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

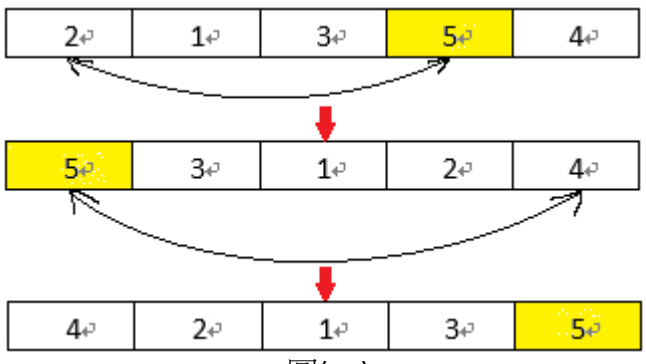
6 2 7 4 8 1 9 5 10 3

1.3

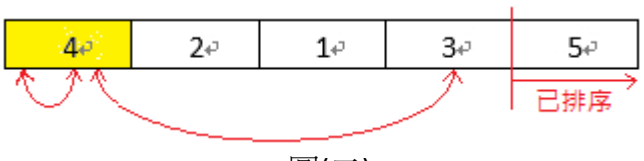
(a)

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)

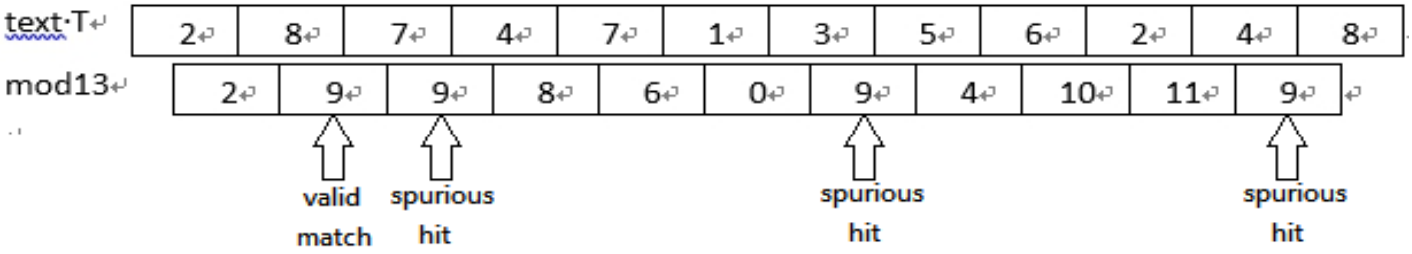
(b)

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

```
1  struct ListNode {
2      int data;
3      struct Listnode *next;
4  };
5  typedef struct ListNode ListNode;
6
7  ListNode *new, *head, *tail;
8  head = (ListNode*)malloc(sizeof(ListNode));
```

```
9   assert(head != NULL);
10  head->data = T[T0.length - 1];
11  head->next = NULL;
12  tail = head;
13  for i = T0.length - 2 to 0
14      new = (ListNode*)malloc(sizeof(ListNode));
15      new->data = T[i];
16      new->next = head;
17      head = new;
18  tail->next = head;
19  while head->data != T[0]
20      head = head->next;
21  for i = 1 to T.length – 1
22      head = head->next;
23      if T[i] != head->data
24          return false;
25  return true;
```

