Data Structure(SE274): Assignment 1

201911189 한현영

A1-1.

```
def example1(S):
    """Return the sum of the elements in sequence S."""
    n = len(S)
    total = 0
    for j in range(n): # loop from 0 to n-1
        total += S[j]
    return total
```

elements n개를 갖는 for loop를 도는 경우의 시간 복잡도는 n이다. 이를 Big-Oh notation으로 표현할 경우 $O(n)$에 해당한다.

A1-2.

```
def example2(S):
    """Return the sum of the elements with even index in sequence S."""
    n = len(S)
    total = 0
    for j in range(0, n, 2): # note the increment of 2
        total += S[j]
    return total
```

range에 따라 for loop 안에서의 시간 복잡도는 $\frac{n}{2}$ 이다. 이를 Big-Oh notation으로 나타낼 경우 $O(n)$ 이다.

A1-3.

```
def example3(S):
    """Return the sum of the prefix sums of sequence S."""
    n = len(S)
    total = 0
    for j in range(n): # loop from 0 to n-1
        for k in range(1+j): # loop from 0 to j
            total += S[k]
    return total
```

$j = n + m$ 이라 하면, for문 안의 시간 복잡도는 $n(n+m) = n^2 + mn$ 이다. 이를 Big-Oh notation으로 나타내면, $O(n^2)$ 이다.

A1-4.

```
def example4(S):
    """Return the sum of the prefix sums of sequence S."""
    n = len(S)
    prefix = 0
    total = 0
    for j in range(n):
        prefix += S[j]
        total += prefix
    return total
```

for loop 안의 시간 복잡도는 $2n$이다. 이를 Big-Oh notation으로 나타내면, $O(n)$이다.

A1-5.

```
def example5(A, B): # assume that A and B have equal length
    """Return the number of elements in B equal to the sum of prefix sums in A."""
    n = len(A)
    count = 0
    for i in range(n): # loop from 0 to n-1
        total = 0
        for j in range(n): # loop from 0 to n-1
            for k in range(1+j): # loop from 0 to j
                total += A[k]
        if B[i] == total:
            count += 1
    return count
```

for loop 안의 시간 복잡도는 $an^3 + bn^2 + cn + \cdots$이다. 이를 Big-Oh notation으로 나타내면 $O(n^3)$이다.

A2-1. Order the following functions by asymptotic growth rate

우선 각각의 함수들을 Big-Oh notation 으로 표기해보자.

$$4nlogn + 2 = O(nlogn),$$

$$2^{10} = O(1)$$

$$2^{logn} = O(n)$$

$$3n + 100logn = O(n)$$

$$4n = O(n)$$

$$2^n = O(2^n)$$

$$n^2 + 10n = O(n^2)$$

$$n^3 = O(n^3)$$

$$nlogn = O(nlogn)$$

이를 통해 1차적인 asymptotic growth rate를 판단해보자.

$$O(1) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n)$$

따라서 각 함수들의 asymptotic growth rate는,

$$2^{10} < 2^{logn} < 3n + 100logn < 4n < nlogn < 4nlogn + 2 < n^2 + 10n < n^3 < 2^n$$

이다.

A2-2. Show that $n2$ is $\Omega(nlogn)$.

$n > n_0$에 대해 $|f(n)| \geq c|g(n)|$을 만족하는 상수가 존재하면 $f(n) = \Omega(g(n))$이다. $n > 0$이라 하고, 양변을 $nlogn$으로 나누어 주면, $\frac{n^2}{nlogn} \geq c$ 이다.

$n \to \infty$ 라고 할 때, $\lim_{n\to\infty} \frac{n}{logn} = \lim_{n\to\infty} \frac{1}{\frac{1}{\ln(1/2)}} = \lim_{n\to\infty} nln(\frac{1}{2}) = \infty$ (l'Hôpital's rule)이므로, $\infty \geq c$ 이고, $n^2 = \Omega(nlogn)$임을 알 수 있다.

A2-3. Show that if $p(n)$ is a polynomial in n, then log$p(n)$ is O(log$n$).

