**theory6**

**1.**

(1)(a)

**Solution：**根据规律，可以总结出位长为k的计数器一轮循环，所有情况下的币值消耗为：

(2^k)\*(2^0) + [2^(k-1)]\*(2^1) + [2^(k-2)]\*(2^2) + ... + (2^1)\*[2^(k-1)]

= 2^k + 2^k + 2^k + ... + 2^k

= (2^k)\*k

而总情况数为2^k，因此，位长为k的计数器平均消耗为k个币，其中已包含所有位翻转的两种最坏情况，其余位和最高位相反的情况下翻转。因此，最坏情况下的平均币值增量为k.

因此k比特计数器，平均翻转币值消耗为k个币，即 O(n).

(1)(b)

**Solution：**由于位0每次均翻转，所需要的币数为1。所以n次增加需要n次翻转，总共需要n个币。

**Solution：**位0每两次翻转，会产生进位，位1将翻转1次，所需要的币数为2。所以n次增加需要n/2向下取整次翻转，总共需要n个币。

(1) (c)

**Solution：**O(kn).

**Solution：**O(n).

**2.**

(4)(a)

**Solution：**

bool queue\_empty(queue Q)

{

return stack\_empty(Q->instack) && stack\_empty(Q->outstack);

}

**Solution：**

void enqueue(elem e, queue Q)

{

push(e, Q->instack);

}

**Solution：**

elem dequeue(queue Q)

//@require !queue\_empty(Q);

{

if (stack\_empty(Q->outstack)) {

while (!stack\_empty(Q->instack)) {

push(pop(Q->instack), Q->outstack);

}

}

return pop(Q->outstack);

}

(1)(b)

**Solution：**

enqueue:

O(1)

因为每次入队列，直接对Q->instack进行入栈操作即可，O(1).

dequeue:

O(q)

最坏情况是Q->outstack栈空，需要将Q->instack中所有的元素依次出栈，并入栈到Q->outstack，最后将Q->outstack栈顶元素出栈，O(q).

(2)(c)

**Solution：**元素仅有入队与出队操作，而对应栈的操作可分为：

Q->instack 入栈、出栈；

Q->outstack 入栈、出栈；

每个元素入队均是入栈Q->instack；而每个元素出栈，均需要先从Q->instack出栈，再入栈Q->outstack，最后在队首时执行出队操作，即出栈Q->outstack。每次入栈和出栈均需要一个代币，所以每个元素均摊4个代币，O(4)，即O(1).

**Solution：**通过上述分析可知，每个元素入队需要1个代币，O(1)；每个元素出队需要3个代币，O(3)，即O(1).