Data Structure Assignment-1

20191103 곽현우

A 1-1

op.

```
1     def example1(S):
1       """Return the sum of the elements in sequence S."""
n       n = len(S)
n x1    total = 0
1       for j in range(n): # loop from 0 to n-1
          total += S[j]
        return total
```

n개.

↳ $2n+3+\cdots \Rightarrow O(n)$.

A 1-2

```
        def example2(S):
          """Return the sum of the elements with even
        index in sequence S."""
1         n = len(S)
1         total = 0
n/2.      for j in range(0, n, 2): # note the increment of 2
n/2 x1      total += S[j]
1         return total
```

↳ $\frac{n}{2} \times 2 + 3 + \cdots \Rightarrow O(n)$.

A 1-3.

op

```
        def example3(S):
          """Return the sum of the prefix sums of
        sequence S."""
1         n = len(S)
1         total = 0
n         for j in range(n): # loop from 0 to n-1
1+2+3+...n(n+1)/2   for k in range(1+j): # loop from 0 to j   1~n.
n(n+1)/2 x1           total += S[k]
1         return total
```

↳ $\frac{n(n+1)}{2} \times 2 + n + 3 + \cdots \Rightarrow O(n^2)$.

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
```

op: 1, 1, 1, n, n×1, n×1, 1

↳ n×3 + 4 + ··· ⇒ $\Theta(n)$.

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
```

op: 1, 1, n, n×1, n×n, n×(1+2+···n) = $\frac{n^2(n+1)}{2}$, $\frac{n^2(n+1)}{2}$ × 1, 0 — n, n×1

↳ $\frac{n^2(n+1)}{2} \times 2 + n^2 + n \times 4 + 2 \cdots \Rightarrow O(n^3)$

A2-1

$4n\log n+2 \Rightarrow O(n\log n)$ $\quad 2^{10} \Rightarrow O(1)$

$2^{\log n} \Rightarrow n^{\log 2} \Rightarrow O(n^{\log 2})$ $\quad 3n+100\log n \Rightarrow O(n)$

$4n \Rightarrow O(n)$ $\quad 2^n \Rightarrow O(2^n)$

$n^2+10n \Rightarrow O(n^2)$ $\quad n^3 \Rightarrow O(n^3)$

$n\log n \Rightarrow O(n\log n)$

이때 같은 Asymptotic growth Rate인 경우 최고차항의 계수를 통해 비교한다

$\therefore 2^{10} < 2^{\log n} < 3n+100\log n \le 4n < n\log n \le 4n\log n+2 < n^2+10n < n^3 < 2^n$

A2-3

$P(n) = a_0 + a_1 n^1 + a_2 n^2 + \cdots + a_k n^k$ 인 n에 대한 k차 다항식을 설정하자. 이때 n이 커질수록 최고차항인 $a_k n^k$에 근사하게 된다.

$\therefore \log P(n)$도 n이 커질수록 $\log a_k n^k$에 근사하게 된다

이때 $\log a_k n^k = k\log n + \log a_k$

$\therefore \log P(n)$ is $O(\log n)$

A2-2

n^2이 $\Omega(n\log n)$을 만족하기 위해서는 $n^2 \ge c\, n\log n$ for $n \ge n_0$을 만족하는 c와 n_0가 존재해야 한다. 위 식에서 n은 항상 커지므로 양수라 생각하면 $n \ge c\log n$ 이된다. 이때 $c\log n$은 그래프 개형상 항상 위로 볼록함으로 양수인 c에대해 성립한다.

$\therefore n^2$ is $\Omega(n\log n)$

A 2-4 $d(n)$ is $O(f(n))$ 라면 $d(n) \le cf(n)$ for $n \ge n_0$

이때 $d(n) \le cf(n) \rightsquigarrow ad(n) \le acf(n)$ 은 $a>0$ 일때 성립한다.

$\therefore ad(n) \Rightarrow g(n)$, $f(n)$의 계수를 e 라 할때

$g(n) \le ef(n)$ for $n \ge n_0$ ($e \ge ac$ 일때) 성립하므로

$ad(n)$ is $O(f(n))$ 이다.

A 2-5 $d(n)$ is $O(f(n))$ & $e(n)$ is $O(g(n))$

$\Rightarrow d(n) \le c_1 f(n)$ for $n \ge n_0'$, $e(n) \le c_2 g(n)$ for $n \ge n_0''$

이때 $d(n) - e(n) \le c_1 f(n) - c_2 g(n)$ 을 만족하지 않는 반례를 보이면

$d(n) - e(n)$ is not necessarily $O(f(n) - g(n))$ 를 보이게 된다.

