DSA HW1

Problem 0:

Problem 1: all by myself!
Problem 2: all by myself!
Problem 3: all by myself!
Problem 4:李沛宸(B10902032)之前教的作法、許睿尹(B10902122)
Problem 5:李沛宸(B10902032)、許睿尹(B10902122)

Problem 1:

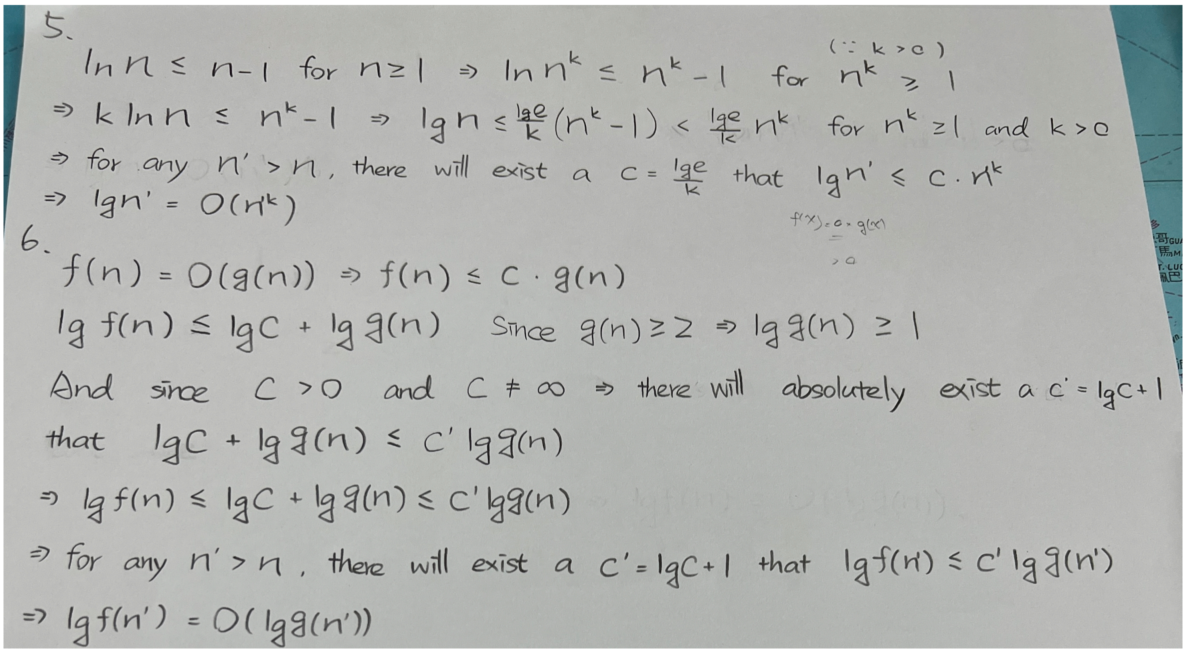
1. Since i grows with base 2, sum would be (2^k)-1 when the loop runs k times. The iteration count of the while loop is log(n), and the time complexity would be Θ(log(n)).

2. We use f(n) as how many times FUNC-B(n) runs, then f(n) = 2f(n-1) + 1. And since f(0) = 1, we can know f(n) = 2^(n+1) - 1, the time complexity would be Θ(2^n).

3. The first while loop will run log(n) times, and the second while loop will run n + n/2 + n/4 + n/8 + … = 2n times. Hence the time complexity would be Θ(n).

4. Since f(n) = O(g(n)), we know that f(n) <= c*g(n) for c is a positive number, then we can know g(n) >= (1/c)f(n).
   Hence we can know f(n) * g(n) >= 1/c (f(n)^2).
   For any n' > 0, there will exist a c' = 1/c that f(n) * g(n) >= c' * (f(n)^2), so f(n) * g(n) = Ω(f(n)^2).

5.
6.



5.

$\ln n \le n-1$ for $n \ge 1$ $\Rightarrow \ln n^k \le n^k - 1$ for $n^k \ge 1$ $(\because k > 0)$

$\Rightarrow k \ln n \le n^k - 1 \Rightarrow \lg n \le \frac{\lg e}{k}(n^k - 1) < \frac{\lg e}{k} n^k$ for $n^k \ge 1$ and $k > 0$

$\Rightarrow$ for any $n' > n$, there will exist a $c = \frac{\lg e}{k}$ that $\lg n' \le c \cdot n'^k$

$\Rightarrow \lg n' = O(n'^k)$

$f(x) = o \cdot g(x)$

> 0

6.

