Onur Yaman
2007961
Homework-05

Task-1

R-6.8

By 32 enqueue operations, size += 32

By 10 first operations, size does not change

By 15 dequeue operations, size -= 15, but 5 of 15 are raised error, so size -= 10. (When queue is empty, 5 dequeue operations are implemented)

$32 - 10 = 22$

size $= 22$

R-6.9

Assuming 32 enqueue operations are implemented before 10 successful dequeue operations, it can be said that size may exceed the capacity 30 if the queue has initially capacity 30

In (circuler) array implementation according to lectures, _front += 10 since only successful dequeue operations would change the index of front.

Task-2

```
S = Array Stack()
Q = Array.Queue()
def pseudo_code(S, Q, x, r = False)
    while not S.is_empty()
        If S.top() != x :
            Q.enqueue(S.pop())
        else :
            r = True
            break
    # Change _front (front inxdex of Q)
    # so that Q._front = Q_size in each iteration.
    while not Q.is_empty()
        S.push(Q.dequeue)
    return r
```

The pseudo-code's time complexity is O(n) since while loops are running n+1 times, with all stack and queue operations whose time complexity is O(1) or O(1)*

```
#The code below is an implementation example for Task 2. It was checked by a stack parameter
#(that is the result cellstack from sample text case in assignment 1)

def pseudo_code(S,x,r=False,c=0):
    Q=ArrayQueue()
    while not S.is_empty():
        if S.top()!=x:
            Q.enqueue(S.pop())
            c+=1
        else:
            r=True
            break
    while not Q.is_empty():
        c-=1
        Q.front(c)
        S.push(Q.dequeue())
    return r,str(S)

class ArrayQueue:
    def __init__(self):
        self._data=[None]
        self._size=0
        self._front=0
    def __len__(self):
        return self._size
    def is_empty(self):
        return self._size==0
    def first(self):
        if self.is_empty():
            raise Exception("Queue is empty")
        return self._data[self._front]
    def dequeue(self):
        if self.is_empty():
            raise Exception("Queue is empty")
        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
    def front(self,value):
        self._front=value
    def __str__(self):
        return str(self._data)

class ArrayStack:
    def __init__(self,lst):
        self._data=lst
    def __len__(self):
        return len(self._data)
    def is_empty(self):
        return len(self._data)==0
    def push(self,e):
        self._data.append(e)
    def top(self):
        if self.is_empty():
            raise Exception("Stack is empty")
        return self._data[-1]
    def pop(self):
        if self.is_empty():
            raise Exception("Stack is empty")
        return self._data.pop()
    def __str__(self):
        return str(self._data)

S=ArrayStack([(1, 0),(1, 1),(2, 1),(3, 1),(3, 2),(3, 3),(3, 4),
   (4, 4),(5, 4),(6, 4),(6, 5),(6, 6),(6, 7),(6, 8),
   (6, 9),(6, 10),(5, 10),(5, 11),(5, 12)])

print(pseudo_code(S,(6,7)))
```