**1.2**

b05902124

http://stackoverflow.com/questions/4999448

**1.3** b05902124

**2.1**

b05902100

系籃學姊

**2.2(b)** b05902004

**2.3** b05902108

**3.2** b05902108

**3.3** b05902108

**3.4** b05902108

**3.5** b05902108 / b05902120

**Problem4**

系籃學姊

b05902004 / b05902041 / b05902100 / b05902120 / b05902124

TA

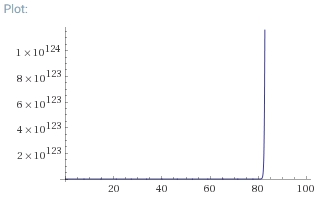
https://openhome.cc/Gossip/AlgorithmGossip/InFixPostfix.htm

**Problem5**

系籃學姊

b05902041 / b05902128

**1.1**



g(1)

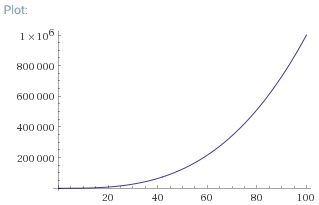
g(1) = n!

g(2) = 2n

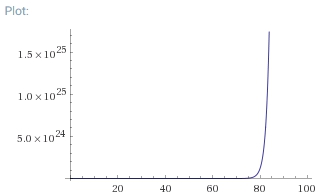
g(3) = n3-n

g(4) = elogn = n

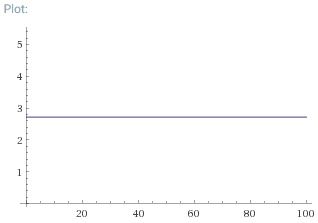
g(5) = n1/logn = e



g(3)

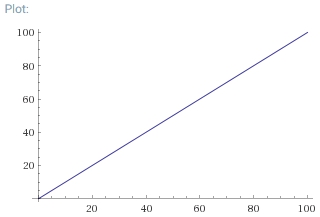


g(2)



g(5)

**1.2**



g(4)



🡪Both of denominator and nominator have n numbers. When n goes to infinity, f(n) goes to infinity.

🡪n! > c(2n)

🡪n! =ω(2n)



🡪Both of denominator and nominator have n numbers. When n goes to infinity, f(n) = 0.

🡪n! < c(nn)

🡪n! =ο(2n)

**1.3**

**(a)**

g(n) =Ω(f(n))

🡪g(n) >= c\*f(n)

🡪c' = 1/c，c'g(n) >= f(n)

🡪f(n) = O(g(n))

f(n) = O(g(n))

🡪f(n) <= c\*g(n)

🡪(1/c)\*f(n) <= g(n)

🡪c' = 1/c，g(n) >= c' \* f(n)

🡪g(n) =Ω(f(n))

**(b)**

f(n) =Ω(g(n))，f(n) = O(g(n))

🡪f(n) >= c1\*g(n)，f(n) <= c2\*g(n)

🡪c1\*g(n) <= f(n) <= c2\*g(n)

🡪f(n) = θ(g(n))

f(n) =θ(g(n))

🡪c1\*g(n) <= f(n) <= c2\*g(n)

🡪 f(n) >= c1\*g(n)、f(n) <= c2\*g(n)

🡪f(n) =Ω(g(n))，f(n) = O(g(n))

**(c)**

f(n)\*g(n) = O(g(n)2)

🡪f(n)\*g(n) <= c\*g(n)2

🡪f(n) <= c\*g(n)

🡪f(n) = O(g(n))

f(n) = O(g(n))

🡪f(n) <= c\*g(n)

🡪f(n)\*g(n) <= c\*g(n)2

🡪f(n)\*g(n) = O(g(n)2)

**(d)**

f(n)2 = O(g(n)2)

🡪f(n)2 <= c\*g(n)2

🡪f(n) <= c1/2\*g(n)

🡪c' = c1/2，f(n) <= c'\*g(n)

🡪f(n) = O(g(n))

f(n) = O(g(n))

🡪f(n) <= c\*g(n)

🡪f(n)2 <= c2\*g(n)2

🡪c' = c2，f(n)2 <= c'\*g(n)2

🡪f(n)2 = O(g(n)2)

