

Problem 1

1. 不，學生所證為一不同之時間複雜度 $O(n^3)$ ，而並無法證明 $O(n^2)$ 。

→ 只要證明存在某 $f(n) = O(n^3)$ 不屬 $O(n^2)$ 即成立

①

for 大 D 設定，存在一正 C, n_0 ，當 $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$
則可設 $f(n) = O(g(n))$

② 考慮一函數 $f(n) = O(n^3)$ 存在，若想證其不屬於 $O(n^2)$

則需考慮 $f(n) = n^3$ ，對任何正常數 C, n_0 ，當 $n > n_0$ 則有式子：

$$f(n) = n^3 \leq C \cdot n^2$$

欲使不等式成立，我們需

$$\textcircled{1} \quad n^3 \leq C \cdot n^2$$

$$\textcircled{2} \quad n \leq C$$

③ 對 $n > C$ 時，上述兩條件之算式顯然不成立（ $\textcircled{2}$ 即失效了！）

i. 對 $f(n) = n^3$ ，它不屬於 $O(n^2)$

故即便學生證明 $O(n^3)$ ，仍無法證明 $O(n^2)$ ，學生不得滿分。

2.

① 根據 Binary-Search 在課堂定義，pseudo-code 略更改

$$\text{原's: } m = l \xrightarrow{\text{底以 4\%}} m = (l+r)/2$$

② 此算法，找到錯誤則只能空閒縮小 1。（極無效率），

故最壞情形則需找整串才找到 $\Rightarrow O(n)$ # 

若原二元方法則為 $O(\log n)$ ，現則堪為 $O(n)$

3. 由定義下手 start prove:

① Big theta₍₀₎ → 存在 C_1, C_2 和 n_0 (for all $n \geq n_0$)，使 $f(n)$ 存在： $n^2 \cdot C_1 \leq f(n) \leq n^2 \cdot C_2$

② 同除 n^2 : $C_1 \leq (f(n)/n^2) \leq C_2$

③ 同取 \lim : $\lim_{n \rightarrow \infty} C_1 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{n^2} \leq \lim_{n \rightarrow \infty} C_2$

④ 化簡: $C_1 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{n^2} \leq C_2$

⑤ $c_1 \leq \lim_{n \rightarrow \infty} \frac{f(n)}{n^2} \leq c_2$ 則表示 $\lim_{n \rightarrow \infty} \frac{f(n)}{n}$ 因 c_1, c_2 为常数而为一常数

$\therefore \lim_{n \rightarrow \infty} \frac{f(n)}{n^2}$ 为常数, 則 $f(n) = \Theta(n^2)$ 才得成立

\Rightarrow **反证法證明**: 若 $f(n) = \Theta(n^2)$, 則 $\lim_{n \rightarrow \infty} \frac{f(n)}{n^2} = c$, 其中 $c > 0$ 为常数

4. 欲證明 $\lg n = O(\sqrt{n})$ 則可考慮若 n 逐渐增長, 則函數值之傾向.

a) 由 $n=1, 2, 3, \dots$ 之兩者值, 想知 \sqrt{n} 是否一直比 $\lg n$ 增長更快

b) 依定義, 則需有存在 C 值 ($\geq C$) 使 $\lg n \leq C\sqrt{n}$

若 $n=4, C=1$, 且

	$\lg n$	$C\sqrt{n}$	關係	
$n=4$	2	2	=	發現應該 \leq , 故該不等式不成立!
$n=5$	2.32	2.24	\geq	
$n=6$	2.58	2.45	\geq	
$n=7$	2.81	2.65	\geq	
⋮				

故 $\lg n = O(\sqrt{n})$ 为不正確

5. 試證 $\sum_{i=0}^n i^n = O(n^n)$ 是否成立, 則一樣回歸定義:

\rightarrow 存在 C 與 n_0 , 在 $C > 0, n_0 \geq 1$ 情況下, 使 $\sum_{i=0}^n i^n \leq C \cdot n^n$ 成立。

① 因總和之最大項為 n^n . 且總和中共有 $n+1$ 項, 故可取其 \max 為其上限

$$\sum_{i=1}^n i^n \leq n \cdot n^n$$

② 若 $C=n$, $n_0=1$, 則不等式對所有 $n \geq 1$ 都成立

$$\sum_{i=1}^n i^n \leq C \times n^n$$

\Rightarrow 故得證. $\sum_{i=1}^n i^n = O(n^n)$ 成立

b.