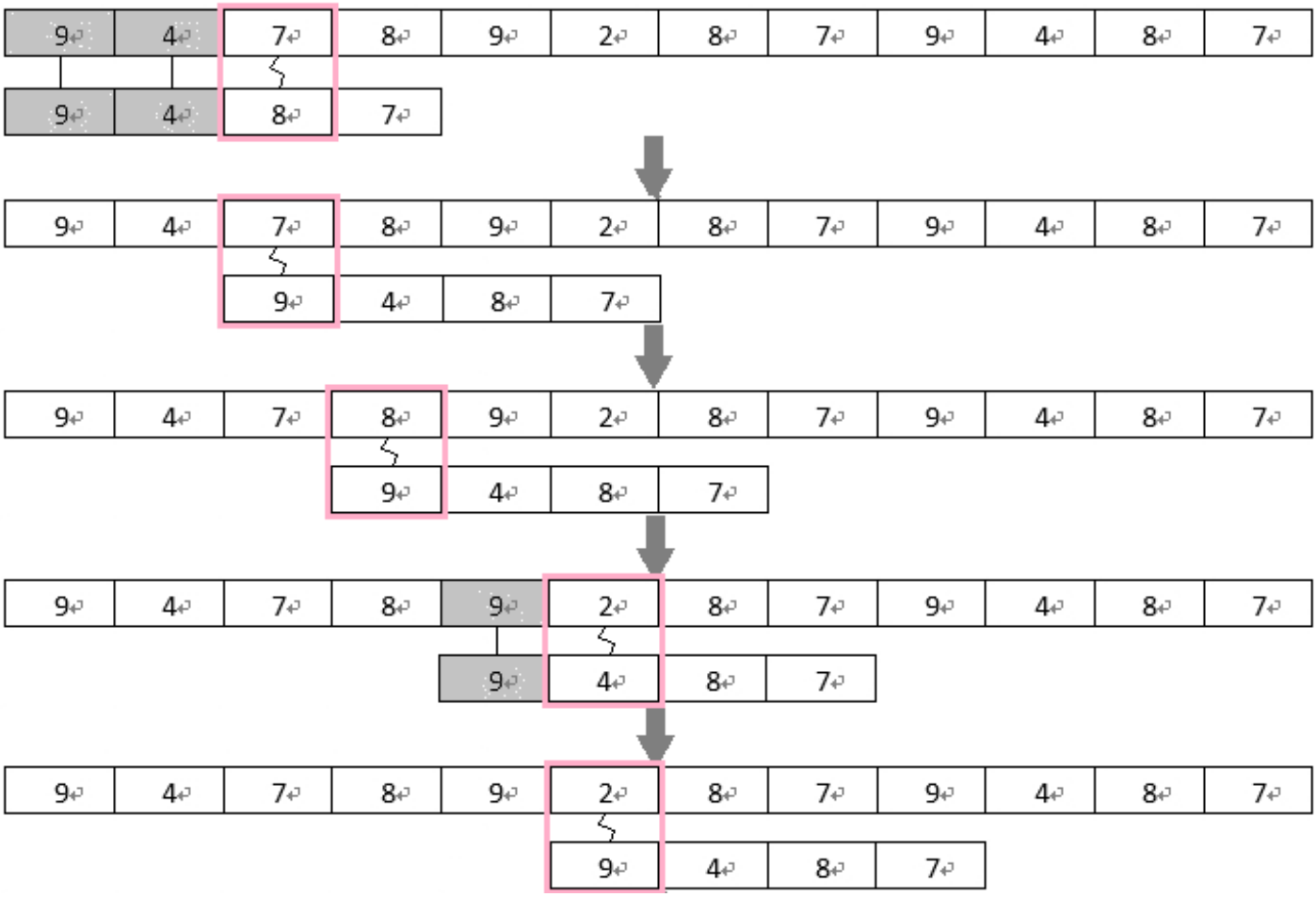
2.3

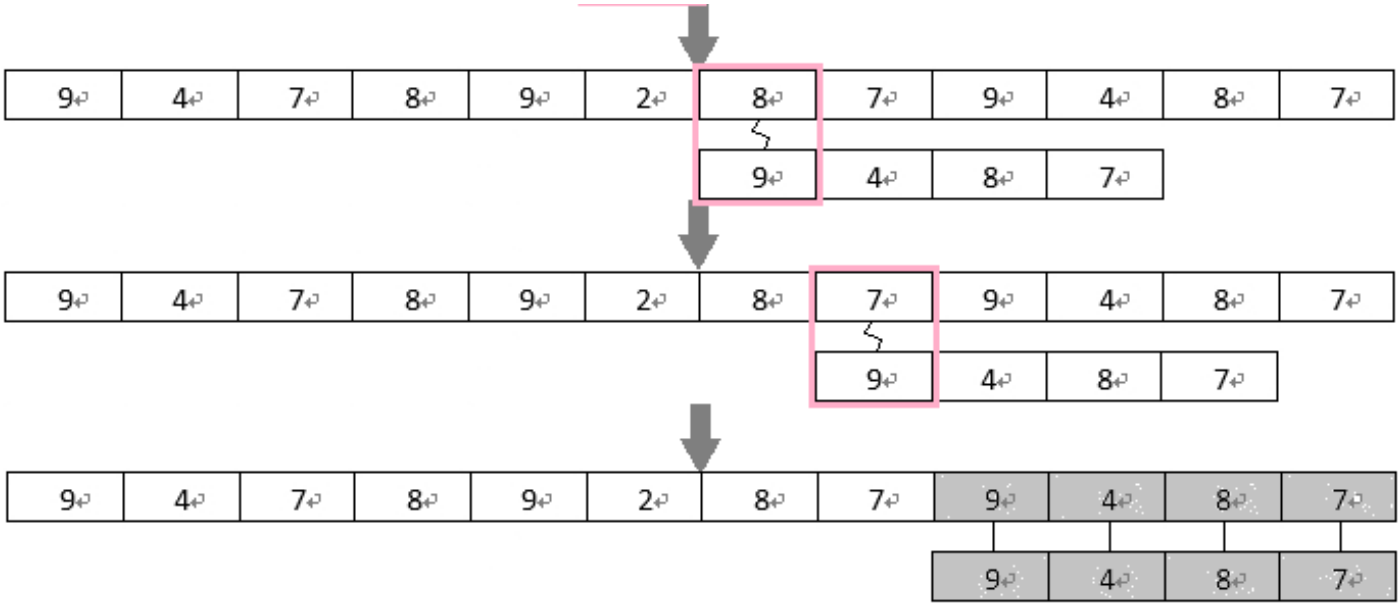
```
1   pi[m + 1] = {0};
2   pi[1] = 0;
3   k = 0;
```

```
4   for q = 2 to m
5       while k > 0 and X[k + 1] != X[q]
6           k = pi[k];
7       if X[k + 1] == X[q]
8           k++;
9       pi[k] = k;
10  for q = m to 1
11      if pi[k] == 1
12          return m/k;
```