$p(n) = a_0 + a_1n + a_2n^2 + a_3n^3 + \cdots + a_kn^k$ 라 하자. $\log(p(n)) = \log(a_0 + a_1n + a_2n^2 + a_3n^3 + \cdots + a_kn^k)$이고, $n \to \infty$라고 할 때, 함수 진행의 dominant factor는 a_kn^k라는 것을 알 수 있다. 따라서 이를 Big-Oh notation으로 나타내면, $\log(a_kn^k) = O(logn)$이다.

A2-4. Show that if $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant a>0.

Big-Oh notation에 의해 $n > n_0$ 에 대해 $|d(n)| \leq c|f(n)|$ 일 것이고, $n > 0$일 때, 양변을 $f(n)$으로 나누면 $\frac{d(n)}{f(n)} \leq c$ 이다. 이때, $n \to \infty$ 인 상태에서 $\frac{d(n)}{f(n)}$ 은 임의의 값에 수렴하고, $d(n)$에 어떠한 상

수가 곱해지더라도 Big-Oh notation을 위한 과정 자체의 c가 조절되기 때문에, 수렴성은 잃지 않는다. 따라서 위는 성립한다.

A2-5. Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)-e(n)$ is not necessarily $O(f(n)-g(n))$.

$d(n) = kn, e(n) = ln$ 이라 하자. 그러면 $f(n) = n = g(n)$ 이다. 이때 $d(n) - e(n) = (k - l)n$ 이고, $O\big(f(n) - g(n)\big) = O(n - n) = O(1)$이고, $(k - l)n \neq O(1)$이므로, 반례가 성립한다.

A3-1. Implement a function that reverses a list of elements by pushing them onto a stack in one order, and writing them back to the list in reversed order.

```
class Stack:            # Stack 클래스 생성
    def __init__(self):
        self.arr = []
    def isEmpty(self):
        return len(self.arr) == 0
    def push(self, elem):
        self.arr.append(elem)
    def pop(self):
        if not self.isEmpty():
            return self.arr.pop(-1)

def reverse(list):
    arr = Stack()
    reversed_arr = []         # input stack의 reversed element가 들어갈 array 생성
    for push_elem in range(len(list)):
        arr.push(list[push_elem])
    for pop_elem in range(len(list)):
        reversed_arr.append(arr.pop())
    return reversed_arr
```

A3-2. Suppose an initially empty stack **S has executed a total of 25 push operations, 12 top operations, and 10 pop operations, 3 of which raised Empty errors that were caught and ignored. What is the current size of **S**?**

Empty error는 빈 리스트에서의 pop에서만 발생하므로, top operations의 경우 size에 관여하지 않는다. 따라서 25 – (10 – 3) = 18 이다.   #3번의 error 횟수 제외

A3-3. Suppose an initially empty queue **Q has executed a total of 32 enqueue operations, 10 first operations, and 15 dequeue operations, 5 of which raised Empty errors that were caught and ignored. What is the current size of **Q**?**

A3-2와 같은 원리로, first operations는 생각하지 않는다. 따라서 32 – (15 – 5) = 22 이다.   #5번의 error 횟수 제외

A3-4. Had the queue of the previous problem A3-3 been an instance of ArrayQueue that used an initial array of capacity 30, and had its size never been greater than 30, what would be the final value of the `_front` instance variable?

이 문제의 '_front' index를 0으로 설정하자. Queue에서 '_front'의 index는 dequeue의 횟수, resize에 따라 변화한다. 현재 고정된 capacity를 가짐에 따라 resize operation을 행하지 않으므로, dequeue의 횟수만 고려한다. enqueue하는 element = e_k $(k = 1,2,3, \dots)$라고 가정한다면, A3-3에서 dequeue의 횟수는 10회이므로, '_front'의 index는 10, 그에 따른 variable은 e_{11}이다.