Problem 1126. Assume that f, g are positive functions and $|\lg(f(n)) - \lg(g(n))| = O(1)$, prove that $f(n) = O(g(n))$.

B11902777's Solution.

$|\lg(f(n)) - \lg(g(n))| = O(1) \Rightarrow |\lg(f(n)) - \lg(g(n))| = c$, for all $n > n_0$, where $n_0, c > 0$ are constants.

$$|\lg(f(n)) - \lg(g(n))| = c \Rightarrow \lg\left(\frac{f(n)}{g(n)}\right) = c \Rightarrow \frac{f(n)}{g(n)} = 2^c$$

Take $c' = \frac{1}{2^c}$, we have $f(n) \leq c'g(n)$ for all $n > n_0$, QED.

key wrong:

這邊推論是錯的，因為反證法，

$$|\lg(f(n)) - \lg(g(n))| \leq c$$

Re-Proof

① Given $|\lg(f(n)) - \lg(g(n))| = O(1)$ exist constant $c > 0$ and $n_0 \geq 1$, 使對於所有 $n \geq n_0$:

$$|\lg(f(n)) - \lg(g(n))| \leq c \quad (\text{不是} \leq c)$$

② 故有不等式存在:

$$-c \leq \lg(f(n))/g(n) \leq c$$

③ 幾何指數函數 (γ 像底) 應用於不等式的所有部分:

$$2^{-c} \leq f(n)/g(n) \leq 2^c$$

④ 不等式可看出, $f(n)$ 上界 γ 是 $g(n)$ 乘常數

$$\Rightarrow f(n) \leq 2^c \cdot g(n)$$

故為常數

⑤

$\therefore \exists$ 有常數 2^c , 使得對於所有 $n \geq n_0$, $f(n) \leq 2^c \times g(n)$

\therefore 故可得 $f(n) = O(g(n))$ 終結

Problem 2

/>「是找尋位置之最快方式, $O(n) = \log n$

1. 請利用 Binary Research, 並用 sorted array 進行排序, 遇到無值者回傳.

Pseudo Code

```
*include < stdio.h>                                // 是第几, index
void Binarysearch ( int sortarray[], int start, int end, int lastone )
    int midIndex = (start + end) / 2
    int midElement = sortarray [0] + midIndex + lastone
    // 第元素      // 中間位置偏移量 // 當前元素數量

    if (int start <= int end) {
        return lastone
    }
    if (sortarray [midIndex] = midElement) {           // 若相等則在右半(尚未發現);
        return binarysearch (sortarray, mid+1, end, lastone+1) // 不相等則在左半(已出現, 便 index)
    }
    else {
        return binarySearch (sortarray, start, end, lastone)
    }
}

void Findmissingone ( int sortarray[], int *FirstI, int *SecondI, int size ) // create func. of finding
    int firstindex = BinarySearch (Sortarray, 0, size, 0)
    *FirstI = Sortarray [0] + firstindex
    *SecondI = *FirstI + 1

int main () {
    int sortarray []
    int size
    int firstI, secondI
    Findmissingone (Sortarray, size, &FirstI, &secondI)
    print firstI, secondI

    return 0
}
```

sol: 因同樣於計算大O時主要用 BinarySearch 種類,
故大O為: $O(\log n)$ #

2.

- ① 使用布林去 return true/false 看是否兩次皆成功，在 main 中傳回
② 每配對到一組皆命名兩元素為一（做標示），若完成後擇一次數列
做 examd

```
# include <stdio.h>
```

```
# include <stdbool.h>           // 整數數組           // 數組大小在 main 中回傳。
```

```
bool whether_is_test_case (int testcase[], int n) {
```

```
    int i = 0;
```

```
    int j = 1;
```

```
    while (j < n) {
```

```
        if (testcase[i] == -1) {
```

```
            i++;

```

```
        } continue;
```

```
        if (testcase[j] == -1) {
```

```
            j++;

```

```
        } continue;
```

// 當後若配對成功者，則標示為-1，否則原值

而-1標示則跳過 scan.

```
        if (testcase[i] * 2 == testcase[j]) {
```

```
            testcase[i] = -1;
```

```
            testcase[j] = -1;
```

```
            i = 0;
```

```
            j = 1;
```

```
        } else {
```

```
            i++;

