论题 2-1 作业

姓名：陈劭源　　　　学号：161240004

1 [TC] Problem 2-1

a. For every sublist of length k, insertion sort can sort it in $\Theta(k^2)$ worst-case time, and there are n/k sublists, so these sublists can be sorted by insertion sort in $\Theta(k^2)n/k = \Theta(nk)$ worst-case time.

b. Apply the divide-and-conquer approach. Divide these sublists into two groups, each contains $n/(2k)$ sublists, and merge these sublists recursively, and finally merge the two groups. Let m denote the number of the sublists, i.e. n/k, and $T(m)$ denote the total running time of merging m sublists. The "divide", "conquer" and "combine" steps take a running time of $\Theta(1)$, $2T(m/2)$, $\Theta(km)$, so the recurrence is

$$T(m) = \begin{cases} \Theta(1) & m = 1 \\ 2T(m/2) + \Theta(km) & m > 1 \end{cases}.$$

Solve this recurrence, we obtain $T(m) = \Theta(km\log(m)) = \Theta(m\log(k))$.

c. A standard merge sort takes a running time of $\Theta(n\log n)$. When $k = \Theta(\log(n))$, $\Theta(nk + n\log(n/k)) = \Theta(n\log n)$. For every $k = \omega(\log n)$, $\Theta(nk + n\log(n/k)) = \Theta(nk) = \omega(n\log n)$. Therefore, the largest value of k is $\Theta(\log n)$.

d. It mainly depends on the constant factors of merge sort and insertion sort. Let c_1 be the constant factor of merge sort, c_2 be the constant factor of insertion sort. We can rewrite the total running time as $T = c_1 nk + c_2 n\log(n/k)$. Minimize T with respect to k. Since $T'_k = nc_1 - nc_2/k$, $k = c_2/c_1$ is a minimum point of T. So we can choose $k = c_1/c_2$ in practice.

2 [TC] Problem 2-2

a. $\langle A'[1], A'[2], \cdots, A'[n]\rangle$ is a permutation of $\langle A[1], A[2], \cdots, A[n]\rangle$.

b. At the start of each iteration, the subarray $A[j..A.length]$ consists of the elements originally in $A[j..A.length]$, and $A[j]$ is the smallest item of $A[j..A.length]$.

Initialization Prior to the first iteration of this loop, we have $j = A.length$. Therefore, the subarray $A[j..A.length]$ consists only one element, and it is the original element in $A[j..A.length]$, and, of course, it is the smallest item of $A[j..A.length]$.

Maintenance In each iteration, we compare $A[j]$ with $A[j-1]$. If $A[j] \geq A[j-1]$, we do nothing. Because $A[j]$ is the smallest item of $A[j..A.length]$, $A[j-1]$ is the smallest item of $A[j-1..A.length]$. It $A[j] > A[j-1]$, we exchange $A[j]$ with $A[j-1]$. Because $A[j]$ is the smallest item of $A[j..A.length]$, $A[j-1]$ is the smallest item of $A[j-1..A.length]$ after exchanging, and $A[j-1..A.length]$ consists of the elements originally from $A[j-1..A.length]$. Summarizing, the invariant still holds after an iteration.

Termination Finally, we get $j = i$. Therefore, the subarray $A[i..A.length]$ consists of the elements originally in $A[i..A.length]$, and $A[i]$ is the smallest item of $A[i..A.length]$, and that is what we want.

c. At the start of each iteration, $A[1..i]$ contains the i smallest elements of $A[1..A.length]$, in sorted order, and $A[i+1..A.length]$ are the rest of the elements.

Initialization Prior to the first iteration, we have $i = 1$, and $A[1..i]$ contains only one element, in sorted order, trivially. Moreover, $A[i+1..A.length]$ are the rest of the elements obviously.

