple\_to\_1**(**vector**<**int**>&** result**,** int start**,** int n**)** **{**

int i **=** 2**;**

**for** **(**i **=** 2**;** start **\*** i **<=** n**;** i**++)** **{**

result**[**start **\*** i **-** 1**]** **=** 1**;**

**}**

**}**

vector**<**int**>** Sieve\_of\_eratosthenes**(**int n**)** **{**

vector**<**int**>** result**(**n**,** 0**);**

int i**;**

result**[**0**]** **=** 1**;**

**for** **(**i **=** 1**;** i **<=** sqrt**(**n**);** i**++)** **{**

**if** **(**result**[**i**]** **==** 0**)** **{**

fill\_all\_multiple\_to\_1**(**result**,** i **+** 1**,** n**);**

**}**

**}**

**return** result**;**

**}**

void Is\_valid\_input**(**int**\*** n**)** **{**

**if** **(\***n **==** **-**1 **||** **\***n **==** 0**)** **{**

**return;**

**}**

**else** **if** **(\***n **<=** 1**)** **{**

cout **<<** "Invalid input, please enter n which is larger than 1: "**;**

cin **>>** **\***n**;**

Is\_valid\_input**(**n**);**

**}**

**}**

void print\_result**(**vector**<**int**>** result**,** int n**)** **{**

cout **<<** "These are prime numbers that small than " **<<** n **<<** ": "**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

**if** **(**result**[**i**]** **==** 0**)** **{**

cout **<<** i **+** 1 **<<** ", "**;**

**}**

**}**

cout **<<** "end.\n\n"**;**

**}**

### (iii) Fermat Primality Testing

#include <iostream>

#include <random>

#include <ctime>

**using** **namespace** std**;**

#define LL long long

#define test\_size 100

LL modulo**(**LL**,** LL**,** LL**);**

bool Fermat\_primality\_testing**(**int**);**

void Is\_valid\_input**(**int**\*);**

void print\_result**(**int**);**

int main**(**void**)** **{**

int n**,** i**;**

double start**,** end**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist**(**2**,** INT\_MAX**);**

start **=** clock**();**

**do** **{**

cout **<<** "Enter a integer n which larger than 1 for Fermat primality testing(enter -1 to quit, 0 to create 100 random n): "**;**

cin **>>** n**;**

Is\_valid\_input**(&**n**);**

**if** **(**n **==** **-**1**)** **{**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms"**;**

**return** 0**;**

**}**

**else** **if** **(**n **==** 0**)** **{**

**for** **(**i **=** 0**;** i **<** test\_size**;** i**++)** **{**

n **=** dist**(**rd**);**

print\_result**(**n**);**

cout **<<** "\n"**;**

**}**

**}**

print\_result**(**n**);**

cout **<<** "\n\n"**;**

**}** **while** **(true);**

**}**

LL modulo**(**LL base**,** LL exponential**,** LL n**)** **{**

LL a **=** 1**;**

LL b **=** base**;**

**while** **(**exponential **>** 0**)** **{**

**if** **(**exponential **%** 2 **==** 1**)**

a **=** **(**a **\*** b**)** **%** n**;**

b **=** **(**b **\*** b**)** **%** n**;**

exponential **=** exponential **/** 2**;**

**}**

**return** a **%** n**;**

**}**

bool Fermat\_primality\_testing**(**int n**)** **{**

int i**,** k**,** tester**;**

k **=** rand**()** **%** 1000 **+** 1**;**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** rand**()** **%** **(**n **-** 1**)** **+** 1**;**

**if** **(**modulo**(**tester**,** n **-** 1**,** n**)** **!=** 1**)** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

void Is\_valid\_input**(**int**\*** n**)** **{**

**if** **(\***n **==** **-**1 **||** **\***n **==** 0**)** **{**

**return;**

**}**

**else** **if** **(\***n **<=** 1**)** **{**

cout **<<** "Invalid input, please enter n which is larger than 1: "**;**

cin **>>** **\***n**;**

Is\_valid\_input**(**n**);**

**}**

**}**

void print\_result**(**int n**)** **{**

**if** **(**Fermat\_primality\_testing**(**n**))** **{**

cout **<<** n **<<** " is a prime number."**;**

**}**

**else** **{**

cout **<<** n **<<** " is not a prime number."**;**

**}**

**}**

### (iv) Miller-Rabin Primality Testing

#include <iostream>

#include <random>

#include <ctime>

**using** **namespace** std**;**

#define LL long long

#define test\_size 100

LL modulo**(**LL**,** LL**,** LL**);**

bool check\_negative\_or\_positive\_1**(**int**,** int**,** LL**);**

bool MillerRabin\_primality\_testing**(**int**);**

void Is\_valid\_input**(**int**\*);**

void print\_result**(**int**);**

int main**(**void**)** **{**

int n**,** i**;**

double start**,** end**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist**(**2**,** INT\_MAX**);**