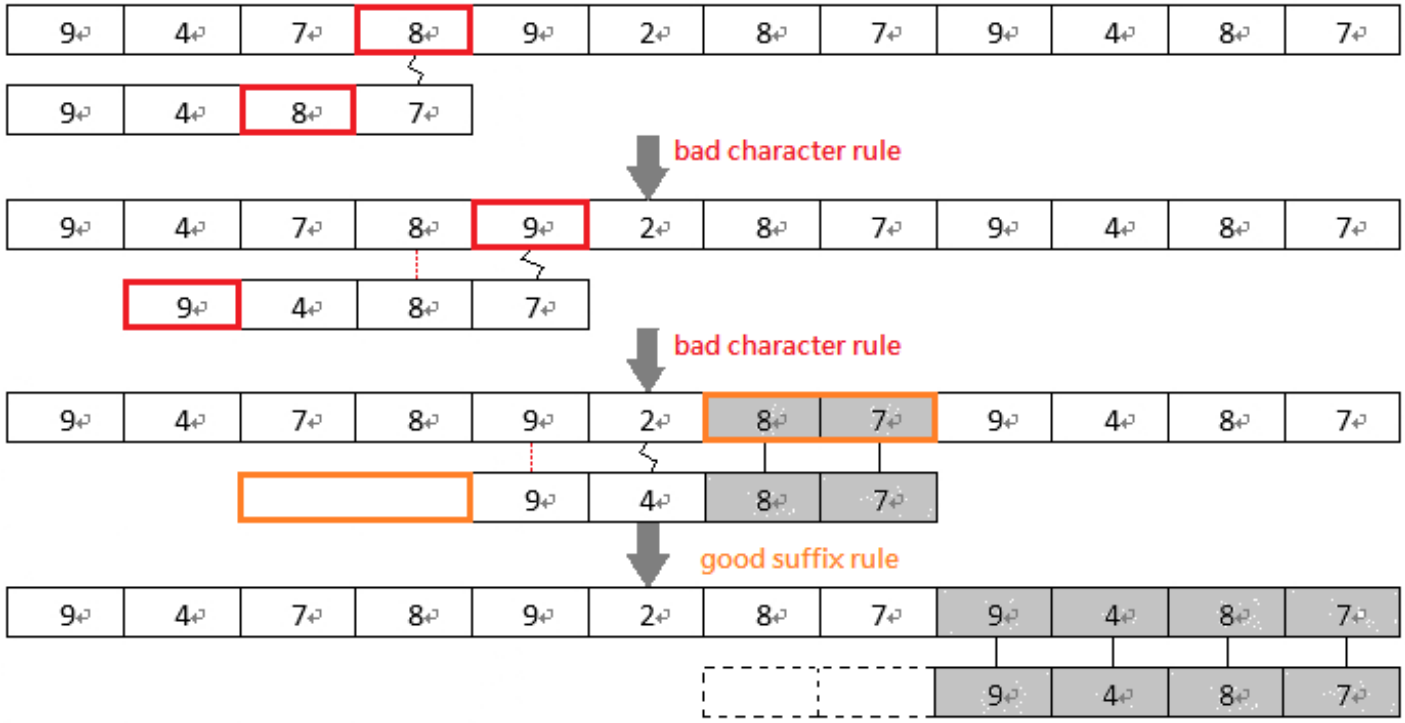
2.4

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。