DSA HW #1

B09902063 資工一 董瑋

Problem 1:

1. Ans : $\Theta(\sqrt{n})$

   Let $sum = 1+2+3+...+k$
   k is the times of while loop and $k > 0$.
   $\therefore sum = \frac{k(k+1)}{2} = \frac{(k^2+k)}{2}$
   And we can know that $\frac{k(k-1)}{2} < n \leq \frac{k(k+1)}{2}$
   By simple calculation we can know that: $\frac{1+\sqrt{1+8n}}{2} > k \geq \frac{-1+\sqrt{1+8n}}{2}$
   Let $f_1(x) = O_1(g_1(x))$ $f_2(x) = O_2(g_2(x))$ $f_3(x) = O_3(g_3(x))$
   Let $x_0 = 1$, $g(n) = \sqrt{n}$, $f(n) = k$
   When $n > 0 \lim_{n\to\infty} \frac{1+\sqrt{1+8n}}{2\sqrt{n}} > \frac{k}{\sqrt{n}} \geq \frac{-1+\sqrt{1+8n}}{2\sqrt{n}}$
   $lim_{n\to\infty}\sqrt{2} > \frac{k}{\sqrt{n}} \geq \sqrt{2}$
   Thus $c = \sqrt{2}$, $x_0 = 0$
   And we can know that $g_1(x) = g_2(x) = g_3(x) = \sqrt{n}$
   $f_1(x) = f_2(x) = f_3(x)$
   Thus the ans is $\Theta(\sqrt{n})$

2. Ans:$\Theta(\log(\log n))$

   Suppose it runs k times to break the while loop for all k greater than 0
   m will equal to 2^{2^k} in the end
   $2^{2^k} \geq n$ $\therefore k \geq \log_2(\log_2(n))$

3. Ans: $\Theta(4^n)$
   It runs $4^{n-87506055} * (3^{8750655})$ times for all n greater then 87506055
   So $\lim_{n\to\infty} \frac{(4^{n-87506055} * (3^{8750655}))}{4^n} = (\frac{3}{4})^{87506055} = c$

4. If $Max(f(n), g(n)) = f(n)$ $\frac{f(n)+g(n)}{f(n)} = 1 + \frac{g(n)}{f(n)}$ $1 < 1 + \frac{g(n)}{f(n)} \leq 2$
   Similarly, when $Max(f(n), g(n)) = g(n)$ $1 < 1 + \frac{f(n)}{g(n)} \leq 2$
   We can know that for all $x > 0$ there is a $c_1 = 1$ and $c_2 = 2$ that
   $2 = c_2 \geq \frac{f(n)+g(n)}{Max(f(n),g(n))} > c_1 = 1$
   $\therefore f(n) + g(n) = \Theta(Max(f(n), g(n)))$

5. Let $\frac{f(n)}{i(n)} \leq c_1$ for all $n > 0$, and $\frac{g(n)}{j(n)} \leq c_2$ for all $n > 0$
   $\because f(n), g(n), i(n), j(n)$ are positive functions, $\frac{f(n)*g(n)}{i(n)*j(n)} \leq c_1 * c_2 = c_3$ for all$n > 0$

$\therefore f(n) * g(n) = O(i(n) * j(n))$

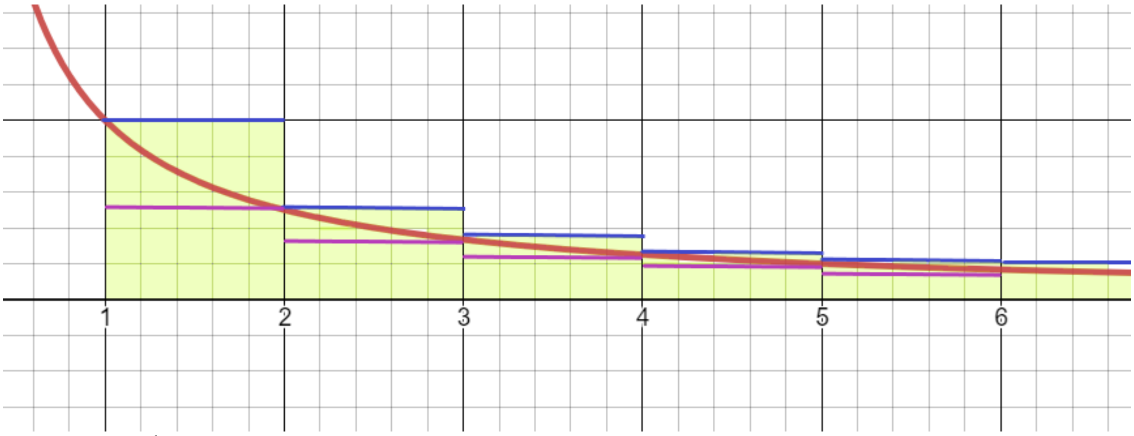
6. $False$

Counterexample: $f(n) = n^3 + n^2, g(n) = n^3, 2^{f(n)} = 2^{n^3+n^2}, 2^{g(n)} = 2^{n^3}$