```

```
            j++;

```

```
            if (j == n) {
```

```
                break // 結束 scan
```

```
        }
```

```
}
```

// 每個/組數字若配對到

則標為-1：

```
for (int k = 0; k < n; k++) {
```

```
    if (testcase[k] != -1 {
```

```
        return false;
```

```
}
```

// 判斷是否 true/false

由掃完是否全-1決定！

```
    else {  
        return true  
    }
```

```
int main {  
    int testcase[] = [ // 自由填入或输入数据 ]  
    int n = sizeof(testcase) / sizeof(testcase[0])  
    printf("Test case : ", whether_is_test_case(testcase, n) ? "True" : "False");  
    return 0;  
}
```

⇒ time complexity $O(n)$.

3. 用 2 个 link list 分别接受，不接受，及
2 个指针 current, selected .

Sol: function merge_list(accepted, unaccepted)

merge = new ListNode

merge.next = NULL

current = merged

```
while accept != NULL or unaccepted != NULL // #selected 指向当前结点  
if accept is NULL  
    selected = unaccepted  
    unaccepted = unaccepted.next  
else if unaccept is NULL  
    selected = accepted  
    accepted = accepted.next  
else  
    if accepted submission-id > unaccepted submission-id  
        selected = accepted  
        accepted = accepted.next
```

else
 selected = unaccepted
 unaccepted = unaccepted.next
 L end if
 selected.next = current.next // 每次判断后，更进一步
 current.next = selected

L end while
 return merged.next
 end function

4. 分正負向 linked list 後，兩者排序！

function fixlist(head)
 positiveHead = null
 negativeHead = null
 current = head // 用指標不斷 point
 while current != null
 if current.data >= 0
 push positiveHead, current, data
 else
 push negativeHead, current, data
 end if
 current = current.next
 end while

```

int submission_id;
struct ListNode *next;
} ListNode;

ListNode *merge_lists(ListNode *accepted, ListNode *unaccepted) {
  ListNode *merged;
  merged.next = NULL;
  ListNode *current = merged;
  ListNode **selected;

  while (accepted || unaccepted) {
    if (*accepted) {
      selected = accepted;
    } else if (*unaccepted) {
      selected = unaccepted;
    } else {
      selected = accepted;
    }

    if (*selected) {
      selected = (*selected)->next;
    }
    ListNode *next_node = (*selected)->next;
    (*selected)->next = current->next;
    current->next = selected;
    selected = next_node;
  }
}
  
```



// 先把目前正項
序列分別都為二序列，

negativeHead = Reverse(negativeHead)

// 確保傳真向數字
以次排序

// 新序列
結合！

new = positiveHead
 while positiveHead != null
 positiveHead = positiveHead.next

// 確保工看正與次序性
連續
 ① 用while / for as concurrent
 ② 用if條件 + for 進行判斷。

end while

positiveHead.next = negativeHead

return 0

Problem 3

$$5 + \underline{1}2 - 40 + \underline{1}$$

1. $5 + 3 \times 4 - (8 \times (2+3) - (1+9 \times 6)/5)$

$$\begin{aligned} & 534x + (823x + x - 5196x + 1) - \\ & - 196x + 51 - \end{aligned}$$

* 5/10 放前 5/10
1/5 放後 1/5

⇒ Ans:

$$534x + 823x + x 196x + 51 - \#$$

~~534~~
~~196~~
~~1~~ 823

2.

$$527x + 634 - x 486x + 41 -$$

4. 55
17, 40, 51
17 - (40 - 11)