반약 $d(n) = 2n^2 + n$, $f(n) = 3n^2$ $e(n) = 2n^2$, $g(n) = 3n^2$ 인 경우

$d(n) \le c_1 f(n)$ 과 $e(n) \le c_2 g(n)$을 만족한다.

그러나 $d(n) - e(n) = n$, $c_1 = c_2 = 1$ 인 경우 $c_1 f(n) - c_2 g(n) = 0$

$\therefore n \le 0$ 성립하지 않으므로 반드시 $O(f(n) - g(n))$ 은 아니다.

뒷장에 A 3-1 있습니다!

A3-1

```
class Stack():
    def __init__(self):
        self.stack = []

    def push(self, element):
        self.stack.append(element)

    def pop(self):
        self.pop_val = self.stack[-1]
        del self.stack[-1]
        return self.pop_val


def reversing_list(list):
    stack = Stack()
    reverse_list = []
    for push_ind in range(len(list)):
        stack.push(list[push_ind])
    for pop_ind in range(len(list)):
        reverse_list.append(stack.pop())
    return reverse_list
```

A3-2

top operation은 Stack의 가장 나중에 push 된, 즉 stack의 가장 나중에 추가된 element를 return 만 받고 remove는 하지않아 size에 영향을 안준다.

< operation . >	< stack size . >
∴ 25 push operation.	25.
10 pop operation (with 3 Error)	25 - (10 - 3) = 18.

∴ Current size of S = 18

A3-3 Stack의 top operation과 마찬가지로 queue의 first operation 역시 가장 먼저 추가된 즉은 head 변수에 있는 element를 return 받기만 할 뿐 queue의 size에 영향을 주지 않는다.

	operation.	queue size.
∴	32 enqueue op.	0 + 32.
	15 dequeue op. (with 5 error)	32 - (15-5) = 22

∴ Current size of Q = 22.

A3-4 강의 자료 중 Basic_Example 에 있는 ArrayQueue 구문을 참고했다.
이때 _front는 dequeue 메소드에서 self._front = (self._front+1) % len(self._data) 와 _resize 메소드에서 self._front=0 이 2가지 operation에 의해 영향을 받는다. 이때 플레이에 큰 영향을 끼므로 주로 _resize 가 발생하며 _front는 dequeue에 의해서 변한다. 이때 dequeue가 10번 일어나고 len(self._data) = 30 이므로 _front의 최종값은 10 이된다

↓

다음 A3-5 입니다.

A.3-5

```
class ArrayDeque():
    def __init__(self, cap =10):
        self.data = [None] * cap
        self.size = 0
        self.front = cap // 2
        self.back = cap // 2
        self.num_element = 0

    def resize(self):
        self.old_data = self.data
        self.new_data = [None] * 2 * self.size
        for i in range(0, self.size):
            self.new_data[self.front + i] = self.old_data[(self.front + i) %
self.size]
        self.back = (self.size - 1) * 2 - self.back
        self.size *= 2
        self.data = self.new_data

    def add_first(self, first_element):
        if self.num_element != self.size:
            if self.num_element == 0:
                self.data[self.front] = first_element
                self.num_element += 1
            else:
                self.data[self.front] = first_element
                self.num_element += 1
                self.front = (self.front - 1) % self.size
        else:
            self.resize()
            if self.num_element != self.size:
                if self.num_element == 0:
                    self.data[self.front] = first_element
                    self.num_element += 1
                else:
                    self.data[self.front] = first_element
                    self.num_element += 1
                    self.front = (self.front - 1) % self.size

    def add_last(self, last_element):
        if self.num_element != self.size:
            if self.num_element == 0:
                self.data[self.back] = last_element
                self.num_element += 1
            else:
                self.data[self.back] = last_element
                self.num_element += 1
                self.back = (self.back + 1) % self.size
        else:
            self.resize()
            if self.num_element != self.size:
                if self.num_element == 0:
                    self.data[self.back] = last_element
                    self.num_element += 1
                else:
                    self.data[self.back] = last_element
                    self.num_element += 1
```

```
                self.back = (self.back + 1) % self.size

    def delete_first(self):
        if self.num_element == 0:
            raise NotImplementedError
        else:
            self.num_element -= 1
            self.delete_first_element = self.data[self.front]
            self.data[self.front] = None
            self.front = (self.front + 1) % self.size
            return self.delete_first_element

    def delete_last(self):
        if self.num_element == 0:
            raise NotImplementedError
        else:
            self.num_element -= 1
            self.delete_last_element = self.data[self.back]
            self.data[self.back] = None
            self.front = (self.back - 1) % self.size
            return self.delete__last_element

    def first(self):
        if self.num_element == 0:
            raise NotImplementedError
        return self.data[self.front]

    def last(self):
        if self.num_element == 0:
            raise NotImplementedError
        return self.data[self.back]

    def is_empty(self):
        if self.size == 0:
            return True

    def len(self):
        return self.size
```

