1.

會計五
B06702064
Algorithm
HW1.

"no collaborator"

Input: A sequence of n numbers $A = \langle a_1, \cdots, a_n \rangle$ and a value v

Output: An index i s.t. $v = A[i]$ or NIL if v does not appear in A.

<Pseudo-code>

```
LinearSearch(A,v)
  for i=1 to A.length
     if A[i] == v
         return i.
  return NIL.
```

<Use loop invariant to show at the start of each iteration of the for loop, the subarray $A[1, \cdots, i-1]$ contains no "v" element>

① Initialization: Before the first loop, when $i=1$, the subarray is empty. Thus, the loop invariant holds.

② Maintenance: During each iteration, we compare $A[i]$ to v. If they are the same, we return i. Otherwise, the loop iteration continues. Thus, we know that at the end of each loop, the subarray $A[1, \cdots, i]$ must not contain element v. Therefore, the loop invariant holds. Incrementing i for the next iteration of the for loop then preserves loop invariant.

③ Termination: There are two conditions that the algorithm terminates. First, when $A[i] == v$, from the maintenance property, we know that the subarray $A[1, \cdots, i-1]$ contains no "v" element and we return i. Second, the condition causing the for loop to terminate is that $i = A.\text{length} = n$. Because we increment i by 1 in each loop iteration, we have $i = n+1$ at that time. Substituting $i = n+1$ into the sentence of loop invariant, we get that the subarray $A[1, \cdots, n]$ contains no "v" element and we return NIL. Observing that $A[1, \cdots, n]$ is the entire array, we conclude that $A[1, \cdots, n]$ doesn't contain "v". Hence, the algorithm is correct.

參考 "https:walkccc.me/CLRS/Chap02/2.1" 與課本敘述.

2.

The binary search algorithm halves the size of the remaining portion of the sequence and drops the unused portion, it yields 1 subproblem with $\frac{1}{2}$ the size of the original. Also, the divide action and the base case both take constant time. Thus, the recurrence form of time complexity is

$$T(n) = \begin{cases} \Theta(1), & n=1 \\ T(\frac{n}{2}) + \underbrace{\Theta(1)}_{\text{divide action}}, & n>1 \end{cases}$$

Assume that $c = \max$ (time of solving the problem when $n=1$, time of divide action)

$$T(n) = \begin{cases} c, & n=1 \\ T(\frac{n}{2}) + c, & n>1 \end{cases}$$

By using the recursion tree,

$T(n)$

$T(\frac{n}{2})$ = c level 1.

$T(\frac{n}{4})$ c level 2

$T(\frac{n}{8})$ c

$\vdots$

$T(1) = T(\frac{n}{2^k})$ c level k

$\Rightarrow T(\frac{n}{2^k}) = T(1)$

$\Rightarrow \frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow k = \lg n$

$\Rightarrow T(n) = c \cdot \lg n$

$\Rightarrow T(n) = \Theta(\lg n)$ #

3. 参考: "https://walkccc.me/CLRS/Chap02/Problems/2-2/"

a. We need to show that A' contains the elements in A but in sorted order.

b. for loop in lines 2-4, the loop invariant property is " at the start of each iteration of the for loop in lines 2-4, the subarray $A[j, \dots, n]$ contains the elements originally in $A[j, \dots, n]$ and the $A[j]$ element should be the smallest one in this subarray. Also, other elements may be moved into different position."

① Initialization: Before the first loop, $j = n$, the subarray contains only $A[n]$. Thus, it is trivially the smallest element.

② Maintenance: During each iteration, we compare $A[j]$ with $A[j-1]$ and we move the smaller element to $A[j-1]$. Thus, after the iteration, the length of the subarray increases by one, and the first element is the smallest of the subarray.

③ Termination: The condition causing the for loop to terminate is that $j = i$. Substituting it to the loop invariant, the subarray $A[i, \dots, n]$ contains the elements originally in $A[i, \dots, n]$ and $A[i]$ is the smallest element in the entire subarray.

