# DSA-Hw1

## Problem0 Proper References

- Problem1
  - 1-1 ~ 1-5 (all by myself)
  - 1-6 (https://reurl.cc/WkYzxL)
- Problem2
  - 2-1 (leacture handout)
- Problem3
  - 3-2 (https://reurl.cc/yQbGAO)

## Problem1 - Analysis Tools

1.

在第k輪迴圈時，$sum = 1 + 2 + ... 2^{k-1} = 2^k - 1$
$\exists k,\ 2^{k-1} - 1 < n \leq 2^k - 1$
$\Rightarrow complexity$ is $\Theta(log(n))$

2.

Total interations count = $1 + 3 + (3-1)\cdot 3 + ((3-1)\cdot 3 - 2)\cdot 3 + ...$
$= 1 + 3 + 2\cdot 3 + 4\cdot 3 + 8\cdot 3 + ...$
$= 1 + 2^0\cdot 3 + 2^1\cdot 3 + ... + 2^{n-1}\cdot 3$
$= 1 + 3\cdot(2^n - 1)$
$\Rightarrow complexity$ is $\Theta(2^n)$

3.

Total interations count = $n + n/2 + n/2^2 + ... n/2^{log(n)}$
$= n(1 + \frac{1}{2} + (\frac{1}{2})^2) + ... + (\frac{1}{2})^{log(n)})$
$= n(2 * (1 - (\frac{1}{2})^{log(n)})) \Rightarrow complexity$ is $\Theta(n)$

4.

$\exists (c, n_0)$ that $(c, n_0)$ is positive, that $f(n) \leq c\cdot g(n)$ for all $n \geq n_0$
$\Rightarrow f(n)^2 \leq c\cdot f(n)\cdot g(n)$
$\Rightarrow \frac{1}{c}\cdot f(n)^2 \leq f(n)\cdot g(n)$
Let $\frac{1}{c} = c'$ , We get $c'\cdot f(n)^2 \leq f(n)\cdot g(n)$ for all $n \geq n_0$
$\Rightarrow f(n)\cdot g(n) = \Omega((f(n)^2))$

5.

$ln(n) \leq n - 1 \leq n$ , for $n >= 1$
$\Rightarrow lg(n) = lg(e)\cdot ln(n) \leq lg(e)\cdot n$

$\Rightarrow lg(n) \leq \frac{lg(e)}{n^{k-1}} \cdot n^k$, for $k > 0$
Let $\frac{lg(e)}{n^{k-1}} = c, n_0 = 1$
$\Rightarrow$ satisfy that $\exists$ positive $(c, n_0)$ that $lg(n) \leq c \cdot n^k$, for $n \geq n_0$

6.

If $f(n) = O(g(n))$, $\Rightarrow \exists (c, n_0)$ that $(c, n)$ is positive, that $f(n) \leq c \cdot g(n)$ and $n \geq n_0$
If there exists some $n_2$ such that $g(n) \geq 2$ for all $n \geq n_2 \Rightarrow lg(g(n)) \geq 1$

$\Rightarrow lg(f(n)) \leq lg(c \cdot g(n)) = lg(c) + lg(g(n))$

For all $n \geq n_2$ :
$\because lg(g(n)) \geq 1$
$\therefore lg(c) + lg(c) \leq (1 + lg(c)) \cdot lg(g(n))$
Let $c' = (1 + lg(c))$
$\Rightarrow$ satisfy that $\exists$ positive $(c', n_0)$ that $lg(f(n)) \leq c' \cdot lg(g(n))$, for $n \geq n_2$
$\therefore lg(f(n)) = O(lg(g(n)))$

## Problem 2 - Stack / Queue

1.

solution:
1 2 + 3 4 5 x + x 9 3 / 7 5 x + +

human algorithm:
Scan the infix expression from left to right, and prepare a stack to store operator

1. if token is a number we output the token.
2. if token is a operator
    1. if token is ')', we keep output POP(stack) until POP(stack) is '('
    2. else check stack, while stack is not empty and top operator in stack is not ')' and has higher/same precedence, we output POP(stack)
3. while stack is not empty, we output POP(stack)

pseudo code:

```
S = empty stack
for each token in input
    if token is a number
        output token
    else if token is an operator
        if token is ')'
            topToken = POP(S)
            while not IS-EMPTY(S) and topToken is not '('
                output topToken
                topToken = POP(S)
        else
```

```
            while not IS-EMPTY(S) and PEEP(S) is higher/same precedence
and PEEP(S) is not '('
                output POP(S)
            PUSH(S, token)

while not IS-EMPTY(S)
    output POP(S)
```