A3-6

```
class ArrayDeque():
    def __init__(self, cap =10):
        self.data = [None] * cap
        self.size = 0
        self.front = 0
        self.back = 0
        self.num_element = 0

    def resize(self):
        self.old_data = self.data
        self.new_data = [None] * 2 * self.size
        for i in range(0, self.size):
            self.new_data[self.front + i] = self.old_data[(self.front + i) %
self.size]
```

```
        self.back = (self.size - 1) * 2 - self.back
        self.size *= 2
        self.data = self.new_data

    def add_first(self, first_element):
        if self.num_element != self.size:
            if self.num_element == 0:
                self.data[self.front] = first_element
                self.num_element += 1
            else:
                self.data[self.front] = first_element
                self.num_element += 1
                self.front = (self.front - 1) % self.size
        else:
            self.resize()
            if self.num_element != self.size:
                if self.num_element == 0:
                    self.data[self.front] = first_element
                    self.num_element += 1
                else:
                    self.data[self.front] = first_element
                    self.num_element += 1
                    self.front = (self.front - 1) % self.size

    def add_last(self, last_element):
        if self.num_element != self.size:
            if self.num_element == 0:
                self.data[self.back] = last_element
                self.num_element += 1
            else:
                self.data[self.back] = last_element
                self.num_element += 1
                self.back = (self.back + 1) % self.size
        else:
            self.resize()
            if self.num_element != self.size:
                if self.num_element == 0:
                    self.data[self.back] = last_element
                    self.num_element += 1
                else:
                    self.data[self.back] = last_element
                    self.num_element += 1
                    self.back = (self.back + 1) % self.size

    def delete_first(self):
        if self.num_element == 0:
            raise NotImplementedError
        else:
            self.num_element -= 1
            self.delete_first_element = self.data[self.front]
            self.data[self.front] = None
            self.front = (self.front + 1) % self.size
            return self.delete_first_element

    def delete_last(self):
        if self.num_element == 0:
            raise NotImplementedError
        else:
```

```
            self.num_element -= 1
            self.delete_last_element = self.data[self.back]
            self.data[self.back] = None
            self.front = (self.back - 1) % self.size
            return self.delete__last_element

class ArrayQueue:
    def __init__(self, capacity):
        self._data = [None] * capacity
        self._size = 0
        self._front = 0

    def dequeue(self):
        if self.is_empty():
            raise NotImplementedError
        answer = self._data[self._front]
        self._data[self._front] = None

        self._front = (self._front + 1) % len(self._data)
        self._size -= 1

        return answer

    def enqueue(self, e):
        if self._size == len(self._data):
            self._resize(2 * len(self._data))
        avail = (self._front + self._size) % len(self._data)
        self._data[avail] = e
        self._size += 1

    def _resize(self, cap):
        old = self._data
        self._data = [None] * cap
        walk = self._front

        for k in range(self._size):
            self._data[k] = old[walk]
            walk = (1 + walk) % len(old)
        self._front = 0
D = ArrayDeque( 8 )
D.add_last( 1 )
D.add_last( 2 )
D.add_last( 3 )
D.add_last( 4 )
D.add_last( 5 )
D.add_last( 6 )
D.add_last( 7 )
D.add_last( 8 )
Q = ArrayQueue( 8 )
Q.enqueue(D.delete_first())
Q.enqueue(D.delete_first())
Q.enqueue(D.delete_first())
Q.enqueue(D.delete_first())
Q.enqueue(D.delete_first())
D.add_last(Q.dequeue())
D.add_last(Q.dequeue())
D.add_last(Q.dequeue())
```

```
D.add_first(Q.dequeue())
D.add_last(Q.dequeue())
```

A.Bonus