B10815057 Algorithm Homework 4

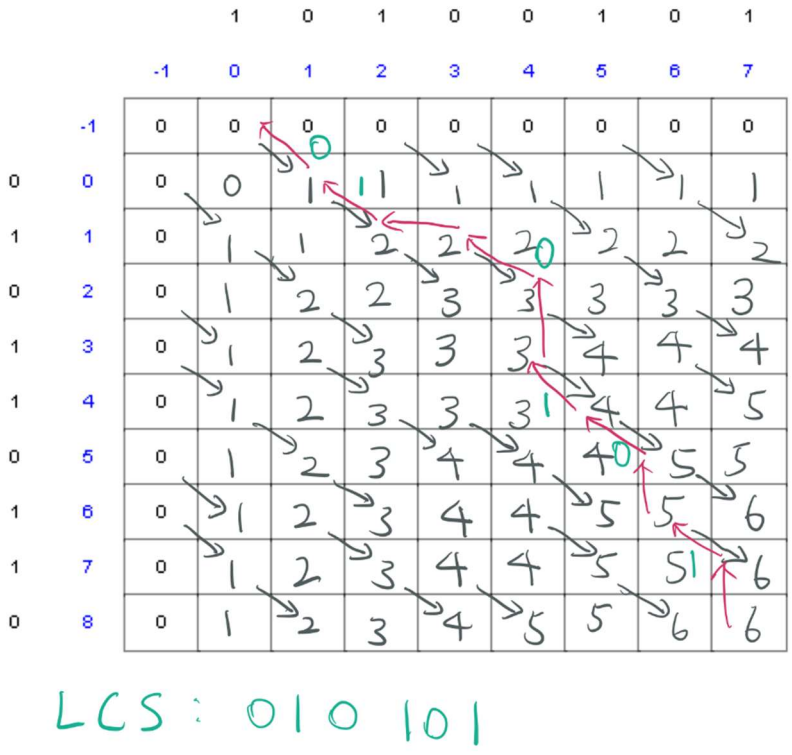
4.1

那兩行的作用是判斷是否有已經算過的紀錄，如果有就回傳之前的計算結果，因此刪除那 2 行後，就會不斷的重複做相同的運算，導致效能低落。

4.2

可表示成 optimal substructure，因為該問題可以將大問題的答案拆分成許多小問題的答案，假設小問題可以找到更多的純量乘法數量，則大問題的最佳答案會被推翻，因此該假設不成立，則反證該問題的最佳答案拆分後會是小許多的最佳答案，這就是 optimal substructure。

4.3



4.4

設輸入 sequence：[-1, 3, 4, 5, 2, 2, 2, 2]

(1) simple increasing subsequences

① 配置與輸入 sequences 同一大小的陣列 (初始化為1)

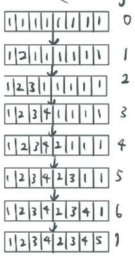
-1	3	4	5	2	2	2	2
1	1	1	1	1	1	1	1

② 變數 index i, j

i：用於迭代 index j 之前的 element

j：用於迭代陣列的每個 element

③



```
for j ← 0 to length(array)
    for i ← 0 to j
        if sequence[j] ≥ sequence[i]
            array[j] ← array[i] + 1
```

j 跑過陣列每個 element

i 跑過 index j 之前的 element

若 index j ≥ index i 則將 array[j] 設 array[i]+1

④ 從陣列中找到最大值，並往回收集第一個遇到的數字直到 1，結果就是 longest increasing subsequence

-1	3	4	5	2	2	2	2
1	2	3	4	2	3	4	5

answer：-1, 2, 2, 2, 2

(2) strictly increasing subsequence

①②步驟同上

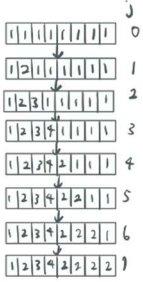
③

```
if sequence[j] ≥ sequence[i]
    array[j] ← array[i] + 1
```

改為

```
if sequence[j] > sequence[i]
    array[j] ← array[i] + 1
```

④



-1	3	4	5	2	2	2	2
1	2	3	4	2	2	2	2

answer：-1, 3, 4, 5

4.5

least duration:選時間最少的並不能得到好的結果，因為雖然時間少，但卻可能與許多 activity 重疊，進而大幅減少可選的數量，得到的解就不是最佳的。

overlaps the fewest:選與其他 activity 重疊最少的 activity 也不能得到好的結果，因為有可能該 activity 占用的時間很少但卻與少數的長時間 activity 重疊，導致最終總時間很少，也不符合最佳的解。

earliest start time:不斷地選最早結束的 activity 可以找到最佳解，因為透過觀察可以得知，activity 結束後就不會與任何 activity 重疊，所以讓 activity 越早結束，後面可放的 activity 數量與時間就越大，這個方法兼顧 activity 數量與時間，比起上面的 2 個方法，較不會出現極端情形，也更可以得到最佳解。