**2.1**

Binary\_Search(A, N, k)

sort(A) 的 time complexity是O(NlogN)

因為：

k為while執行的次數

N/2k <= 1 🡪 N <= 2k 🡪 log2N <= k

N/2k-1 > 1 🡪 N > 2k/2 🡪 log2N > k-1 🡪 log2N+1 > k

log2N <= k < log2N+1

O(log2N)

所以：

for loop 的 time complexity是O(NlogN)

c1 : sort(A)

c2 : for loop

c3: return

T(N) = c1\*NlogN+c2\*NlogN+c3

Time complexity 🡪 O(NlogN)

Count\_Search(A, N, K, k)

因為：

一個for loop的time complexity是O(N)

c1 : malloc

c2 : for loop

c3 : return

T(N) = c1+c2\*N+c3

Time complexity 🡪 O(N)

因為：

最大的k可能是K+(K – 1)

所以：

最糟的情況是開一個大小為2K的動態陣列

Space complexity 🡪 O(K)

I think that Count\_Search is the better.

By time complexity, we know O(NlogN)>O(N) when N goes to a large number. Therefore, it will take less time than Binary\_Search when N goes to a large number.

**2.2**

**(a)**

for i = 0 to (A.length - 1)

for j = 0 to (A.length - 1)

if A[i] + A[j] == k

M++;

1

2

3

4

**(b)**

merge(arr, reg, 0, N)

M = 0;

for i = 0 to N - 1

search = k - A[i];

left = 0;

right = N - 1;

while left <= right

mid = (left + right)/2;

if A[mid] == search

M++;

else if A[mid] > search

right = mid - 1;

1

2

3

4

5

6

7

8

9

10

11

12

13

else if A[mid] < search

left = mid + 1;

void merge(int arr[], int reg[], int start, int end)

if start >= end

return;

len = end - start + 1;

mid = (start + end)/2;

start1 = start;

end1 = mid;

start2 = mid + 1;

end2 = end;

merge(arr, reg, start1, end1);

merge(arr, reg, start2, end2);

k = start;

while start1 <= end1 and start2 <= end2

if arr[start1] < arr[start2]

reg[k] = arr[start1];

start1++;

else

reg[k] = arr[start2];

start2++;

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

k++;

while start1 <= end1

reg[k] = arr[start1];

k++;

start1++;

while start2 <= end2

reg[k] = arr[start2];

k++;

start2++;

for k = start to end

arr[k] = reg[k];

return;

35

36

37

38

39

40

41

42

43

44

45

**2.3**

1

for i = 0 to N - 3

now\_k = k - A[i];

Count\_Search (A, N, K, now\_k);

Count\_Search (A, N, K, k) {

B = malloc (K);

for i = 0 to N - 1

B[A[i]] = true

for i = 0 to N - 1

2

3

4

5

6

7

8

9

if B[k - A[i]]

return true;

return false;

}

10

11

12

**3.1**

push 1

push 2

push 3

pop

pop

push 4

pop

pop

push 5

pop

**3.2**

for i = 1 to n

if A[i] != i

return false;

return true;

1

2

3

4

Time complexity 🡪 O(n)

**3.3**

依題意照1, 2, 3, 4.....順序push，若要pop某一個數字必定已經push所有比她小的數字，且遵守stack last in first out 的規則，已push的數字由小到大，則pop出來的數字必是由大到小，所以判斷是否為stack-valid，只要以一數為基準，被pop出來所有比她小的數字必須由大到小。

for i = 0 to n – 2

key = A[i];

for j = i + 1 to n - 1

if A[j] < A[i]

key--;

if A[i] != key

return false;

return true;

1

2

3

4

5

6

7

8

Time complexity 🡪 O(n2)

**3.4**

因為：

queue-valid的唯一可能性是[1, 2, 3, 4, 5]，pseudo code跟3.2一樣

所以：

只要設一個變數i從1開始跑，一次判斷一個數字再i++，判斷順序是不是[1, 2, 3, 4, 5]

Space complexity 🡪 O(1)

Time complexity 🡪 O(n)

**3.5**

因為：

設三個變數，i以哪個數字為基準、key--數字依序遞減、j紀錄現在的位置，開始往後跑，pseudo code跟3.3一樣

所以：

再三個space 🡪 O(1)additional space