start **=** clock**();**

**do** **{**

cout **<<** "Enter a integer n which larger than 1 for Miller-Rabin primality testing(enter -1 to quit): "**;**

cin **>>** n**;**

Is\_valid\_input**(&**n**);**

**if** **(**n **==** **-**1**)** **{**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms"**;**

**return** 0**;**

**}**

**else** **if** **(**n **==** 0**)** **{**

**for** **(**i **=** 0**;** i **<** test\_size**;** i**++)** **{**

n **=** dist**(**rd**);**

print\_result**(**n**);**

cout **<<** "\n"**;**

**}**

**}**

print\_result**(**n**);**

cout **<<** "\n\n"**;**

**}** **while** **(true);**

**}**

LL modulo**(**LL base**,** LL exponential**,** LL n**)** **{**

LL a **=** 1**;**

LL b **=** base**;**

**while** **(**exponential **>** 0**)** **{**

**if** **(**exponential **%** 2 **==** 1**)**

a **=** **(**a **\*** b**)** **%** n**;**

b **=** **(**b **\*** b**)** **%** n**;**

exponential **=** exponential **/** 2**;**

**}**

**return** a **%** n**;**

**}**

bool check\_negative\_or\_positive\_1**(**int n**,** int times**,** LL input**)** **{**

int i**;**

**for** **(**i **=** 1**;** i **<** times**;** i**++)** **{**

input **=** **(**input **\*** input**)** **%** n**;**

**if** **(**input **==** 1**)** **{**

**return** **false;**

**}**

**else** **if** **(**input **+** 1 **==** n**)** **{**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

bool MillerRabin\_primality\_testing**(**int n**)** **{**

**if** **(**n **%** 2 **==** 0**)** **{**

**return** **false;**

**}**

int r**,** d**,** k**,** i**,** tester**,** x**;**

int count\_2 **=** n **-** 1**;**

bool flag**;**

k **=** rand**()** **%** 100 **+** 1**;**

**for** **(**r **=** 0**;** count\_2 **%** 2 **!=** 1**;** r**++)** **{**

count\_2 **/=** 2**;**

**}**

d **=** **(**n **-** 1**)** **/** pow**(**2**,** r**);**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** rand**()** **%** **(**n **-** 4**)** **+** 2**;**

x **=** modulo**(**tester**,** d**,** n**);**

**if** **(**x **==** 1 **||** x **==** n **-** 1**)** **{**

**continue;**

**}**

flag **=** check\_negative\_or\_positive\_1**(**n**,** r**,** x**);**

**if** **(!**flag**)** **{**

**return** flag**;**

**}**

**}**

**return** **true;**

**}**

void Is\_valid\_input**(**int**\*** n**)** **{**

**if** **(\***n **==** **-**1 **||** **\***n **==** 0**)** **{**

**return;**

**}**

**else** **if** **(\***n **<=** 2**)** **{**

cout **<<** "Invalid input, please enter n which is larger than 2: "**;**

cin **>>** **\***n**;**

Is\_valid\_input**(**n**);**

**}**

**}**

void print\_result**(**int n**)** **{**

**if** **(**MillerRabin\_primality\_testing**(**n**))** **{**

cout **<<** n **<<** " is a prime number."**;**

**}**

**else** **{**

cout **<<** n **<<** " is not a prime number."**;**

**}**

**}**

2. Input Code Format

(1) 53

(2) 193

(3) 221

3. Output Code Format

**(1)**

To (i)、(iii) and (iv): 53 is a prime number.

To (ii): These are prime numbers that small than 53: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,

31, 37, 41, 43, 47, 53, end.

**(2)**

To (i)、(iii) and (iv): 193 is a prime number.

To (ii): These are prime numbers that small than 193: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,

31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,

127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, end.

**(3)**

To (i)、(iii) and (iv): 221 is not a prime number.

To (ii): These are prime numbers that small than 221: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,

31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,

127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211,

end.