Maintenance After each iteration, $A[i+1..A.length]$ consists of the elements originally in $A[i+1..A.length]$, and $A[i+1]$ is the smallest item of $A[i+1..A.length]$. Since $A[1..i]$ are the i smallest elements of $A[1..A.length]$, $A[i+1]$ is greater than or equal to every element in $A[1..i]$, but less than or equal to every element in $A[i+2..A.length]$. Therefore, $A[1..i+1]$ contains the $i+1$ smallest elements of $A[1..A.length]$ in sorted order, and $A[i+2..A.length]$ is the rest of the elements, i.e. the loop invariant still holds.

Termination Finally we get $i = A.length$, therefore $A[1..A.length]$ contains the $A.length$ smallest elements of $A[1..A.length]$ in sorted order. Hence, the algorithm is correct.

d. Whatever the original sequence is, lines 3-4 will always be executed $n(n-1)/2$ times, where n is the number of the elements of the original sequence. Therefore the worst-case running time of bubble sort is $\Theta(n^2)$, as much as the worst-case running time of insertion sort.

3 [TC] Problem 2-3

a. $\Theta(n)$.

b. Given the coefficients $a_0, a_1, \cdots, a_n$ and a value for x:

```
y = 0
for i = 0 to n
    t = 1
    for j = 1 to i
        t = t * x
    y = y + a_i * t
```

c. **Initialization** Prior to the first iteration, we have $y = 0$ and $i = n$, so the loop invariant trivially holds.
Maintenance Assume, prior the tth iteration, we have $i = i_t$ and $y = y_t$. After the iteration and incrementing i, we get $i_{t+1} = i_t - 1$ and

$$\begin{aligned} y_{t+1} &= a_{i_t} + x * y_t = a_{i_t} + x * \sum_{k=0}^{n-(i_t+1)} a_{k+i_t+1} x^k \\ &= a_{i_t} + \sum_{k=0}^{n-(i_{t+1}+2)} a_{k+i_{t+1}+2} x^{k+1} = a_{i_{t+1}+1} + \sum_{k=1}^{n-(i_{t+1}+1)} a_{k+i_{t+1}+1} x^k \\ &= \sum_{k=0}^{n-(i_{t+1}+1)} a_{k+i_{t+1}+1} x^k, \end{aligned}$$

therefore the invariant still holds.

Termination At termination, we have $i = -1$. Substituting i for -1 in invariant, we obtain

$$y = \sum_{k=0}^{n} a_k x^k,$$

so the algorithm is correct.

4 [TC] Problem 2-3

a. $(2,1),(3,1),(8,6),(8,1),(6,1)$.

b. $\{n, n-1, \cdots 1\}$.
$n(n-1)/2$.

c. Assume there are $I(n)$ inversions in the input array, then the running time of insertion sort is $\Theta(n+I(n))$.
Proof: In page 26, t_j stands for the number of times the **while** loop test is executed for that value of j. Every time we execute the **while** loop, we insert $A[j]$ into the correct position, and exactly $t_j - 1$ inversions are eliminated. Finally, all the inversions are eliminated, and the array is sorted in order. Substituting $\sum_{j=2}^{n}(t_j - 1)$ for $I(n)$ and rewrite the formula, we get

$$T(n) = an + bI(n) + c$$

, therefore the running time of insertion sort is $\Theta(n+I(n))$.

d. Let c be an integer which stores the number of inversions.

$c = 0$
MODIFIED-MERGE-SORT$(A, 1, A.length)$

MODIFIED-MERGE-SORT(A, p, r)
if $p < r$
    $q = \lfloor (p+r)/2 \rfloor$
    MODIFIED-MERGE-SORT(A, p, q)
    MODIFIED-MERGE-SORT$(A, q+1, r)$
    MODIFIED-MERGE(A, p, q, r)

```
MODIFIED-MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
Let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[i]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        // Add the number of inversions (A[i], R[j]), (A[i + 1], A[j]) ··· (A[q], A[j]) to c
        c = c + (q − i + 1)
        j = j + 1
```