用 stack 中 push 和 pop 作

$$\begin{bmatrix} 5, 2, 7 \\ 54, 14x \\ 19 \end{bmatrix}$$

$$\begin{bmatrix} 19, 6, 3, 4 \\ * -1 \\ -6 \end{bmatrix}$$

$$\begin{bmatrix} 19, -6, 4, 8, 6 \\ | + | \times | \\ 52 48 \end{bmatrix}$$

$$\begin{bmatrix} 19, -6, 52, 4 \\ | - | \times | \\ -13 \end{bmatrix}$$

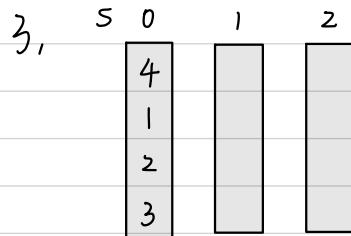
$$\begin{bmatrix} 19, -19 \end{bmatrix}$$

$$\begin{bmatrix} 38 \end{bmatrix}$$

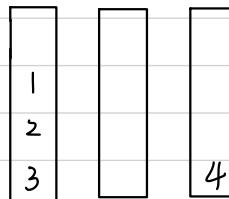
Ans: 38

#

Stack



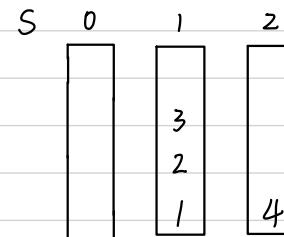
Step 1, 从 stack1 pop out 4, 然后 push 在 stack2



Step 2, 从 stack1 pop "1" 然后 push 在 stack2,

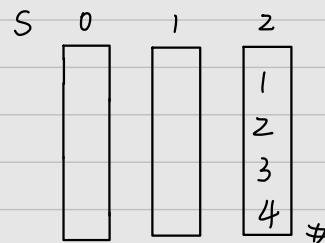
← → pop "1" ← → (→)

← → pop "2" ← → (→)



Step 3 与 Step 2 依序相同

Stack1 pop out, 然后 push 到 stack2 中。



完成，其描述可行！

4. 可设计最小 $O(n^2)$ 算法 (要求 $O(n^5)$ 的话可依另外加迴圈
增加複雜度, 但意義不大...)

① 初始化一個計數器變量 $count = 0$

② 計數先將 $stack_0 \rightarrow stack_1$ 中紀錄最大值 'max';
 $stack_1 \rightarrow stack_0$ 放回(或不一定需要) $\Rightarrow O(N \text{ or } 2N)$

③ 重複以下 Steps n 次 (用迴圈)

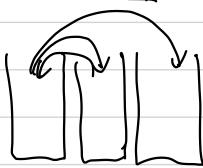
a. Stack 中，從 0 推出放到 2，再把最大的推出到 n $\Rightarrow O(n^2)$

b. 將最大元素從 $stack_0$ pop out 並 push 入 $stack_2$

c. 其它除最大元素的元素皆從 $stack_2$ 轉移至 $stack_0$

d. 進堆 'count' (++)

③



$$\text{而 } O(n)_{\text{final}} = O(n/2n) + O(n^2)$$

$$\text{故 } O(n)_{\text{final}} = O(n^2) \#$$

5. 初步位置: $[0, 10, 15.5, 17]$

1st 同學: $[0, 5, 10, 15.5, 17]$

2nd ++: $[0, 5, 10, 10.75, 15.5, 17]$

3rd ++: $[0, 5, 1.5, 10.75, 15.5, 17]$

(也可)

∴ 第 3 位同學

騎腳踏車最短距離 = 2.5

6.

func distant_to_nearest_bike(nowbike, m)

distant = new array size (n-1)

//先走遍歷每行 bike

for $i=1$ to $n-1$

distant[i-1] = existing-bike[i] - existing-bike[i-1]

end for

Sort distant in decenting order //使其保持 1st 位置

for $i=1$ to $m-1$ // m 为常数的

max-distant = distant[0]

split-distant = max-distant / 2

distant[0] = split-distant

// 然后的 split 要保持降序排列，再 sort 一次

Sort distant in decenting order

end for

return distant[0] / 2

end function

分析/proof:

步驟
分析

相鄰自行車距離 $\rightarrow O(n)$

距離 sorting $\rightarrow O(n^* \log(n))$

更新 $m-1$ 次距離 $\rightarrow O(m^*(n-1))$
(m 次 $n-1$ 律)

故 Time Complexity = $O(n) + O(n^* \log(n)) + O(m^*(n-1))$

$\therefore \underline{O(m+n)}$ (# $n, m \rightarrow \infty$)

空
間

distant 整組大小 $n \rightarrow O(n)$

m 個學生撥放 $\rightarrow O(m)$
($1 \rightarrow m$)

故複雜度 $O(n) + O(m) = \underline{O(n+m)}$ (#