3. c.

for loop in lines 1-4, the loop invariant property is that "at the start of each iteration of the for loop in lines 1-4, the subarray $A[1, \dots, i-1]$ contains the $i-1$ smallest elements in $A[1, \dots, n]$ in sorted order. Also, $A[i, \dots, n]$ contains the remaining $n-(i-1)$ elements in $A[1, \dots, n]$ that are larger than all elements in $A[1, \dots, i-1]$.

① Initialization: Before the first loop, when $i=1$, the subarray is empty. Thus, the loop invariant holds.

② Maintenance: At the beginning of the outer loop, the subarray $A[1, \dots, i-1]$ contains the $i-1$ elements in $A[1, \dots, n]$ in sorted order and $A[i, \dots, n]$ contains the elements greater than elements in $A[1, \dots, i-1]$. According to part b, after the inner loop, $A[i]$ will be the smallest element of the subarray $A[i, \dots, n]$. Thus, at the end of the outer loop (when the inner loop is finished), $A[1, \dots, i]$ contains the elements that are smaller than $A[i+1, \dots, n]$ in sorted order.

③ Termination: The condition causing the outer for loop to terminate is that $i=n$. Substituting into the loop invariant property, the array $A[1, \dots, n-1]$ contains the $n-1$ smallest elements in $A[1, \dots, n]$ in sorted order that are smaller than $A[n]$. Thus, the array $A[1, \dots, n]$ contains all elements in sorted order.

d. The worst-case of bubble sort happens when the entire array is in reverse order. In this situation, the outer loop will run $(n-1)$ iterations and the inner loop will run $[n-(i+1)+1] = (n-i)$ iterations. Thus the worst-case time complexity of bubble sort $= (n-1)+(n-2)+\dots+1$.

$= \frac{[1+(n-1)] \times (n-1)}{2} = \Theta(n^2)$, which is the same as the worst-case running time of insertion sort.

参考 https://atekihcan.github.io/CLRS/03/E03.01-01/

4. prove $\max(f(n), g(n)) = \Theta(f(n)+g(n))$

$\Leftrightarrow \exists C_1, C_2 > 0$ and $n_0 > 0$ s.t. $0 \le C_1[f(n)+g(n)] \le \max(f(n), g(n)) \le C_2[f(n)+g(n)], \forall n \ge n_0$

$\boxed{\text{for RHS}}$ $\because f(n), g(n)$ are asymptotically nonnegtive functions, we can assume that for $n > n_1$, $f(n) > 0$ and for $n > n_2$, $g(n) > 0$.

choose $n_0 = \max(n_1, n_2)$, we know that $f(n)+g(n) \ge \max(f(n), g(n))$ $\forall n \ge n_0 = \max(n_1, n_2)$

$\boxed{\text{for LHS}}$ Also, $f(n)+g(n) \le 2\max(f(n), g(n)) \Leftrightarrow \frac{f(n)+g(n)}{2} \le \max(f(n), g(n))$, $\forall n \ge n_0 = \max(n_1, n_2)$

(since $\max(f(n), g(n))$ is either $f(n)$ or $g(n)$).

Combine the two results, we get $0 \le \frac{f(n)+g(n)}{2} \le \max(f(n), g(n)) \le f(n)+g(n)$ $\forall n \ge n_0$

$n_0 = \max(n_1, n_2)$

Thus, $\max(f(n), g(n)) = \Theta(f(n)+g(n))$

5.

By sterling's approximation, $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \in \Theta(n^{n+\frac{1}{2}} \cdot e^{-n})$

$$(\log n)! \in \Theta\left((\log n)^{(\log n+\frac{1}{2})} e^{-\log n}\right)$$

$$= \Theta\left((\log n)^{(\log n+\frac{1}{2})} n^{-\log e}\right) = \Theta\left(\log n^{\log n} \frac{\log n^{\frac{1}{2}}}{n^{\log e}}\right)$$

Some useful functions:

① $\lg^k n = (\lg n)^k$

② $\lg\lg n = \lg(\lg n)$

③ $n^n \in \Omega(n!)$ $(n^n > n!)$

④ $n^n > n! > a^n$

X 1. 2^{2^n}

X 2. $\log(n!) = \log 1 + \log 2 + \cdots + \log n \le n \cdot \log n$

$\log(n!) \in \Theta(n \log n)$

and $\log 1 + \cdots \log\frac{n}{2} + \cdots + \log n \ge \log\frac{n}{2} + \cdots \log n$

