1. Amortized Analysis

(a)

- **Q:** Show that in the worst case, the total cost in tokens to increment the counter is O(n).
- **A:** 在最坏情况下，翻转前计数器中的数是 $1+2+2^2+2^3+\ldots+2^{k-1}=2^k-1=n-1$，需要翻转所有 k 位，需要的花费代价也是 $1+2+2^2+2^3+\ldots+2^{k-1}=2^k-1=n-1$，故复杂度是 $O(n)$。

(b)

- **Q:** How many tokens are required to flip bit 0 through n increments? (THINK: How often does bit 0 flip when we increment n times?)
- **A:** 第 0 位每次自增都需要翻转，n 次自增需要消耗 $2^0\times n=n$ 个代币。
- **Q:** How many tokens are required to flip bit 1 through n increments? (THINK: How often does bit 1 flip when we increment n times?)
- **A:** 第 1 为每 2 次自增翻转一次，n 次自增需要消耗 $2^1\times\frac{n}{2}=n$ 个代币。

(c)

- **Q:** Based on your answers to the previous questions, how many tokens are required to increment this counter through n increments using big O notation?
- **A:** 根据先前的分析，第 i 位每增加 2^i 次反转一次，n 次自增第 i 位一共需要消耗 $2^i\times\frac{n}{2^i}=n$ 个代币。

  一共有 k 位，所以一共需要 kn 个代币。
- **Q:** Based on your answer above, what is the amortized cost of a single increment using big O notation?
- **A:** 由于 $k=\log_2 n$，所以每次操作的均摊复杂度为 $O(\log n)$

2. A New Implementation of Queues

(a)

- **Q:** Write the queue function `queue_empty` that returns true if both stacks are empty.
- **A:**

```
bool queue_empty(queue Q)
{
    return stack_empty(Q->instack) && stack_empty(Q->outstack);
}
```

- **Q:** Write the queue function `enqueue` based on the description of the data structure above.
- **A:**

```
void enqueue(elem e, queue Q)
{
    push(e, Q->instack);
}
```

- **Q:** Write the queue function `dequeue` based on the description of the data structure above.
- **A:**

```
elem dequeue(queue Q)
//@requires !queue_empty(Q);
{
    if (stack_empty(Q->outstack))
        while(!stack_empty(Q->instack))
            push(pop(Q->instack), Q->outstack);
    //@assert !stack_empty(Q->outstack);
    return pop(Q->outstack);
}
```

(b)

- **Q:** We now determine the runtime complexity of the `enqueue` and `dequeue` operations.

  Let q represent the total number of elements in the queue.

  What is the worst-case runtime complexity of each of the following queue operations based on the description of the data structure implementation given above? Write ONE sentence that explains each answer.

- **A:**
  `enqueue`：$O(1)$，因为每次都是直接插入一个元素进入 `instack`，而 `push` 函数是 $O(1)$ 的

  `dequeue`：$O(q)$，因为可能所有元素全部都在 `instack` 内，需要进行 $q+1$ 次 `pop` 操作，q 次 `push` 操作，所以是 $O(q)$

(c)

- **Q:** How many tokens in total are required to enqueue an element? State for what purpose each token is used.
- **A:** 3 个代币，1 个代币用于 `enqueue` 中的 `push` 操作。另外 2 个代币之后再使用，分别用于在 `outstack` 为空的时候，从 `instack` 中 `pop` 出来，然后 `push` 进入 `outstack`。
- **Q:** How many tokens are required to dequeue an element? When the `outstack` is empty and we move elements from `instack` to `outstack`, why does this not require additional tokens?
- **A:** 1 个代币，从 `outstack` 中 `pop` 出来。因为在 `enqueue` 的时候已经预支过了。