**陳盈如 B05902118**

**Reference:**

**Problem2.3:**

B05902032

**Problem2.4:**

<https://www.youtube.com/watch?v=4Xyhb72LCX4>

https://www.youtube.com/watch?v=Wj606N0IAsw

**Problem3:**

B05902127 / B05902041 / B05902074

**Problem4:**

B05902074

**1.1**

(a)4

(b)2

(c)3

**1.2**

6 2 7 4 8 1 9 5 10 3

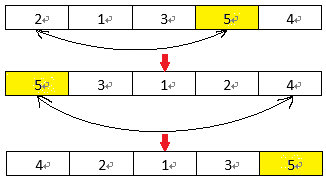
**1.3**

**(a)**

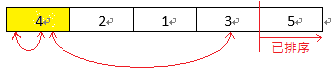
利用1.2的algorithm（sorting左至右由小至大），從最大值開始排序，每次呼叫FLIP function的時候必定一次呼叫兩次function，從第一個數字開始讀，尋找未排好序列的最大值，找到最大的數字後先flip一次將最大值flip到最前面，再flip一次將最大值flip到未排序的數字的最後面（如圖(一)），已經排序好的數值就不需要去考慮，繼續重複找下一個最大值。

總共N個數，只要排2到N - 1個數，1就會自動排好。每個數字需要flip兩次，從原本位置flip到最前面，再flip到最後面，flip的次數為2(N - 1)次，但是當最大值已經在最前端時，flip到最前面的code就等於垃圾一樣沒做事（如圖(二)），因此N個數字排列，呼叫FLIP function的次數不會超過2N - 2，pancake number就不會超過2N - 2。

圖(一)



圖(二)



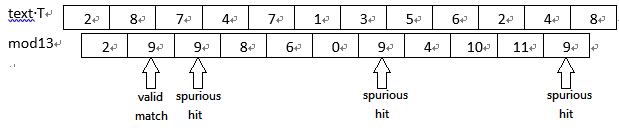
**(b)**

N個數字的所有排列方法數，裡面最少有一種情況的pancake number = N – 1。

沿用(a)的algorithm的想法，N個數字的pancake number必定不會大於2N – 2，因為會有FLIP function沒有作用的時候，那所有N個數字的排列方法數中一定會有一種排列方式是每次在未排序數字中找到的最大值都在第一個位置，也就是說(a)圖(二)的情形會發生在每一個數字。因此N個數字中需要排列的N - 1個數字都只需要將最前端的數字flip到未排序數字的最後面，pancake number就會是N – 1。

**2.1**

87(mod 13) = 9



**2.2**

struct ListNode {

int data;

struct Listnode \*next;

};

typedef struct ListNode ListNode;

ListNode \*new, \*head, \*tail;

head = (ListNode\*)malloc(sizeof(ListNode));

1

2

3

4

5

6

7

8

assert(head != NULL);

head->data = T[T0.length - 1];

head->next = NULL;

tail = head;

for i = T0.length - 2 to 0

new = (ListNode\*)malloc(sizeof(ListNode));

new->data = T[i];

new->next = head;

head = new;

tail->next = head;

while head->data != T[0]

head = head->next;

for i = 1 to T.length – 1

head = head->next;

if T[i] != head->data

return false;

return true;

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

**2.3**

pi[m + 1] = {0};

pi[1] = 0;

k = 0;

1

2

3

for q = 2 to m

while k > 0 and X[k + 1] != X[q]

k = pi[k];

if X[k + 1] == X[q]

k++;

pi[k] = k;

for q = m to 1

if pi[k] == 1

return m/k;