**(五) 執行結果、討論與心得**

1. 執行結果

(i) Basic Prime Testing

一張含有 文字, 螢幕擷取畫面, 陳列, 電腦 的圖片

自動產生的描述

(ii) Sieve of Eratosthenes

一張含有 文字, 電子產品, 螢幕擷取畫面, 陳列 的圖片

自動產生的描述

(iii) Fermat Primality Testing

一張含有 文字, 螢幕擷取畫面, 電腦, 軟體 的圖片

自動產生的描述

(iv) Miller-Rabin Primality Testing

一張含有 文字, 螢幕擷取畫面, 電腦, 陳列 的圖片

自動產生的描述

1. 討論

The following discussing will follow the format: sample code I used and the purpose of discussing. To start the testing, just replace the same-named functions and check for all comments. When checking my github website, please read the “Readme.txt” in HW1\_Primality for understanding anything that you don’t know about. Here is the URL for the file:

<https://github.com/Star369666/NUTN_Algoritm/blob/main/HW1_Primality/Readme.txt>

Testing main function for (i) and (iii) for the clean execution time without input time:

int main**(**void**)** **{**

int n **=** 2**;**

double start**,** end**;**

start **=** clock**();**

print\_result**(**n**);**

cout **<<** "\n"**;**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms\n"**;**

**return** 0**;**

**}**

Testing main function for (ii) for the clean execution time without input time:

int main**(**void**)** **{**

int n **=** 2**;**

vector**<**int**>** result**;**

double start**,** end**;**

start **=** clock**();**

result **=** Sieve\_of\_eratosthenes**(**n**);**

print\_result**(**result**,** n**);**

end **=** clock**();**

cout **<<** "The program run " **<<** **(**end **-** start**)** **<<** "ms\n"**;**

**return** 0**;**

**}**

1. **Running time & Problem size *n***

All results for (i), (ii) and (iii) pictures is in the following URL:

i: <https://github.com/Star369666/NUTN_Algoritm/tree/main/HW1_Primality/HW1_(i)_and_(iii)/(i)>

ii:

<https://github.com/Star369666/NUTN_Algoritm/tree/main/HW1_Primality/HW1_(ii)>

iii:

<https://github.com/Star369666/NUTN_Algoritm/tree/main/HW1_Primality/HW1_(i)_and_(iii)/(iii)>

**一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述**

Testing main function and Fermat function for (iii) for the correct rate result:

int main**(**void**)** **{**

int n **=** 2**;**

Fermat\_primality\_testing**(**n**);**

**return** 0**;**

**}**

void Fermat\_primality\_testing**(**int n**)** **{**

int i**,** tester**,** j**;**

int k **=** 5**;**

bool flag**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist2**(**1**,** n **-** 1**);**

**for** **(**j **=** 0**;** j **<** 25**;** j**++)** **{**

flag **=** **true;**

**if** **(**j **%** 2 **==** 0**)** **{**

k **\*=** 2**;**

**}**

**else** **{**

k **\*=** 5**;**

**}**

cout **<<** "With k = " **<<** k **<<** ": "**;**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** dist2**(**rd**);**

**if** **(**modulo**(**tester**,** n **-** 1**,** n**)** **!=** 1**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

flag **=** **false;**

**break;**

**}**

**}**

**if** **(**flag**)** **{**

cout **<<** n **<<** " is a prime number.\n"**;**

**}**

**}**

**}** //Having to comment the print\_result function and change the return of Fermat function to void, or it would happen error

When I found the correct rate of Fermat Primality Testing is 100%, I tried to find the mistake case intentionally:

void Fermat\_primality\_testing**(**int n**)** **{**

int i**,** tester**,** j**;**

int k**;**

bool flag**;**

random\_device rd**;**

uniform\_int\_distribution**<**int**>** dist2**(**1**,** n **-** 1**);**

**for** **(**k **=** 1**;** j **<=** 25**;** k**++)** **{**

flag **=** **true;**

cout **<<** "With k = " **<<** k **<<** ": "**;**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** dist2**(**rd**);**

