PS1-231830106

Problem Set 1

231830106 朱逸宸

Problem 1

1. 定义循环不变式：A[k]为一个包含k个数的数组。
2. 假设前提：A[1]显然有序。
3. 归纳假设：在第k次迭代后，A[k]有序。
4. 归纳证明：第k+1次迭代后，插入的第k+1个数，通过while循环可以确保A[k]在变为A[k+1]后仍然有序。
5. 证明while循环中插入的第k+1个数使得A[k+1]仍然有序：

```
for j = 2 to A.length
    key = A[j]
    i = j − 1
    while (i>0 and A[i]>key)
        A[i+1] = A[i]
        i = i − 1
    A[i+1] = key
```

5.1 定义循环不变式：

- A[1,2,3…i]为有序数组
- A[i]>key，将 `A[i]` 右移到 `A[i+1]` 。
  5.2 假设前提：
- key = A[j] = A[i+1]，A[1,2,3…j-1]有序
  5.3 归纳假设：
- 第k次迭代时，数组 A[1…i+1]A[1…i+1] 中，前 ii 个元素是有序的，并且 `A[i] > key` 。
  5.4 归纳步骤：
- 在第 k+1次迭代中，`A[i]` 被移动到 `A[i+1]`，且调整 `i = i − 1`，比较A[i-1]和key的大小
- $$\begin{cases} 将A[i]右移 & A[i-1] > key \\ 退出while循环 & A[i-1] \le key \end{cases}$$
- 此时，`key` 将被插入到 `A[i+1]`，保证了插入后，子数组 A[1…j]仍然是有序的。
  5.5 循环结束
  `while` 循环在以下两种情况下终止：

- 找到了一个 `A[i] ≤ key`，这意味着 `key` 应该插入到 `A[i+1]`，以保持数组的有序性。
- `i` 减到了 0，这意味着 `key` 应该插入到数组的最前面。
  循环终止后，`key` 被正确插入到子数组 A[1…j]A[1…j] 中，确保了子数组仍然有序。
  5.6 即证

6. while循环有效，归纳成立

Problem 2

$$GCD(x, y) = GCD(y, x\ mod\ y) \tag{1}$$

```
def GCD(x1, x2):
    if x2 == 0:
        return x1
    else:
        return GCD(x2, x1 % x2)
```

证明：

1. 有限时间内结束：

- 将 （x1 mod x2） 赋给了 x2 , 其中 （x1 mod x2）必然严格小于 x2或者等于0。而两者都是整数，因此GCD函数接受的第二个参数必然严格递减至0。

2. 已知(1)式正确，利用归纳法：
   2.1前提：GCD(x,0) = x
   2.2归纳假设：(1)式正确
   2.3归纳步骤：GCD函数接受的第二参数必然严格递减至0，此时GCD(x,0)=x，即证。

Problem 3

- (a) 反例：2^x 其中 $f(cn) = 2^{cn}$而$f(n) = 2^n$

  $$\lim_{n\to+\infty} \frac{2^{cn}}{2^n} = +\infty$$

  显然原式不正确
- (b)证明：
- $$\lim_{n\to+\infty} \frac{(n+a)^b}{n^b} = 1$$
- 因此$c_1 n^b \leq (n+a)^b \leq c_2 * n^b$，原式即证

Problem 4

$$\lg(\lg^* n) \quad 2^{\lg^* n} \quad (\sqrt{3})^{\lg n} \quad n^2 \quad n! \quad (\lg n)!$$
$$(10/9)^n \quad n^3 \quad \lg^2 n \quad \lg(n!) \quad 2^{2^n} \quad n^{1/\lg n}$$
$$\ln\ln n \quad \lg^* n \quad n\cdot 2^n \quad n^{\lg\lg n} \quad \ln n \quad 1 \quad 2^{\lg n}$$
$$\lg n \quad (\lg n)^{\lg n} \quad e^n \quad 4^{\lg n} \quad (n+1)! \quad \sqrt{\lg\lg n}$$
$$\lg^*(\lg n) \quad 2^{\sqrt{2}\lg n} \quad n \quad 2^n \quad n\lg n \quad 2^{2n+1}$$

结果:

$$\begin{aligned}
&1 = n^{1/\lg n} \ll 2^{\lg^* n} \ll \lg(\lg^* n) \ll \lg^* \lg n \ll \lg^* n \ll \sqrt{\lg\lg n} \ll \ln\ln n \ll \ln n = \lg n \\
&\ll \lg^2 n \ll n\lg n \ll (\sqrt{3})^{\lg n} \ll 2^{\lg n} = n \\
&\ll (\lg n)^{\lg n} \ll \lg(n!) \ll n^2 \ll n^3 \ll (\lg n)! \ll 2^{\sqrt{2}\lg n} \ll 4^{\lg n} \\
&\ll n^{\lg\lg n} \ll (10/9)^n \ll 2^n \ll n\cdot 2^n \ll e^n \ll 2^{2n+1} \ll n! \\
&\ll (n+1)! \ll 2^{2^n}
\end{aligned}$$

Problem 5

1. 概述：

- stack1用于储存数据，实现ENQUEUE功能
- stack2用于翻转stack1，实现FIFO从而实现DEQUEUE功能

2. 伪代码:

```
procedure ENQUEUE(x):
        stack1.push(x)

procedure DEQUEUE():
        if stack2 is empty:
                while not stack1.is_empty():
                        stack2.push(stack1.pop())
        return stack.pop()
```

3. 时间复杂度：
   3.1 ENQUEUE：时间复杂度与stack一样为Θ(1)。
   3.2 DEQUEUE：考虑到每个元素至多被压入stack2一次同时被stack2弹出一次而完成一个队列的弹出，时间复杂度应为Θ(1)。

Problem 6

1. 概述：

- stack用于储存数据
- min_stack用于维护最小值

2. 思路：
   2.1 push(x)将元素压入stack，同时与min_stack.top()进行比较，若并非最小则压入最小值，若确实更小则压入min_stack，从而实现同步。
   2.2 pop()直接将两个栈同时pop()即可
   2.3 min()直接弹出min_stack即可
3. 伪代码：

```
procedure PUSH(x):
        stack.push(x)
        if min_stack.is_empty or x < min_stack.top():
                min_stack.push(x)
        else:
                min_stack.push(min_stack.top())

procedure POP():
        if not stack.is_empty():
                stack.pop()
                min_stack.pop()

procedure MIN():
        if min_stack.is_empty():
                return min_stack.pop()
```

4. 时间复杂度与空间复杂度：
   4.1 时间复杂度：push()对两个栈进行了push()显然时间复杂度为Θ(1)
   pop()对两个栈进行了pop()显然时间复杂度也为Θ(1)
   min()让min_stack.pop()显然时间复杂度也为Θ(1)
   4.2 空间复杂度：格外使用了一个栈，但是两个栈道使用都是线性的，因此空间复杂 度应该为Θ(n)

Bonus Problem

(a)

```
function f(S):
        result = 0
        for element in S:
```

```
                result = result ⊕ element
        return result
```

(b)

```
function f(S):
        one,two = 0,0
        for element in S:
                two = two^(one & element)
                one = element & one
                common = one & two
                one &= one & ~common
                two &= two & ~common
        return one
```