A3-5. Give a complete ArrayDeque implementation of the double-ended queue, implemething the following ADT:

```
class ArrayDeque:              #ArrayDeque 클래스 생성

    def __init__(self, size = 10): #Default size는 10

        self.size = size

        self.front = 0

        self.rear = 0

        self.arr = [None] * size

        self.num_elem = 0              # __len__ func을 위함


    def __len__(self):

        return self.num_elem


    def resize(self): #Doubling process

        self.old_arr = self.arr
```

```
        self.new_arr = [None]*self.size*2
        for i in range(self.size):
            k = self.front
            self.new_arr[i] = self.old_arr[(k+i) % self.size]
        self.arr = self.new_arr
        self.front = 0
        self.rear = self.size - 1
        self.size *= 2

    def is_empty(self):
        return self.front == self.rear

    def is_full(self):              #Arr 안
        return self.front == (self.rear + 1) % self.size

    def add_first(self, elem):
        if self.is_full():
            self.resize()
        self.front = (self.front - 1) % self.size
        self.arr[self.front] = elem
        self.num_elem += 1

    def add_last(self, elem):
        if self.is_full():
            self.resize()
        self.rear = (self.rear + 1) % self.size
        self.arr[self.rear] = elem
        self.num_elem += 1

    def delete_first(self):
        if self.is_empty():
            raise EmptyError
        else:
```

```
            self.first_elem = self.arr[self.front]
            self.arr[self.front] = None
            self.front = (self.front + 1) % self.size
            self.num_elem -= 1
            return self.first_elem

    def delete_last(self):
        if self.is_empty():
            raise EmptyError
        else:
            self.last_elem = self.arr[self.rear]
            self.arr[self.rear] = None
            self.rear = (self.rear - 1) % self.size
            self.num_elem -= 1
            return self.last_elem

    def first(self):
        if self.is_empty():
            raise EmptyError
        else:
            return self.arr[self.front]

    def last(self):
        if self.is_empty():
            raise EmptyError
        else:
            return self.arr[self.rear]
```

A3-6. Suppose you have a deque **D containing the numbers \`(1,2,3,4,5,6,7,8)\`, in this order. Suppose further that you have an initially empty queue **Q**. Give a code fragment that uses only **D** and **Q** (and no other variables) and results in **D** storing the elements in the order \`(1,2,3,5,4,6,7,8)\`.**

#initial statement: D=(1,2,3,4,5,6,7,8) Q=(*,*,*,*,*,*,*,*), Deque의 first는 array의 0번째, last는 array의 -1번째로 가정

Q.enqueue(D.delete_first) #D=(*,2,3,4,5,6,7,8) Q=(1,*,*,*,*,*,*,*) Q.enqueue(D.delete_last) #D=(*,2,3,4,5,6,7,*) Q=(1,8,*,*,*,*,*,*)

Q.enqueue(D.delete_first) #D=(*,*,3,4,5,6,7,*) Q=(1,8,2,*,*,*,*,*)

Q.enqueue(D.delete_last) #D=(*,*,3,4,5,6,*,*) Q=(1,8,2,7,*,*,*,*)

Q.enqueue(D.delete_first) #D=(*,*,*,4,5,6,*,*) Q=(1,8,2,7,3,*,*,*)

Q.enqueue(D.delete_last) #D=(*,*,*,4,5,*,*,*) Q=(1,8,2,7,3,6,*,*)

Q.enqueue(D.delete_first) #D=(*,*,*,*,5,*,*,*) Q=(1,8,2,7,3,6,4,*)

Q.enqueue(D.delete_last) #D=(*,*,*,*,*,*,*,*) Q=(1,8,2,7,3,6,4,5)

D.add_first(Q.dequeue) #D=(1,*,*,*,*,*,*,*) Q=(*,8,2,7,3,6,4,5)

D.add_last(Q.dequeue) #D=(1,8,*,*,*,*,*,*) Q=(*,*,2,7,3,6,4,5)

D.add_first(Q.dequeue) #D=(2,1,8,*,*,*,*,*) Q=(*,*,*,7,3,6,4,5)

D.add_last(Q.dequeue) #D=(2,1,8,7,*,*,*,*) Q=(*,*,*,*,3,6,4,5)

D.add_first(Q.dequeue) #D=(3,2,1,2,7,*,*,*) Q=(*,*,*,*,*,6,4,5)

D.add_last(Q.dequeue) #D=(3,2,1,8,7,6,*,*) Q=(*,*,*,*,*,*,4,5)