**if** **(**modulo**(**tester**,** n **-** 1**,** n**)** **!=** 1**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

flag **=** **false;**

**break;**

**}**

**}**

**if** **(**flag**)** **{**

cout **<<** n **<<** " is a prime number.\n"**;**

**}**

**}**

**}**

Testing main function and Miller-Rabin function for (iv) for the correct rate result:

int main**(**void**)** **{**

int n**;**

MillerRabin\_primality\_testing**(**n**);**

**return** 0**;**

**}**

void MillerRabin\_primality\_testing**(**int n**)** **{**

**if** **(**n **%** 2 **==** 0**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

**}**

int r**,** d**,** k**,** i**,** tester**,** x**;**

int count\_2 **=** n **-** 1**;**

int j **=** 0**;**

bool flag**,** flag2**;**

k **=** 5**;**

**for** **(**j **=** 0**;** j **<** 14**;** j**++)** **{**

flag2 **=** **true;**

**if** **(**j **%** 2 **==** 0**)** **{**

k **\*=** 2**;**

**}**

**else** **{**

k **\*=** 5**;**

**}**

cout **<<** "With k = " **<<** k **<<** ": "**;**

**for** **(**r **=** 0**;** count\_2 **%** 2 **!=** 1**;** r**++)** **{**

count\_2 **/=** 2**;**

**}**

d **=** **(**n **-** 1**)** **/** pow**(**2**,** r**);**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** rand**()** **%** **(**n **-** 4**)** **+** 2**;**

x **=** modulo**(**tester**,** d**,** n**);**

**if** **(**x **==** 1 **||** x **==** n **-** 1**)** **{**

**continue;**

**}**

flag **=** check\_negative\_or\_positive\_1**(**n**,** r**,** x**);**

**if** **(!**flag**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

flag2 **=** **false;**

**break;**

**}**

**}**

**if** **(**flag2**)** **{**

cout **<<** n **<<** " is a prime number.\n"**;**

**}**

**}**

**}** //Having to comment the print\_result function and change the return of Miller-Rabin function to void, or it would happen error

When I found the correct rate of Miller-Rabin Primality Testing is 100%, I tried to find the mistake case intentionally:

void MillerRabin\_primality\_testing**(**int n**)** **{**

**if** **(**n **%** 2 **==** 0**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

**}**

int r**,** d**,** k**,** i**,** tester**,** x**;**

int count\_2 **=** n **-** 1**;**

int j **=** 0**;**

bool flag**,** flag2**;**

k **=** 5**;**

**for** **(**k **=** 1**;** k **<=** 25**;** k**++)** **{**

flag2 **=** **true;**

cout **<<** "With k = " **<<** k **<<** ": "**;**

**for** **(**r **=** 0**;** count\_2 **%** 2 **!=** 1**;** r**++)** **{**

count\_2 **/=** 2**;**

**}**

d **=** **(**n **-** 1**)** **/** pow**(**2**,** r**);**

**for** **(**i **=** 0**;** i **<** k**;** i**++)** **{**

tester **=** rand**()** **%** **(**n **-** 4**)** **+** 2**;**

x **=** modulo**(**tester**,** d**,** n**);**

**if** **(**x **==** 1 **||** x **==** n **-** 1**)** **{**

**continue;**

**}**

flag **=** check\_negative\_or\_positive\_1**(**n**,** r**,** x**);**

**if** **(!**flag**)** **{**

cout **<<** n **<<** " is not a prime number.\n"**;**

flag2 **=** **false;**

**break;**

**}**

**}**

**if** **(**flag2**)** **{**

cout **<<** n **<<** " is a prime number.\n"**;**

**}**

**}**

**}**

1. **Correct rate & Problem size *n***

For both Fermat Primality Testing and Miller-Rabin Primality Testing, because I can’t

find any case make them mistake. The correct rate are both 100%.

All of results for each is pushed on the following URL. I tried from 3 digits to 9 digits, and finally test with randomly digits and selection between odd or even numbers. Ps. I use the same samples for both Fermat Primality Testing and Miller-Rabin Primality Testing.

<https://github.com/Star369666/NUTN_Algoritm/tree/main/HW1_Primality/HW1_(iii)_and_(iv)>