4

5

6

7

8

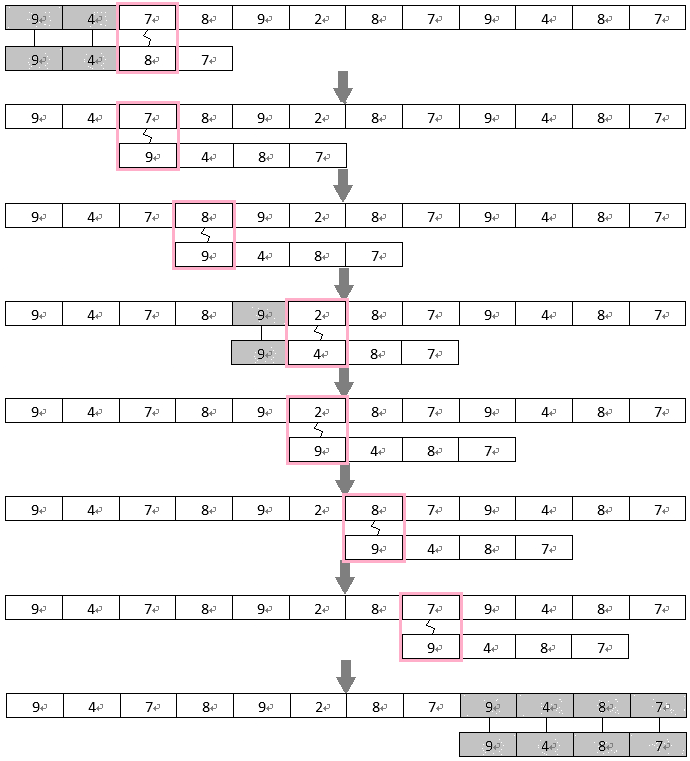
9

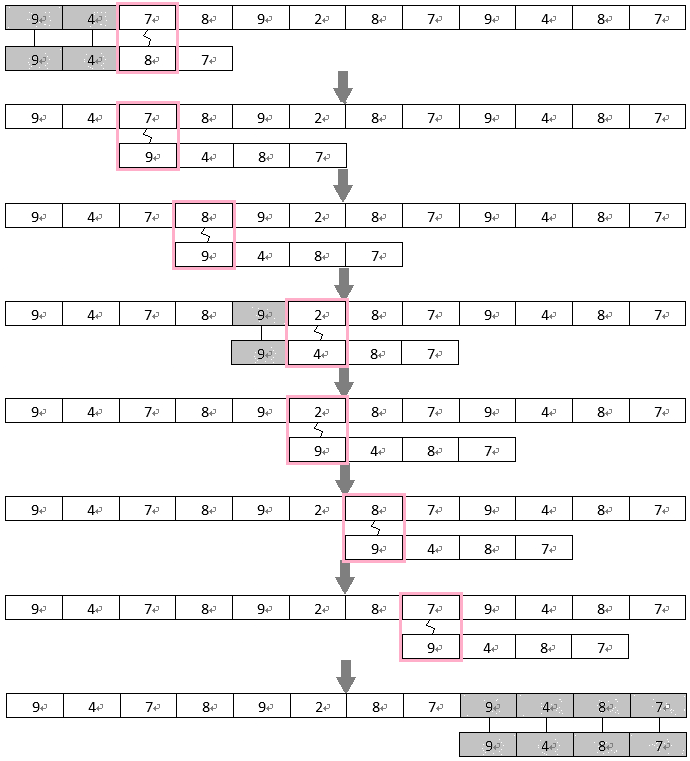
10

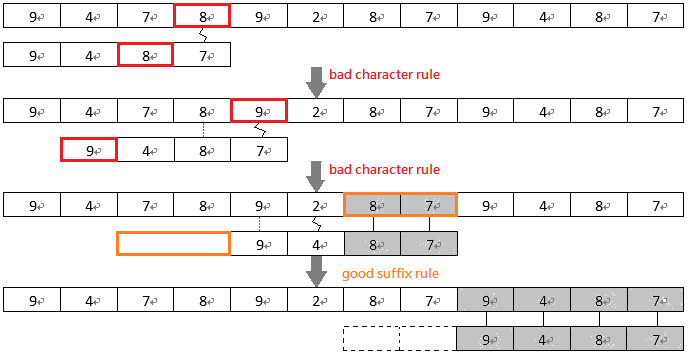
11

12

**2.4**

*KMP* algorithm:

*Boyer-Moore* algorithm:



第一個箭頭**：**

從最後一個數字開始配對，發現8就無法配對到了，利用BC(The Bad Character Rule的縮寫)可以向右移動1，然後因為根本沒有一樣的suffix所以無法利用GS(The Good Suffix Rule的縮寫)。

第二個箭頭**：**

與第一個箭頭一樣，最後一個數字9就無法配對，BC = 3，沒有suffix所以也無法利用GS。

第三個箭頭**：**

無法配對的數字是2，P裡面沒有這個數字，因此利用BC直接將整個P跳過2，BC = 2，GS = 4，因為分別利用BC或GS發現GS可以跳過較多不必要的比較，所以最後選擇用GS。