Tutorial (week 5)

All the answers can be given with pseudocode or with Python. Remember that most of the time, many solutions are possible. Note that certain classes defined in the solutions are not predefined in Python. For example, `Stack()` is not a predefined Python class.

1. Suppose you have access to an empty stack `S`. Write the content of `S` after execution of each of these commands:

```
S.push(4)
S.push(6)
S.pop()
S.push(8)
S.push(2)
S.push(1)
S.pop()
S.pop()
S.pop()
```

Solution:
$[4]$
$[4, 6]$
$[4]$
$[4, 8]$
$[4, 8, 2]$
$[4, 8, 2, 1]$
$[4, 8, 2]$
$[4, 8]$
$[4]$

2. Same question than before, but this time with a queue `Q`:

```
Q.enqueue(4)
Q.enqueue(6)
Q.dequeue()
Q.enqueue(8)
Q.enqueue(2)
Q.enqueue(1)
Q.dequeue()
Q.dequeue()
Q.dequeue()
```

Solution:
$[4]$

$[4, 6]$
$[6]$
$[6, 8]$
$[6, 8, 1]$
$[6, 8, 2, 1]$
$[8, 2, 1]$
$[2, 1]$
$[1]$

3. Suppose you are given an array, A, containing n numbers in order. Describe in Python or pseudocode an efficient algorithm for reversing the order of the numbers in A using a single for-loop that indexes through the cells of A, to insert each element into a stack, and then another for-loop that removes the elements from the stack and puts them back into A in reverse order. What is the running time of this algorithm?

   Solution:

```
def reverse_with_stack(A):
    '''
    INPUT: A n-element array A
    OUTPUT: Array A in reverse order, using only a stack
    '''

    s = Stack()  # create the stack
    for value in A:   # insert all elements in the stack
        s.push(value)
    for i in range(len(A)): # put them back in A from the stack
        A[i] = s.pop()

    return A
```

4. Solve the previous exercise using a queue instead of stack. That is, suppose you are given an array, A, containing n numbers in order, as in the previous exercise. Describe in Python or pseudocode an efficient algorithm for reversing the order of the numbers in A using a single for-loop that indexes through the cells of A, to insert each element into a queue, and then another for-loop that removes the elements from the queue and puts them back into A in reverse order. What is the running time of this algorithm?

   Solution:

```
def reverse_with_queue(A):
    '''
    INPUT: A n-element array A
    OUTPUT: Array A in reverse order, using only a queue
    '''

    q = Queue()  # create the queue
    for value in A:    # insert all element in the queue
        q.enqueue(value)
    for i in reversed(range(len(A))): # put them back in A from the queue in reverse
        order
        A[i] = q.dequeue()

    return A
```

5. Describe how to implement a queue using two stacks, so that the amortized running time for dequeue and enqueue is $O(1)$, assuming that the stacks support constant-time push, pop, and size methods. What is the worst-case running time of the `enqueue()` and `dequeue()` methods in this case ?

   Solution:
   Name the two stacks as E and D, for we will enqueue into E and dequeue from D. To implement `enqueue(e)`, simply call `E.push(e)`. To implement `dequeue()`, simply call `D.pop()`, provided that D is not empty. If D is empty, iteratively pop every element from E and push it onto D, until E is empty, and then call `D.pop()`.

   For the amortized analysis, charge \$2 to each enqueue, using \$1 to do the push into E. Imagine that we store the extra cyber-dollar with the element just pushed. We use the cyber-dollar associated with an element when we move it from E to D. This is sufficient, since we never move elements back from D to E. Finally, we charge \$1 for each dequeue to pay for the push from D (to remove the element returned). The total charges for n operations is $O(n)$, hence each operation runs in $O(1)$ amortized time.

6. Describe how to implement a stack using two queues. What is the running time of the `push()` and `pop()` methods in this case ?

   Solution:
   To implement a stack using two queues, Q1 and Q2, we can simply enqueue elements into Q1 whenever a push call is made. This takes $O(1)$ time to complete. For pop calls, we can dequeue all elements of Q1 and enqueue them into Q2 except for the last element which we set aside in a temp variable. We then return the elements to Q1 by dequeing from Q2 and enqueing into Q1. The last element that we set aside earlier is then returned as the result of the pop. Thus, performing a pop takes $O(n)$ time.

Hints

- **Question 3:** think about the order of the indexing for each of the for-loops.
- **Question 4:** think about the order of the indexing for each of the for-loops.
- **Question 5:** use one stack for enqueues and the other for dequeues (you still need to say how and you also need to do the amortized analysis).
- **Question 6:** use one queue as auxiliary storage and keep track of sizes as you are using it.