For all $n > 1$, $\frac{f(n)}{g(n)} = 1 + \frac{1}{n} < 2$

$\frac{2^{f(n)}}{2^{g(n)}} = 2^{n^2}$ And $\lim_{n\to\infty} 2^{n^2}$ converges

7. ref : 和李長諺、石容居討論



$\lg n = \int_{p=1}^{n} \frac{1}{p} dp$

From area under $y = \frac{1}{x}$ that $\Sigma_{k=1}^{n}(\frac{1}{k})$ is greater than $\lg n$.

And $(\Sigma_{k=1}^{n}(\frac{1}{k+1}))$ is less than $\lg n$ $\quad E.g. (\Sigma_{k=1}^{n}(\frac{1}{k+1})) < \lg n < \Sigma_{k=1}^{n}(\frac{1}{k})$

For all $n > 1$, there is a $c_1 = 1$ that $c_1 = 1 < \frac{\Sigma_{k=1}^{n}(\frac{1}{k})}{\lg n}$ $\therefore \Sigma_{k=1}^{n}(\frac{1}{k}) = O((\lg n)$

$1 > \frac{\Sigma_{k=1}^{n}(\frac{1}{k+1})}{\lg n} = \frac{\Sigma_{k=1}^{n}(\frac{1}{k})-1+\frac{1}{n+1}}{\lg n}$ When n is greater than 2, $\frac{-1+\frac{1}{n+1}}{\lg n}$ is graeter than-1

$\therefore 2 > \frac{1-\frac{1}{n+1}}{\lg n} + 1 > \frac{\Sigma_{k=1}^{n}(\frac{1}{k})}{\lg n}$

For $n > 2$ there is a $c_2 = 2$ that $c_2 > \frac{\Sigma_{k=1}^{n}(\frac{1}{k})}{\lg n}$ $\therefore \Sigma_{k=1}^{n}(\frac{1}{k}) = \Omega(\lg n)$

Thus $\Sigma_{k=1}^{n}(\frac{1}{k}) = \Theta(\lg n)$

8. ref : 和李長諺、石容居討論

$lg(n!) = lg1 + lg2 + ... + lgn$

Obiviously $lg(n!) \geq nlgn$ For all n >1 $\therefore$ $lg(n!) = O(nlgn)$

And we know that $lg(n!) \geq \frac{n}{2}(lg\frac{n}{2}) = \frac{nlgn}{2} - \frac{n}{2}$,

From 併吞律 $lgn! = \Omega(nlgn)$

So $\lg n! = \Theta(n \lg n)$

9. Ans

$f(n) = n\lg n + n\lg(\frac{n}{2}) + n\lg(\frac{n}{4}) + ... + n\lg(\frac{n}{2^{\lceil \lg n\rceil - 1}}) + n$

$\quad = n\lg n * \lceil \lg n\rceil - n\lg 2(1 + 2 + ... + \lceil \lg n\rceil - 1) + n$

$\quad = n\lg n * \lceil \lg n\rceil - \frac{n(\lceil \lg n\rceil - 1)\lceil \lg n\rceil}{2} + n$

$\quad = n\lg n * \lceil \lg n\rceil - \frac{n\lceil \lg n\rceil\lceil \lg n\rceil - n\lceil lgn\rceil}{2} + n$

And $\quad lgn - 1 \leq \lceil lgn\rceil \leq lgn$

由併吞律可知$f(n) = \Theta(nlg^2 n)$-time

10. $f_k(n) = a_k n^2 + b_k n + c_k \ (a_k, b_k, c_k \in R \ and \ a_k! = 0)$
    $\Sigma_{k=1}^{n} f_k(n) = n^2 \Sigma_{k=1}^{n} a_k + n \Sigma_{k=1}^{n} b_k + \Sigma_{k=1}^{n} c_k$
    $a_k! = 0$ but $\Sigma_{k=1}^{n} a_k$ may be zero.
    For instance, $f_k(n) = n^2$ if $n \ mod \ 2 = 1$ else $f_k(n) = -n^2$

---

11. ref : 和李長諺、石容居討論

    令$m > n$,在前幾次迴圈中，(m, n)的變化將會是:
    $(m, n) \rightarrow (n, m \ mod \ n) \rightarrow (m \ mod \ n, n \ mod \ (m \ mod \ n))$
    若$\frac{m}{2} \geq n$ 則 $m \ mod \ n < \frac{m}{2}$
    若$\frac{m}{2} < n$ 又因為 $m > n$ 所以 $m \ mod \ n = m - n$
    因此 $m \ mod \ n < \frac{m}{2}$

