r10525103_homework1

Problem 0 :

- Problem 1 : all by myself
- Problem 2 : all by myself
- Problem 3 :
    1. all by myself
    2. discord討論區
    3. all by myself
- Problem 5: all by myself

Problem 1

1. i會以指數的速率成長，在時間上相當於對n取log，故時間複雜度為Θ(lg(n))
2. 每次的遞迴皆會產生兩個新任務，相當於2開n次方，故時間複雜度為Θ(2^n))
3. 兩個迴圈的分別會做logn以及n次，故時間複雜度為Θ(nlg(n))
4. True, g(n)之時間複雜度必≥f(n)之時間複雜度，故f(n).g(n)必≥(f(n))^2，也就是f(n).g(n)=Ω((f(n))2)
5. True, lg2及ln2的時間複雜度相同，故題目可替換成ln2 = O(n^k)。

   對兩邊作為分可得1/n = O(kn^(k-1)) ⇒ 1 = O(kn^k)

   其中O(kn^k)必為正數，也就是1 = O(kn^k)，故為正確
6. True, 因asymptotically non-negative functions的時間複雜度關係不會因取log而有差異，故為正確

Problem 2

1. 12+345*+*93/75*++

   除非遇到括號，否則依照先乘除後加減的順序，將運算符號放置到兩個運算的數字之後

2. (((1+2)*(5-3))*6)/5

   每遇到一個運算符號，便把前面兩個已計算好的運算元拿來做運算

3.

1	NIL	NIL	NIL
1	5	NIL	NIL
1	5	3	NIL
NIL	5	3	NIL
NIL	5	3	4
6	5	3	4
6	NIL	3	4

4. 從最右邊開始的連續三個1，和結果左邊的連續三個1可以用共用既有的牌，其餘必須透過換牌來取得。其數量為10。
5. 選定一點開始照順序看數字。先將第一個的數字push進stack，接下來，若遇到和stack最上方相同的數字，就對他做pop，不同的數字就做push。當繞完一圈時，若stack中還有數字，則圈圈中有交叉，
   若沒有，則沒有交叉。
6. 每當經過一個數字，只要檢查stack最上方數字和當前數字是否相同，並做對應動作即可。共需做2n次，故時間複雜度為O(n)

Problem 3

1.

```
a = L.head
b = L.head
while a ->next -> next != NULL
      a = a -> next -> next
      b = b -> next
return b
```

當a往前兩步時，b往前一步，則當a之後兩點不存在時，也就代表b到了線的中間

只要循著線走一次即可知道結果，故time complexity = O(n)

另外需要兩個空間存a, b，故extra-space complexity = O(1)

2.

```
令arrary之index從1開始計算
a = 0
b = 0
for i = 1, 2, ... n
  if array[i] != 0 //依序檢查array之內容
    a = array[i]   //當內容!=0，則取其值為a
    b = array[a]   //檢查array[a]的內容，並用b來存
      while b != 0
        b = array[b] //先將取得的值暫存，並將其存取的位置的內容設為0。我們透過將index內之
        array[a] = 0 //內容設為0，來紀錄在數字租出現的順序中存在此值。重複以上步驟。
for i = 1, 2, ... n  //當以上流程做完後，我們重新檢查array，若遇到
  if i != 0          //不為0的值，則我們知道他為缺少的值
    return i
```

檢查n次，故time complexity = O(n)

另外需要兩個空間存a, b，故extra-space complexity = O(1)

3.

```
令arrary之index從1開始計算
left_weight = 0                 //分別維護左右weight
right_weight = 0
for i = array.length, array.length-1, ... 2   //對array做初始化
  if i == array.length
    array[i] = array[i]
    right_weight += array[i]
  else
    array[i] = array[i] + array[i+1]//除array[array.length]及[1]之外，每個array[i]
                                    //的內容，皆為original array[i] += array[i+1]
    right_weight += array[i]

for i = 1, 2, ... array.length              //從1開始依序對左右weight做比較
  if right_weight == left_weight            //若相等，則回傳i
    return i
  else
    if i == 1
      left_weight += array[i]                //針對array內容做變化，並改變weight
      right_weight -+ array[i+1]
    else
      array[i] = array[i-1] + array[i] - array[i+1]
```

```
        left_weight += array[i]
        right_weight -+ array[i+1]

return 0                                                    //若找不到pivot，則回傳0
```

以下以內容為3, 6 , 1, 9, 5的情境，畫圖展示流程邏輯