

HW2

Prob1

1. 先將 array 分為前半部及後半部
2. 利用一個迴圈同時查找陣列中有沒有 target
3. 若是找到了再判斷是否為第一個(或最後一個)，再更新
4. 若無則 output:[-1, -1]

時間複雜度：

原先認為這個方法算是用二元搜尋了，解果認真思考之後發覺應該是 $O(n)$ ，因為雖然說只利用了一個 for 迴圈同時找尋前半部以及後半部 array，但實際上市程式執行仍然是遍歷了整個 array，所以並沒有將效率提高到 $O(\log n)$ 。

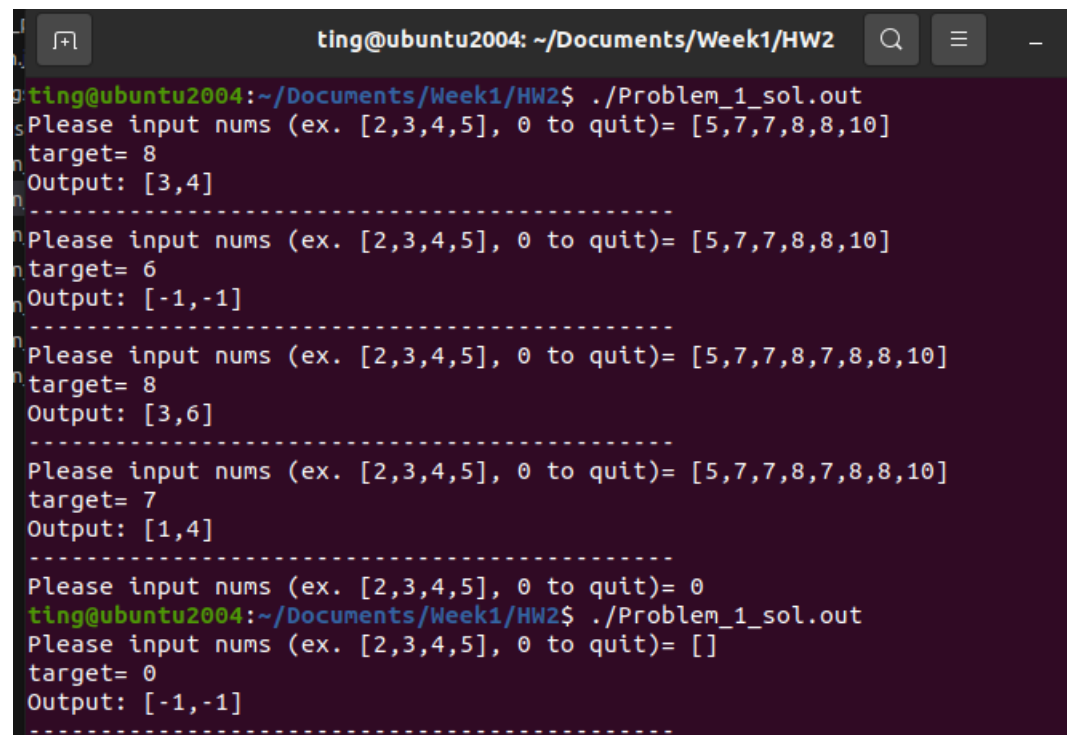
測試數據：

[5, 7, 7, 8, 8, 10], target = 8, Ans:[3, 4]

[5, 7, 7, 8, 8, 10], target = 6, Ans:[-1, -1]

[5, 7, 7, 8, 7, 8, 8, 10], target = 8, Ans:[3, 6]

[5, 7, 7, 8, 7, 8, 8, 10], target = 7, Ans:[1, 4]



```
ting@ubuntu2004: ~/Documents/Week1/HW2
ting@ubuntu2004:~/Documents/Week1/HW2$ ./Problem_1_sol.out
Please input nums (ex. [2,3,4,5], 0 to quit)= [5,7,7,8,8,10]
target= 8
Output: [3,4]
-----
Please input nums (ex. [2,3,4,5], 0 to quit)= [5,7,7,8,8,10]
target= 6
Output: [-1,-1]
-----
Please input nums (ex. [2,3,4,5], 0 to quit)= [5,7,7,8,7,8,8,10]
target= 8
Output: [3,6]
-----
Please input nums (ex. [2,3,4,5], 0 to quit)= [5,7,7,8,7,8,8,10]
target= 7
Output: [1,4]
-----
Please input nums (ex. [2,3,4,5], 0 to quit)= 0
ting@ubuntu2004:~/Documents/Week1/HW2$ ./Problem_1_sol.out
Please input nums (ex. [2,3,4,5], 0 to quit)= []
target= 0
Output: [-1,-1]
-----
```