    由此可知，每兩次GCD後m的值都至少會縮小一半 $\therefore$ time complexity = $O(lgn)$
    由併吞律可以再推得 time complexity = $O(lg(n + m))$

Problem 2

1. Pseudo code:

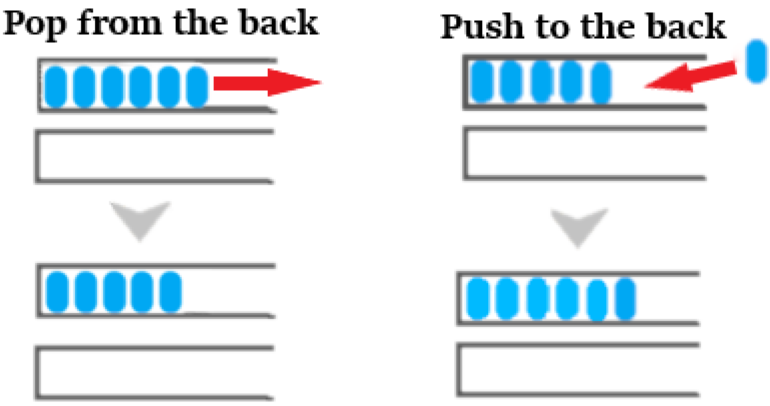
```
enqueue(front(source) , helper)
dequeue(source)

while size(helper)!=0:
    if size(helper)==0 :
         while size(source)>1:
             enqueue(front(source) , helper)
             dequeue(source)
    while size(helper)>1:
        enqueue(front(helper) , helper)
        dequeue(helper)
    if size(helper)>0:
        enqueue(front(helper) , source)
        dequeue(helper)
```

---

2. At first it runs n-1 times to move elements from $source$ to $helper$.
   For every loop, it runs n-1 , n-2 , n-3 ...... 1 times to move element from $helper's$ front to rear.
   Besides, it runs 1 time to enqueue element to source from $helper$ in every loop.
   So it runs $n(n-1) + n - 1 = n^2 - 1$ times in total.
   For all $n > 1$, $\frac{n^2-1}{n^2} = 1 + \frac{1}{n} - \frac{1}{n^2}$ must less than 2. Thus my algorithm runs in $O(n^2)$-time

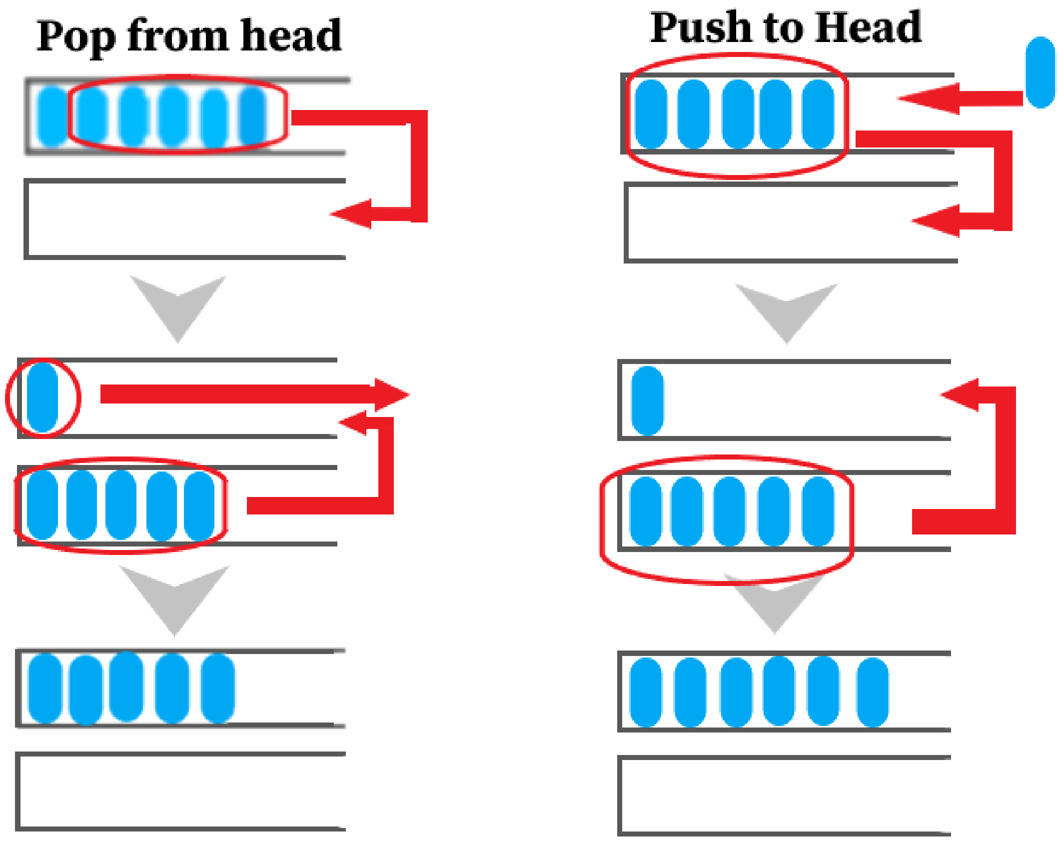
---

3. At first we put all the elements into stack1, our main stack.
   When we want to push/pop element to/from the back, deriectly do push/pop on main stack.



