Ps1-201300066

麻超　201300066

南京大学人工智能学院

1　Problem 1

解：1. 除此之外，还应证明结果与原数组有相同的元素。因为该算法只有交换两数位置这一过程，没有添加或删除元素，故其正确。

2. 循环不变量：A[i······n] 中最小元素的位置最多为 j.

证明：第一次循环时显然可得。

设 j=k，即证明 A[i······n] 的最小元素的位置至多为 k. 则需要比较 A[k] 与 A[k-1]. 在进行循环时，需要比较 A[k] 与 A[k-1] 的大小，易得无论这二者大小如何，最小的元素必然会存在于先前的 k-1 个元素之中。得证。

3. 循环不变量：A[1,······,i] 在循环过程中是按照顺序排列的。

证明：当 i=1 时，可知 j=2 时，A[1,······,n] 中 A[1] 为最小值，自然按顺序排列。

设当 i=k 时成立，即 A[1,······,k] 为按顺序排列。由 b 可知 A[k,······,n] 中 A[k] 为最小值。

则当 i=k+1 时，同样由 b 可知，A[k+1,······,n] 中 A[k+1] 为最小值。

所以 A[1,······,k+1] 按顺序排列。

当 i=n-1 时，循环终止，且 A[1,······,n] 按顺序排列。

由上述归纳法可知，A[1,······,i] 在循环过程中是按照顺序排列的。这一结论正确。

2　Problem 2

1.T(n) =c1+c2(n+2)+c3(n+1)=(c2+c3)n+(c1+2c2+c3) = $\Theta(n)$.

2. 循环不变量：在第 i 次循环后，$y = \sum_{j=i}^{n} c_j x^{j-i}$，为不变量。

证明：奠基：当 i=n 时，$y = c_n + x \times 0 = c_n x^0 \sum_{j=i}^{n} c_j x^{j-i}$

归纳：设当 i=k 时原式成立，即 $y = \sum_{j=k}^{n} c_k x^{j-k}$

则当 i=k+1 时，$y' = c_{k+1} + x \times y = c_{k+1} + x \times \sum_{j=k+1}^{n} c_j x^{j-k} = c_{k+1} + \sum_{j=k}^{n} c_j x^{j-k+1} = \sum_{j=k+1}^{n} c_{k+1} x^{j-k-1}$，原式成立。

故由数学归纳法，原式成立。

3 Problem 3

1. $f \in \Theta(g)$
2. $f \in O(g)$
3. $f \in O(g)$
4. $f \in \Theta(g)$
5. $f \in \Theta(g)$
6. $f \in \Theta(g)$
7. $f \in \Omega(g)$
8. $f \in \Omega(g)$
9. $f \in \Omega(g)$
10. $f \in \Omega(g)$
11. $f \in \Omega(g)$
12. $f \in O(g)$
13. $f \in O(g)$
14. $f \in \Theta(g)$
15. $f \in \Omega(g)$
16. $f \in O(g)$

4 Problem 4

$$1 = n^{1/\lg n} \ll \lg(\lg^* n) \ll \lg^* n = \lg^*(\lg n) \ll 2^{\lg^* n}$$
$$\ll \ln\ln n \ll \sqrt{\lg n} \ll \ln n \ll \lg^2 n \ll 2^{\sqrt{2\lg n}}$$
$$\ll (\sqrt{2}^{\lg n}) \ll n = 2^{\lg n} \ll n\lg n = \lg(n!) \ll n^2 \ll 4^{\lg n}$$
$$\ll n^3 \ll (\lg n)! \ll (\lg n)^{\lg n} \ll n^{\lg\lg n} \ll (3/2)^n$$
$$\ll 2^n \ll n * 2^n \ll e^n \ll n! \ll (n+1)! \ll 2^{2^n} \ll 2^{2^{n+1}}$$

5 Problem 5

解：数组为 [1⋯⋯n]. 其长度为 n.

设两个栈分别为 S，T. 令 S 的栈底元素为 A[1],T 的栈底元素为 A[n]，依次向数组中央靠拢。由于数组长度为 n，故若两堆栈的元素数量之和小于 n，不会发生溢出。设 S.top=k,T.top=m,k<m. 初始情况下 S.top=-1,T.top=n.

在一般和未到 n 的情况下，分别有如下操作：

```
1.Push(S,x):
    if S.top+1!=T.top
        S.top=S.top+1
        S[S.top]=x
    else return "errorsize"
2.Push(T,x):
    if S.top!=T.top
        T.top=T.top-1
        T[T.top]=x
    else return "errorsize"
3.Pop(S):
    if S.empty()=True:
        error"underflow"
    else:
        S.top=S.top-1
        return S[S.top+1]
4.Pop(T):
    if T.empty()=True:
```

```
        error"underflow"
    else:
        T.top=T.top+1
    return T[T.top+1]
5.S.empty():
    if S.top=-1
        return True
    else return false
6.T.empty():
    if T.top=n
        return True
    else return false
7.S.size():
    return S.top+1
8.T.size:
    return n-T.top
9.search(S,x):
    for i in 0 to S.top:
    if A[i]==x:
        return True
    return False
10.search(T,x):
    for i from T.top to n-1:
    if A[i]==x:
        return True
    return False
```

6 Problem 6

解：令 S，T 为两堆栈.

算法：令如下操作：

- Enqueue(x): 将 X 放入 S 中。

- Dequeue(x): 若 T 为空堆栈，则从堆栈 S 中反复取出元素并将其放入堆栈 T 中，直到堆栈 S 为空。

经过上述操作后，从 T 中取出元素，并 return it.

证明：堆栈的特点为后进先出，队列为先进先出。设 x,y 为两个元素。

first step:Enqueue: 将 x 和 y 先后入队，即放入 S 中。

next step:Dequeue: 出队操作，此时 T 为空，则将 S 中元素分别放入 T 中，由于堆栈执行后进先出原则，则 y 相较于 x 先进入 T 中，之后，从 T 中取出元素，同样由于堆栈的特点，x 相较于 y 会先出，这样便实现了先进先出的功能。

利用数学归纳法证明：

奠基：当只有一个元素时，自然正确。

归纳：设当有 n 个元素时正确，则当有 n+1 个元素时，将前 n 个元素分为一类，第 n+1 个元素分为第二类，则和上面 next step 的算法基本步骤一样，不再赘述。得证。

故该算法的正确性得到验证。

由于只进行了入队和出队操作，故时间复杂度为 $\Theta(1)$ time.