$f(n) = O(g(n)) \Rightarrow f(n) \le c \cdot g(n)$

$\lg f(n) \le \lg c + \lg g(n)$ Since $g(n) \ge 2 \Rightarrow \lg g(n) \ge 1$

And since $c > 0$ and $c \ne \infty \Rightarrow$ there will absolutely exist a $c' = \lg c + 1$

that $\lg c + \lg g(n) \le c' \lg g(n)$

$\Rightarrow \lg f(n) \le \lg c + \lg g(n) \le c' \lg g(n)$

$\Rightarrow$ for any $n' > n$, there will exist a $c' = \lg c + 1$ that $\lg f(n') \le c' \lg g(n')$

$\Rightarrow \lg f(n') = O(\lg g(n'))$

Problem 2:

1. 1, 2, +, 3, 4, 5, *, +, *, 9, 3, /, 7, 5, *, +, +

I call a mathematical expression like 3 + 4 * 5 - 6 an infix expression below.
Human Algorithm:
First, we should add parenthesis to the infix expression and make every part of it looks like this: (number, operator, number).
Then the infix expression will be (((1+2)*(3+(4*5)))+((9/3)+(7*5))).
Now we change the order from (number1, operator, number2) to (number1, number2, operator), and remove all the parenthesis.
Then we will get 1, 2, +, 3, 4, 5, *, +, *, 9, 3, /, 7, 5, *, +, + which is the answer.

2. (1 + 2) * (5 - 3) * 6 / 5 = 36/5

I call a mathematical expression like 3 4 + 5 * an postfix expression below.
Human Algorithm:
Once we encounter an operator, we
(1) take out the previous two numbers, caculate them with the operator,
(2) and put the answer back to the postfix expression.
Ex. f(x) = number1, number2, number3, operator …
(1) We take out number2 and number3 and caculate them
-> number2 operator number3 = number4
(2) put it back -> f(x) = number1, number4 …

3.

	a[0]	a[1]	a[2]	a[3]
Enqueue 1	1	NIL	NIL	NIL
Enqueue 5	1	5	NIL	NIL
Enqueue 3	1	5	3	NIL
Dequeue	NIL	5	3	NIL
Enqueue 4	NIL	5	3	4
Enqueue 6	6	5	3	4
Dequeue	6	NIL	3	4

4.
At least 10 rounds.
Since genius-mahjong-saki always puts the new tile in the rightmost of her hand and discards the leftmost tile, we can know it is a kind of queue.
Before:4p,2p,9p,8p,7p,6p,5p,4p,3p,2p,1p,1p,1p
After :1p,1p,1p,2p,3p,4p,5p,6p,7p,8p,9p,9p,9p
By comparison, we can know that genius-mahjong-saki needs at least 10 rounds to win chuuren poutou(discard the previous ten tiles and get 2p, 3p, 4p, 5p, 6p, 7p, 8p, 9p, 9p, 9p in order).

5.
We can use a stack to solve this problem. If input == top, we pop the top, else we push the top. And after we scanned all the inputs, if there's still anything in the array, there will have at least one intersect.
Pseudo Code:

```
a[ ] stores the input, index starts from 0
b[ ] is used as stack, b[0] = 0
index = 0
——————————————————-
For k from 0 to 2n-1:
    if a[k] == b[index]
        b[index] = 0
        index -= 1
    else
        index += 1
        b[index] = a[k]
    end
——————————————————-
if index == 0
    no intersection
else
    have at least one intersection
```

6.
Since the first part and the third part both run only one time(O(1)), and the second part runs 2n times(from 0 to 2n-1), the total time complexity is O(n).

Problem 3:

1.
Pseudo Code:

```
ptr1 records the last node we have already traversed
ptr2 records the middle one we have already traversed
ptr1 = head, ptr2 = head
flip = 1

while ptr1 -> next != NULL

    ptr1 = ptr1 -> next
    if flip == -1
        ptr2 = ptr2 -> next
    flip = -flip
    end

return ptr2
```

————————————————————————

Since we only traverse one time, the time complexity is O(n).
And since we only have countable extra variables, the extra space complexity is O(1).

2.
Pseudo Code:

```
a[ ] stores the given numbers, index starts from 1
i = 1

While i <= n:
    if a[i] <= n && a[a[i]] != a[i] && a[i] != i
        swap(a[i], a[a[i]])
    else
        i += 1
    end

For i from 1 to n:
    if a[i] != i
        return i
    end

return n+1
```

————————————————————————

Every time we run the while loop, we can secure that one more element in the array is at the right position. So to make every element in the right

position, we need to run the while loop n times. Also we'll run the for loop for at most times, so the time complexity is O(n).
And there's only countable extra variables, so the extra space complexity is O(1).

3.
Pseudo Code:

```
a[ ] stores the weight of the stones, index starts from 1
sum = 0, Wsum = 0
index = 0

For i from 1 to n:
        sum += a[i] * i
        Wsum += a[i]
        end

init = sum
while sum*2 > init
        sum -= Wsum
        index += 1

if sum*2 == init
        the index is the smallest location to place pivot.
else
        no answer
```

————————————————————————

Since we run the for loop n times, and run the while loop for at most n times, the time complexity is O(n).
And there's only countable extra variables, so the extra space complexity is O(1).