When we want to push element at the front of the dequeue, we move all the elements to the stack2, our helper stack, and then push the element wanted into main stack. Finally we put back all the elements to the main stack.

When we want to pop element from the front, we move all the elements except the first one in main stack to helper stack. And then we pop left element in main stack. Finally we move back all the element in helper stack.



---

4. Ans: $O(n)$

```
while size(stack1)>0:
    push(rear(stack1),stack2)
    pop(stack1)
push(x,stack1)
while size(stack2)>0:
    push(rear(stack2),stack1)
    pop(stack2)
```

We have to move n-1 elements to the other stack twice.

---

5. Ans: $O(1)$

```
push(x,stack1)
```

Just put it into the stack.

---

6. Ans: $O(n)$

```
while size(stack1)>1:
    push(rear(stack1),stack2)
    pop(stack1)
pop(stack1)
while size(stack2)>0:
    push(rear(stack2),stack1)
    pop(stack2)
```

We have to move n-1 elements to the other stack twice.

---

7. Ans: $O(1)$

```
pop(stack1)
```

Just pop it out from the stack

---

8. Only when the stack is full that we need to enlarge it.If its capacity is enough for pushing all the N elements, it needs $O(N)$-time. Or we need to enlarge the stack.
Every time we call $void\ enlarge(structStack * S)$ it makes the capacity become 3 times bigger and the initial capacity is 1. Thus, only when capacity equal to 3^k $(k \in Nork = 0)$,the stack will be enlarged.
Therefore, we can say that its time complexity is $O(1+3+9+...+3^K)$ $K = [log_3 n]$
Let $n = 3^i, i \in N$. Then $1+3+9+...+n = \frac{3n-1}{2}$
So it needs $O(n)$-time

Problem 3

---

1. 先開一個大小為n且所有值為0的新陣列，用來記錄青蛙走過的地方，0是還沒走過1是已走過
   接著開始跑迴圈讓青蛙跳，每次到了新的地方就先檢查是否符合停止條件，若符合就回答青蛙會停。若不符合條件再檢查是否走過，若走過則青蛙將永遠不會停止。若未走過且未符合停止條件則青蛙紀錄步伐後繼續向下個點跳。
   空間複雜度隨著原本陣列大小而改變，最糟情況下必須走遍所有點才能得到結果
   ∴Time compexity = O(n) Space complexcity = O(n)

---

2. 先開一個大小為n且所有值為0的新陣列，用來記錄青蛙走過的地方，裡面紀錄的值是走過該點的次數
   先讓青蛙開始跳，當青蛙第一次走到第一個重覆的點時(新陣列中的值改為2時)開始計算步數，當青蛙再次走到第一個重覆的點時(新陣列中的值改為3時)結束計算
   最糟情況下青蛙會走遍所有點三次，而新陣列空間會和原始陣列相同
   因此Time compexity = O(n) Space complexcity = O(n)

---

3. 由於陣列嚴格遞增，所以最後面陣列的中位數必為最大，最前面的必為最小。而中間的中位數可以不去考慮。而在固定i的情況下，出現最小差值的j必出現在第$i+1$項 (因為數列嚴格遞增)
   所以只要遍訪每個點並記錄最小差值出現位置，因此時間複雜度為$O(n)$
   空間複雜度則因為只有多使用紀錄最小差值和i,j的位置因此為$O(1)$

---

4. ref : 和李長諺、石容居討論

   首先我們得先找出兩個decreasing node，讓它們的next成為$head1, head2$而這兩個decreasing node 則分別作為$end2, end1$將這個 circularly linked list先拆為兩個部分x和y。

   先將$head1, head2$中較小的設為$head-new$建立一個link list z。接著開始遍歷x,y 並比較它們的第一個元素，每次比較完後選出較小的那個利用指標接在z後面，當x,y其中之一沒有node時再將仍有node的另一方接在z後面。最後再將z的尾巴接回頭
   找decreasing node時最糟時跑遍L所有nodes最多需要n次，比較並重新排列時最多也是n次，因此時間複雜度為$O(2n)=O(n)-time$ 而額外使用的空間只有紀錄$head1, head2, end1, end2$不需要額外記憶體。因此空間複雜度為$O(1)-space$。