$\ge \log\frac{n}{2} + \log\frac{n}{2} + \cdots + \log\frac{n}{2}$

$= \frac{n}{2}\log\frac{n}{2}$ $(n \log n)$

X 3. $2^{2^{n+1}}$

X 4. $n^{\frac{1}{\log n}} = 2^{\log n \cdot \frac{1}{\log n}} = 2.$

X 5. $2^{\sqrt{n}} = 2^{n^{\frac{1}{2}}} =$

X 6. $\log n$

X 7. $2^{2\log n} = n^{2\log 2} = n^2$

X 8. n^2

X 9. $n^{\frac{1}{2}}$

X 10. $n \log n$

X 11. 2^n

X 12. $(\log n)! > (\log n)^{(\log n+\frac{1}{2})} \cdot \frac{1}{n} = (\log n)^{\log n} \cdot \frac{(\log n)^{\frac{1}{2}}}{2}$

X 13. $n!$

X 14. $(\sqrt{2})^{\log n} = (2^{\frac{1}{2}})^{\log n} = (2^{\log n})^{\frac{1}{2}} = n^{\frac{1}{2}}$

X 15. 1

Ans: $2^{2^{n+1}} > 2^{2^n} > n! > 2^n > 2^{\sqrt{n}} > (\log n)! > n^2 = n^{\frac{1}{\log n}} >$

$n \log n = \log(n!) > n^{\frac{1}{2}} = (\sqrt{2})^{\log n} > \log n > n^{\frac{1}{\log n}} = 1$

6. 參考 "https://walkccc.me/CLRS/Chap04/4.1/"

```
iterativeFindMaximumSubarray(A)
    n = A.length
    bestSum = -∞
    currentSum = -∞
    for j = 1 to n
        currentEnd = j
        if currentSum > 0
            currentSum = currentSum + A[j]
        else
            currentBegin = j
            currentSum = A[j]
        if currentSum > bestSum
            bestSum = currentSum
            bestBegin = currentBegin
            bestEnd = currentEnd
    return (bestBegin, bestEnd, bestSum)
```

想法: 如果現在的A[1,...,j+1]之和為負, 重新從j+1開始計maximum subarray, 因為新的subarray不可能加上一個負值後變得更大

Time complexity:

recursive: $\Theta(n \log n)$

Iterative: $\Theta(n)$

7. (a) 參考 "https://ita.skanev.com/04/02/04.html"

Suppose that we have $C = A \times B$, where A, B are two $n \times n$ matrices with $n = 3^m$.

By Strassen's method, we can divide A, B, C into $\frac{n}{3} \times \frac{n}{3}$ matrix respectively. Then, we can compute each $\frac{n}{3} \times \frac{n}{3}$ submatrix in C by recursively compute k matrix products $P_1, \dots, P_k$.

Thus, the recursive running time $T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ k \cdot T(\frac{n}{3}) + \Theta(n^2), & \text{if } n > 1. \end{cases}$

By master theorem, if $n^2 = O(n^{\log_3 k - \varepsilon})$, $T(n) = \Theta(n^{\log_3 k})$

Assuming $k \ge 10$ and $\varepsilon = 0.01$, we get $T(n) = \Theta(n^{\log_3 k})$

Thus, $T(n) = o(n^{\lg 7}) \Leftrightarrow n^{\log_3 k} < n^{\lg 7}$

$\Leftrightarrow \log_3 k < \lg 7$

$\Leftrightarrow \log k < \lg 7 \cdot \log 3$

$\Leftrightarrow k < e^{\lg 7 \cdot \log 3} = 21.849\dots$

$\Rightarrow$ the largest value of k is 21 (which also satisfies our assumption) #

7. (b)

If $k = 21$, the time complexity of this algorithm is $T(n) = \Theta(n^{\log_3 21})$ #.

8. b.