2.

solution:
(((1+2) x (5-3)) x 6) / 5

outcome:
36/5

human algorithm:
Scan the postfix expression from left to right, and prepare a stack to store number

1. if the token is number we push token into stack
2. if the token is operator, we let b = POP(stack), a = POP(stack), than push (a operator b) into stack
3. in the end of postfix expression, we output POP(stack)

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.

solution:
10 rounds, if she is lucky, she needs to discards 10 from left and get new in seqence: 2p, 3p, 4p, 5p, 6p, 7p, 8p, 9p, 9p, 9p

5.

pseudo code:

```
labels is a length-n array
S is a empty stack

checkCordsIntersect(labels, S)
{
    for label in labels
        if stack.length > 0 and label equal to S.peep()
            S.pop()
        else
            S.push(label)
    if stack.length > 0
        return false
    else if stack.length == 0
        return true
}
```

1. Prepare a empty stack to store label,
2. if stack is empty or label is not equal to top label in stack, we push it into stack.
3. if stack is not empty and label is equal to top label in stack, we pop the top label from stack.
4. In the end if there is label in the stack, it means the red cords might entangle with each other in the circle.

6.

Because the time complexity of stack.pop() and stack.peep() and stack.push() are O(1), the overall time complexity is O(n) + O(1) = O(n)

## Problem 3 - Array / Linked Lists

1.

pseudo code:

```
quick_ptr = head
slow_ptr = head

while quick_ptr->next != NIL
    slow_ptr = slow_ptr->next
    quick_ptr = quick_ptr->next
    if quick_ptr->next != NUL
        quick_ptr = quick_ptr->next

return slow_ptr
```

1. let quick_ptr and slow_ptr start from head
2. slow_ptr move 1 step one time, quick_ptr move 2 steps one time until reach the tail node

3. when quick_ptr reaches the tail node, slow_ptr is at the middle node.

time complexity and extra-space complexity:
time complexity = O(1) + O(1) + O(n) = O(n)
extra-space complexity = O(1) + O(1) = O(1)

2.

pseudo code:

```
numbers is the unsigned integer array

for i from 1 to n
    while numbers[i] != numbers[numbers[i]] and numbers[i] <= n
        temp = number[i]
        number[i] = numbers[numbers[i]]
        numbers[numbers[i]] = temp

for i from 1 to n
    if numbers[i] != i
        return numbers[i]

return n + 1
```

1. for i = 1 to i = length of array, we want to put the array[i] to index j, which j = array[i]
2. if array[i] < length of array and array[i] is not at index j, which j = array[i], we swap(array[i], array[array[i]])
3. in the end all the array[i] should be at index j, which j = array[i]
4. then the first index i != array[i] is the first missing number
5. if all number i = arrray[i] for i in range 1 to length of array, the first missing number is length of array + 1

time complexity and extra-space complexity:
For each swap, we can put one element to its right position. So there is at most n swap. The time complexity of swap two element in array is O(1).
time complexity = O(n) + O(n) = O(n)
extra-space complexity = O(1)

3.

```
a = 0
b = 0

for i = 1 to i = n
    a = a + array[i] * i
    b = b + array[i]

if a % b == 0 and a / b > 0
```

```
        return a/b
    else
        return NIL
```

suppose the balance position is x, for element $a_i$ in array, we can get the equation that
$a_1 \cdot (1-x) + a_2 \cdot (2-x) + \ldots + a_n \cdot (k-x) = 0$
$\Rightarrow \sum_{i=1}^{n} a_i \cdot i - x \cdot (\sum_{i=1}^{n} a_i) = 0$
$\Rightarrow x = \frac{\sum_{i=1}^{n} a_i \cdot i}{\sum_{i=1}^{n} a_i}$
if x is not positive integer return NIL, means there is no pivot under one of the stones.

`time complexity and extra-space complexity:`

time complexity = O(1) + O(1) + O(n) = O(n) extra-space complexity = O(1) + O(1) = O(1)