## **3. 心得**

本實作讓我了解四種不同的測試質數演算法，複習計算big O空間複雜度與時間複雜度，以及學習大數模除、word編排、撰寫虛擬碼、有系統性地寫出一份類論文的報告、測試各種參數與結果的深層關係、繪製程式流程圖等……受益匪淺。一開始我並不能完全看懂虛擬碼在寫什麼，即使有完整的描述也一樣，我只能先去看影片解說，搭配圖形才能補全我對做法的認知，我覺得該篇報告的格式就很有邏輯性，先是就自己的想法去描述該演算法的邏輯，接著將演算法以實際步驟一一剖析，其次才是付諸實行編寫程式，最後再比對參數對結果的影響，其中，花費我最長時間的是編寫程式，並不是因為我不會寫出同樣邏輯的程式，而是我得去思考該怎麼寫出「我的程式」而不是完全照著他的邏輯編寫，在虛擬碼中一行帶過的描述要實現出來，可能就得使用好幾個函數支持，還得去考慮程式的順序，I/O的形式，不能使用超過兩層巢狀，迴圈優化等等，雖然堅持這些原則可能導致我花費的時間倍增，但是寫到這邊之後，跨越難題的成就感，仍然是激勵人心的。該報告最終的討論，我感到有些許遺憾，我明白該演算法(iii)、(iv)具備的缺失，因此k越小應該會存在誤判，然而，我無法測出，這或許跟我寫得程式有關係，在(iv)中，n進入r-1次for迴圈時的x = n-1的判斷，我將其改為return true，因為如果是continue的話，n最終的結局只有被判定為合數，即使出了迴圈也一樣，這在我用53去逐行運行時發現。另一個有趣的學習是大數模除，如果只是依靠pow()，甚至是long long是沒辦法處理龐大base的好幾次方，模除便在此發揮作用。縱使這次有不足之處，但珍貴的經驗會不斷增進我的能力，周而復始。

**參考文獻**

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms," Third Edition, The MIT Press, 2009.
2. R.C.T. Lee, S.S. Tseng, R.C. Chang, and Y.T.Tsai, "Introduction to the Design and Analysis of Algorithms," McGraw-Hill, 2005.
3. Anany V. Levitin, "Introduction to the Design and Analysis of Algorithms," 3rd Edition, Addison Wesley, 2012.
4. Richard Neapolitan and Kumarss Naimipour, "Foundations of Algorithms," Fourth Edition, Jones and Bartlett Publishers, 2010.
5. <https://www.hackerearth.com/practice/math/number-theory/primality-tests/tutorial/>
6. <https://reurl.cc/K3bVNy>
7. <https://reurl.cc/OjAxdv>
8. [Sieve Of Eratosthenes (visualized) - YouTube](https://www.youtube.com/watch?v=dhfhu9Q5g8U&ab_channel=MathematicalVisualProofs)
9. [初學者學演算法｜從時間複雜度認識常見演算法. 程式麻瓜的程式知識課（四） | by Cheng-Wei Hu | 胡程維 | AppWorks School | Medium](https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E5%BE%9E%E6%99%82%E9%96%93%E8%A4%87%E9%9B%9C%E5%BA%A6%E8%AA%8D%E8%AD%98%E5%B8%B8%E8%A6%8B%E6%BC%94%E7%AE%97%E6%B3%95-%E4%B8%80-b46fece65ba5)
10. [費馬小定理 - 維基百科，自由的百科全書 (wikipedia.org)](https://zh.wikipedia.org/zh-tw/%E8%B4%B9%E9%A9%AC%E5%B0%8F%E5%AE%9A%E7%90%86)
11. [Testing for Primality (Miller-Rabin Test) - YouTube](https://www.youtube.com/watch?v=8i0UnX7Snkc&ab_channel=NesoAcademy)
12. [(5) Add Code Snippets in Word | Microsoft Word Code Converter & Syntax Highlighter - YouTube](https://www.youtube.com/watch?v=oaaPUqmTP00&ab_channel=ChesterTugwell)
13. [How do you display code snippets in MS Word preserving format and syntax highlighting? - Stack Overflow](https://stackoverflow.com/questions/387453/how-do-you-display-code-snippets-in-ms-word-preserving-format-and-syntax-highlig)
14. [[C++ ] 程式該如何輸出成exe?? - 看板 NTUBIME102HW - 批踢踢實業坊 (ptt.cc)](https://www.ptt.cc/bbs/NTUBIME102HW/M.1254913652.A.2B5.html)
15. [Word 教學課程--製作目錄 (dyu.edu.tw)](https://aries.dyu.edu.tw/~lhuang/class/office/word-directory.htm)
16. [C Program to Perform Fermat Primality Test - Sanfoundry](https://www.sanfoundry.com/c-program-perform-fermat-primality-test/)
17. [modular arithmetic - Modulus of large(ish) numbers - Mathematics Stack Exchange](https://math.stackexchange.com/questions/213342/modulus-of-largeish-numbers?rq=1)
18. [C/C++ 使用 rand 函數產生隨機亂數教學與範例程式碼 - G. T. Wang (gtwang.org)](https://blog.gtwang.org/programming/c-cpp-rand-random-number-generation-tutorial-examples/)
19. [ChatGPT (openai.com)](https://chat.openai.com/)