$T(n) = 3T(\frac{n}{3}) + \frac{n}{\lg n}$

$= 3[3 \cdot T(\frac{n}{9}) + \frac{\frac{n}{3}}{\lg \frac{n}{3}}] + \frac{n}{\lg n} = 9T(\frac{n}{9}) + \frac{n}{\lg n - \lg 3} + \frac{n}{\lg n}$

$= 9[3T(\frac{n}{27}) + \frac{\frac{n}{9}}{\lg \frac{n}{9}}] + \frac{n}{\lg n - \lg 3} + \frac{n}{\lg n}$

$= 27T(\frac{n}{27}) + \frac{n}{\lg n - \lg 9} + \frac{n}{\lg n - \lg 3} + \frac{n}{\lg n}$

$= 3^{\log_3 n} T(1) + n \cdot \sum_{k=0}^{(\log_3 n)-1} \frac{1}{\lg(\frac{n}{3^k})}$

$= \Theta(n) + \Theta(n \log\log_3 n)$

$= \Theta(n \lg\lg n)$ #.

$* \sum_{k=0}^{\log_3 n - 1} \frac{1}{\lg(\frac{n}{3^k})} = \frac{1}{\lg n} + \frac{1}{\lg \frac{n}{3}} + \frac{1}{\lg \frac{n}{9}} + \dots + \frac{1}{\lg 3}$

also, $\lg n = \frac{\log_3 n}{\log_3 2} \Rightarrow \frac{1}{\lg n} = \frac{\log_3 2}{\log_3 n}$

$= \log_3 2 \left(\frac{1}{\log_3 n} + \frac{1}{\log_3 n - 1} + \frac{1}{\log_3 n - 2} + \dots + \frac{1}{\log_3 9} + \frac{1}{\log_3 3} \right)$

$= \log_3 2 \left(\sum_{i=1}^{\log_3 n} \frac{1}{i} \right)$.

$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$

$= \Theta(\log\log_3 n) = \Theta(\lg\lg n)$

$\log_3 n = \frac{\lg n}{\lg 3} = \Theta(\lg n)$.

d.

$T(n) = 3T(\frac{n}{3} - 2) + \frac{n}{2}$

when n is large, $T(n) = 3T(\frac{n}{3}) + \frac{n}{2}$

$f(n) = \frac{n}{2} = \Theta(n^{\log_3 3}) = \Theta(n)$.

By master theorem, $T(n) = \Theta(n \lg n)$ #.

J.

$T(n) = \sqrt{n}\,T(\sqrt{n}) + n$.

參考: stack overflow,
"What would cause an algorithm to have $O(\log\log n)$ complexity?"

$n \rightarrow n \times 1$

$n^{\frac{1}{2}} \rightarrow n^{\frac{1}{2}} \times n^{\frac{1}{2}} = ''$ → 個數

$n^{\frac{1}{4}} \rightarrow n^{\frac{1}{4}} \times (n^{\frac{1}{4}} \times n^{\frac{1}{2}}) = n$

$\vdots$

$T(1) \rightarrow n$.

height = $\lg\lg n$.

*Height explanation: Since each time you take the square root of n, you "halve the exponent in this equation". Therefore, there can be only $O(\log k)$ square roots applied before k drops to one or lower. Assume $n = 2^k$, $k = \log n$, $O(\log k)$ $= O(\log\log n)$

(k最多遞減 $\log k$ 次, $n=2^k$ 最多遞減 $\log k = \log\log n$ 次)

$\Rightarrow T(n) = \Theta(n\lg\lg n)$ #.

9.
參考 "algorithm.cs.nthu.edu.tw/~course/Extra-Info/Divide and Conquer - Supplement.pdf"

(a) Divide step:

Assume n is the power of 2

Let $A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$.