D.add_first(Q.dequeue) #D=(4,3,2,1,8,7,6,*) Q=(*,*,*,*,*,*,*,5)

D.add_last(Q.dequeue) #D=(4,3,2,1,8,7,6,5) Q=(*,*,*,*,*,*,*,*)

Q.enqueue(D.delete_first) #D=(*,3,2,1,8,7,6,5) Q=(4,*,*,*,*,*,*,*)

Q.enqueue(D.delete_last) #D=(*,3,2,1,8,7,6,*) Q=(4,5,*,*,*,*,*,*)

Q.enqueue(D.delete_first) #D=(*,*,2,1,8,7,6,*) Q=(4,5,3,*,*,*,*,*)

Q.enqueue(D.delete_last) #D=(*,*,2,1,8,7,*,*) Q=(4,5,3,6,*,*,*,*)

Q.enqueue(D.delete_first) #D=(*,*,*,1,8,7,*,*) Q=(4,5,3,6,2,*,*,*)

Q.enqueue(D.delete_last) #D=(*,*,*,1,8,*,*,*) Q=(4,5,3,6,2,7,*,*)

Q.enqueue(D.delete_first) #D=(*,*,*,*,8,*,*,*) Q=(4,5,3,6,2,7,1,*)

Q.enqueue(D.delete_last) #D=(*,*,*,*,*,*,*,*) Q=(4,5,3,6,2,7,1,8)

D.add_first(Q.dequeue) #D=(4,*,*,*,*,*,*,*) Q=(*,5,3,6,2,7,1,8)

D.add_first(Q.dequeue) #D=(5,4,*,*,*,*,*,*) Q=(*,*,3,6,2,7,1,8)

D.add_first(Q.dequeue) #D=(3,5,4,*,*,*,*,*) Q=(*,*,*,6,2,7,1,8)

D.add_last(Q.dequeue) #D=(3,5,4,6,*,*,*,*) Q=(*,*,*,*,2,7,1,8)

D.add_first(Q.dequeue) #D=(2,3,5,4,6,*,*,*) Q=(*,*,*,*,*,7,1,8)

D.add_last(Q.dequeue) #D=(2,3,5,4,6,7,*,*) Q=(*,*,*,*,*,*,1,8)

D.add_first(Q.dequeue) #D=(1,2,3,5,4,6,7,*) Q=(*,*,*,*,*,*,*,8)

D.add_last(Q.dequeue) #D=(1,2,3,5,4,6,7,8) Q=(*,*,*,*,*,*,*,*), **FINISH!**

A3-Bonus. Show how to use a stack S and a queue Q to generate all possible subsets of an n-element set T nonrecursively (Don't use a recursion algorithm).

```
class Stack:        # 간단한 Stack 클래스 생성
    def __init__(self):
        self.stack = []
        self.pointer = 0
    def __len__(self):
        return self.pointer
    def is_empty(self):
        return len(self.stack) == 0
    def push(self, elem):
        self.pointer += 1
        return self.stack.append(elem)
    def pop(self):
        self.pointer -= 1
        return self.stack.pop(-1)


class Queue:        # 간단한 Queue 클래스 생성
    def __init__(self):
        self.queue = []
        self.pointer = 0
    def __len__(self):
        return self.pointer
    def is_empty(self):
        return len(self.queue) == 0
    def enqueue(self, elem):
        self.pointer += 1
        return self.queue.append(elem)
    def dequeue(self):
        self.pointer -= 1
        return self.queue.pop(0)
```

```
stack = Stack()
queue = Queue()
queue.enqueue(set())

for i in range(len(n)):
    stack.push(n[i])

while stack.is_empty() == False:        #스택의 element가 없어질 때까지 생성한 집합과 뽑힌 element가 합집합을 생성함, 이를 queue에 저장
    var = stack.pop()
    for i in range(len(q)):
        a = queue.dequeue()
        queue.enqueue(a)
        b = a|{var}
        queue.enqueue(b)

while queue.is_empty() == False:        #큐의 elenent가 없어질 때까지 생성한 부분집합을 print()함
    new = queue.dequeue()
    print(new)
```