5 [TC] Problem 3-2

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^{ε}	yes	yes	no	no	no
n^k	c^n	yes	yes	no	no	no
$\sqrt{n}$	$n^{s}inn$	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

6 [TC] Problem 3-3

a. List:

$2^{2^{n+1}}$	2^{2^n}	$(n+1)!$	$n!$	$n2^n$	e^n
2^n	$(\frac{3}{2})^n$	$n^{\lg\lg n}$	$(\lg n)^{\lg n}$	$(\lg n)!$	$\sqrt{2}^{\lg n}$
n^3	$4^{\lg n}$	n^2	$n\lg n$	$2^{\lg n}$	n
$2^{\sqrt{2\lg n}}$	$\lg^2 n$	$\lg(n!)$	$\ln n$	$\sqrt{\lg n}$	$\ln\ln n$
$2^{\lg^* n}$	$\lg^* n$	$\lg^*(\lg n)$	$\lg(\lg^* n)$	$n^{1/\lg n}$	1

Equivalence classes: $\{2^{2^{n+1}}\}$, $\{2^{2^n}\}$, $\{(n+1)!\}$, $\{n!\}$, $\{n2^n\}$, $\{e^n\}$, $\{2^n\}$, $\{(\frac{3}{2})^n\}$, $\{(\lg n)^{\lg n}, n^{\lg\lg n}\}$, $\{(\lg n)!\}$, $\{\sqrt{2}^{\lg n}\}$, n^3, $\{n^2, 4^{\lg n}\}$, $\{n\lg n\}$, $\{n, 2^{\lg n}\}$, $\{2^{\sqrt{2\lg n}}\}$, $\{\lg^2 n\}$, $\lg(n!)\}$, $\{\sqrt{\lg n}\}$, $\{\ln n$, $\{\ln\ln n\}$, $\{2^{\lg^* n}\}$, $\{\lg^*(\lg n), \lg^* n\}$, $\{\lg(\lg^* n)\}$, $\{1, n^{1/\lg n}\}$.

b. $f(n) = (2^{2^{n+2}})^{\sin n}$

7 [TC] Problem 3-4

a. False. Take $f(n) = n$, $g(n) = n^2$, $f(n) = O(g(n))$, but $g(n) \neq O(f(n))$.

b. False. Take $f(n) = n$, $g(n) = n^2$, $f(n) + g(n) = n + n^2 = \Theta(n^2) \neq \Theta(\min(f(n), g(n))) = \Theta(n)$.

c. True. Since $f(n) = O(g(n))$, there exists positive constant $c > 1$ such that for all sufficiently large n, $1 \leq f(n) \leq cg(n)$ holds. Therefore, $\lg(f(n)) \leq \lg c + \lg(g(n))$. Take $c' = \lg c + 1$, then $0 \leq \lg(f(n)) \leq c' \lg(g(n))$ for all sufficiently large n. Hence, $\lg(f(n)) = O(\lg(g(n)))$.

d. False. Take $f(n) = n\lg n$, $g(n) = \lg(n!)$, $f(n) = O(g(n))$, however, $2^{f(n)} = n^n \neq O(2^{g(n)}) = O(n!)$.

e. False. Take $f(n) = 1/n$, then $(f(n))^2 = 1/n^2$, however, $\lim_{n\to\infty} f(n)/(f(n))^2 = +\infty$, that means, $f(n)$ could not be asymptotically upper-bounded by $(f(n))^2$.

f. True. By transpose symmetry we know this is true.

g. False. Take $f(n) = 4^n$, then $\Theta(f(n/2)) = \Theta(2^n)$, however $4^n \neq \Theta(2^n)$.

h. True. By the definition of o-notation, for any positive constant c, there exists a positive integer n_0, for any integer $n > n_0$, $0 \leq o(f(n)) \leq cf(n)$ holds. Take $c = 1$, for sufficiently large positive integer n, $f(n) \leq f(n) + o(f(n)) \leq 2f(n)$, therefore $f(n) + o(f(n)) = \Theta(f(n))$.