Prob2

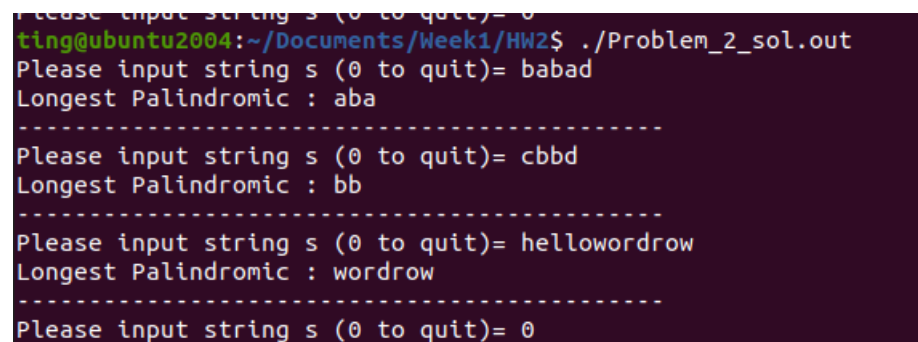
1. 用一個迴圈跑 string 中的每一個字元
2. 每次跑到的 index 視為回文的中心點查找
3. 回文分為奇數或偶數
奇數:則中間開始的左右 $\pm j$ 都必須相同, ex. abcba
偶數:則最中間的會與右邊的相同, 接續往左右類推, ex. abba
4. 接著判斷找出的回文是否為最長的, 有的話更新

測試數據:

babad ,Ans: aba

cbbd , Ans:bb

helloworldrow , Ans:wordrow



```
Please input string s (0 to quit)= 0
ting@ubuntu2004:~/Documents/Week1/HW2$ ./Problem_2_sol.out
Please input string s (0 to quit)= babad
Longest Palindromic : aba
-----
Please input string s (0 to quit)= cbbd
Longest Palindromic : bb
-----
Please input string s (0 to quit)= helloworldrow
Longest Palindromic : wordrow
-----
Please input string s (0 to quit)= 0
```

Prob3

1. 確保 arryl 比較短, 如果沒有的話將兩個順序對調
2. 特殊情況先處理:如果兩個 array 有一個為空, 直接計算其餘另一個的中位數。利用 $\%2$ 判斷機偶決定要不要相加 $/2$ 。
3. 之後使用二元搜尋
先找到切割點 cut1, cut2, 再利用切割點確定中位數的左右邊界值 (l1 和 r1 分別為 nums1 在 cut1 左右的值, 如果超出範圍則為 INT_MIN 或 INT_MAX), 最後利用邊界值長度(是奇數還是偶數)判斷要不要相加 $/2$ 。
4. 如果沒有找到適合的切割點就要調整切割點 cut1 ± 1 , 繼續查找。

時間複雜度 $O(\log(m+n))$, 使用二元搜尋, 不用將兩個 array 進行合併再遍歷, 直接使用已經排序好的兩個 array 用兩個分割點來尋找中位數的位置。

測試數據:

[1,3]/[2] , Ans:2

[1,2]/[3,4] , Ans:2.5

```
Please input string s (0 to quit): 0
ting@ubuntu2004:~/Documents/Week1/HW2$ ./Problem_3_sol.out
Please input nums (ex. [1,2], nums1-> 0 to quit)
nums1 = [1,3]
nums2 = [2]
Median : 2
Please input nums (ex. [1,2], nums1-> 0 to quit)
nums1 = [1,2]
nums2 = [3,4]
Median : 2.5
Please input nums (ex. [1,2], nums1-> 0 to quit)
nums1 = 0
```