$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$

$A(x) = A_0(x) + A_1(x)\, x^{\frac{n}{2}}$

$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$

$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$

$B(x) = B_0(x) + B_1(x)\, x^{\frac{n}{2}}$

$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)\, x^{\frac{n}{2}} + A_1(x)B_0(x)\, x^{\frac{n}{2}} + A_1(x)B_1(x)\, x^n$

(The original problem of size n is divided into 4 problems of size $\frac{n}{2}$)

Conquer step:

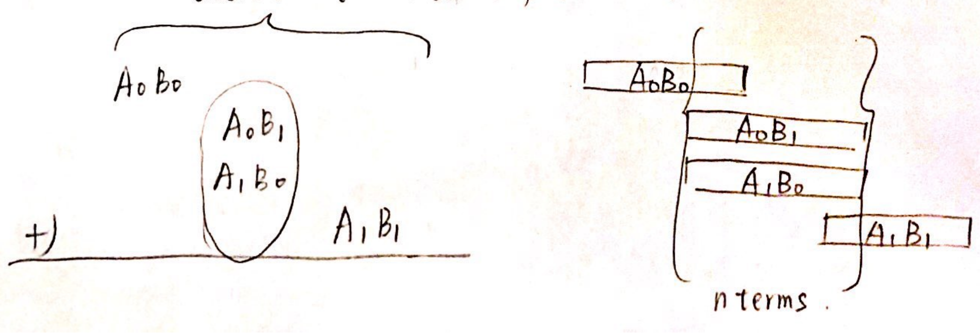
compute $A_0(x)B_0(x)$, $A_0(x)B_1(x)$, $A_1(x)B_0(x)$, $A_1(x)B_1(x)$ by recursively calling the algorithm 4 times.

Combine step:

We do the following calculation

$A_0(x)B_0(x) + A_0(x)B_1(x)\, x^{\frac{n}{2}} + A_1(x)B_0(x)\, x^{\frac{n}{2}} + A_1(x)B_1(x)\, x^n$.

there are $(2n+1)$ terms to be added $\Rightarrow \Theta(n)$ time



Pseudocode:

PolyMultiply(A(x), B(x))

$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$

$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$

$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$

$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$

$U(x) =$ PolyMultiply$(A_0(x), B_0(x))$

$V(x) =$ PolyMultiply$(A_0(x), B_1(x))$

$W(x) =$ PolyMultiply$(A_1(x), B_0(x))$

$Z(x) =$ PolyMultiply$(A_1(x), B_1(x))$

return $(U(x) + [V(x) + W(x)] x^{\frac{n}{2}} + Z(x) x^n)$.

(b)

$$T(n) = \begin{cases} 4T(\frac{n}{2}) + (2n+1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

(ignore cost of divide here, since it's miscelloneous compared to $(2n+1)$).

$f(n) = 2n+1 = O(n^{\log_2 4 - \epsilon}) = O(n^{2-\epsilon})$

choose $\epsilon = 0.1$

By master theorem, $T(n) = \Theta(n^2)$ #.

(c)

In fact, we only need 3 terms in the conquer steps.

$(A_0B_0,\ A_0B_1 + A_1B_0,\ A_1 + B_1)$

This can be obtained by using 3 multiplications

$Y = (A_0 + A_1)(B_0 + B_1)$

$U = A_0 B_0$

$Z = A_1 B_1$.

we can get $A_0B_1 + A_1B_0 = Y - U - Z$

(d)

PolyMultiplyBetter(A(x), B(x))

$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$

$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$

$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$

$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$

$Y(x) =$ PolyMultiplyBetter$(A_0(x) + A_1(x), B_0(x) + B_1(x))$

$U(x) =$ PolyMultiplyBetter$(A_0(x), B_0(x))$

$Z(x) =$ PolyMultiplyBetter$(A_1(x), B_1(x))$

return $(U(x) + [Y(x) - U(x) - Z(x)] x^{\frac{n}{2}} + Z(x) X^n)$

(e)

$$T(n) = \begin{cases} 3T(\frac{n}{2}) + (2n+1), & \text{if } n > 1 \\ 1, & \text{if } n = 1. \end{cases}$$

$f(n) = 2n+1 = O(n^{\log_2 3 - \epsilon})$

choose $\epsilon = 0.01$ $\quad (\frac{\log 3}{\log 2} = 1.585)$

By master theorem, $T(n) = \Theta(n^{\log_2 3})$

$= \Theta(n^{1.585})$ #