第二章程序的性能：35 页 Q15 、Q16；
36 页 Q18；P81 Q20；
39 页 Q24；
54 页 Q37(b)(d)(f)(g)(h)(i)(m)

Q15

The third `for` loop is entered n^3 times. So, the total number of multiplications is n^3.

Q16

The third `for` loop is entered `mpn` times and the number of multiplications is `mpn`.

Q18

The first minmax function makes zero comparisons when `n` < 1 and `2(n - 1)` comparisons when `n` ≥ 1.

The second function also makes zero comparisons when `n` < 1. However, when `n` ≥ 1, the best case number of comparisons is `n - 1` and the worst case number is `2(n - 1)`. The second function is expected to run faster on average.

Q20

The best case is one and the worst `n + 1`. In comparing the two codes, we see that the new code has an additional assignment `a[n] = x`. However, the `for` loop is simplified and does not include the check `i < n`. In a successful search, the tradeoff is between one assignment and between 1 and `n` comparisons of type `i < n`. In an unsuccessful search, the tradeoff is between 1 assignment and an additional comparsion between elements of type `T` and `n + 1` comparisons of the for `i < n`. For most data types `T`, we expect the new code to run faster than the old one.

Q24

(a) $t(n)=\begin{cases}1, n<1\\ 1+t(n-1), n\geq 1\end{cases}$ so, t(n) = 1 + t(n - 1) = 2 + t(n - 2) = ⋯ = n

(b) 4 + O(n) = O(n)

(c) 3 + O(n) = O(n)

Q37

(b)

Statement	s/e	Frequency	Total steps
int MinMax(T a[], int n, int& Min, int& Max)	0	0	$\Theta(0)$
{// Find min and max elements in a[0:n-1].	0	0	$\Theta(0)$
if (n < 1) return 0;	1	1	$\Theta(1)$
Min = Max = 0;	1	1	$\Theta(1)$
for (int i = 1; i < n; i++) {	1	$\Theta(n)$	$\Theta(n)$
if (a[Min] > a[i]) Min = i;	1	$\Theta(n)$	$\Theta(n)$
if (a[Max] < a[i]) Max = i;	1	$\Theta(n)$	$\Theta(n)$
}	0	0	$\Theta(0)$
return 1;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{MinMax}(n) = \Theta(n)$$

(d)

Statement	s/e	Frequency	Total steps
void Mult(T **a, T **b, T **c, int n)	0	0	$\Theta(0)$
{// Multiply the n x n matrices a and b to get c.	0	0	$\Theta(0)$
for (int i = 0; i < n; i++)	1	$\Theta(n)$	$\Theta(n)$
for (int j = 0; j < n; j++) {	1	$\Theta(n^2)$	$\Theta(n^2)$
T sum = 0;	1	$\Theta(n^2)$	$\Theta(n^2)$
for (int k = 0; k < n; k++)	1	$\Theta(n^3)$	$\Theta(n^3)$
sum += a[i][k] * b[k][j];	1	$\Theta(n^3)$	$\Theta(n^3)$
c[i][j] = sum;	1	$\Theta(n^2)$	$\Theta(n^2)$
}	0	0	$\Theta(0)$
}	0	0	$\Theta(0)$

$$t_{Mult}(n) = \Theta(n^3)$$

(f)

Statement	s/e	Frequency	Total steps
int Max(T a[], int n)	0	0	$\Theta(0)$
{// Locate the largest element in a[0:n-1].	0	0	$\Theta(0)$
int pos = 0;	1	1	$\Theta(1)$
for (int i = 1; i < n; i++)	1	$\Theta(n)$	$\Theta(n)$
if (a[pos] < a[i])	1	$\Theta(n)$	$\Theta(n)$
pos = i;	1	$O(n)$	$O(n)$
return pos;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{Max}(n) = \Theta(n)$$

(g)

Statement	s/e	Frequency	Total steps
T PolyEval(T coeff[], int n, const T& x)	0	0	$\Theta(0)$
{// Evaluate the degree n polynomial with	0	0	$\Theta(0)$
// coefficients coeff[0:n] at the point x.			
T y = 1, value = coeff[0];	1	1	$\Theta(1)$
for (int i = 1; i <= n; i++) {	1	$\Theta(n)$	$\Theta(n)$
// add in next term	0	0	$\Theta(0)$
y *= x;	1	$\Theta(n)$	$\Theta(n)$
value += y * coeff[i];	1	$\Theta(n)$	$\Theta(n)$
}	0	0	$\Theta(0)$
return value;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{PolyEval}(n) = \Theta(n)$$

(h)

Statement	s/e	Frequency	Total steps
T Horner(T coeff[], int n, const T& x)	0	0	$\Theta(0)$
{// Evaluate the degree n polynomial with	0	0	$\Theta(0)$
// coefficients coeff[0:n] at the point x.	0	0	$\Theta(0)$
T value = coeff[n];	1	1	$\Theta(1)$
for (int i = 1; i <= n; i++)	1	$\Theta(n)$	$\Theta(n)$
value = value * x + coeff[n - i];	1	$\Theta(n)$	$\Theta(n)$
return value;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{Horner}(n) = \Theta(n)$$

(i)

Statement	s/e	Frequency	Total steps
void Rank(T a[], int n, int r[])	0	0	$\Theta(0)$
{// Rank the n elements a[0:n-1].	0	0	$\Theta(0)$
for (int i = 0; i < n; i++)	1	$\Theta(n)$	$\Theta(n)$
r[i] = 0;	1	$\Theta(n)$	$\Theta(n)$
for (i = 1; i < n; i++)	1	$\Theta(n)$	$\Theta(n)$
for (int j = 0; j < i; j++)	1	$\Theta(\sum_{i=1}^{n-1} i)$	$\Theta(n^2)$
if (a[j] <= a[i]) r[i]++;	1	$\Theta(n^2)$	$\Theta(n^2)$
else r[j]++;	1	$\Omega(0)$, $O(n^2)$	$\Omega(0)$, $O(n^2)$
}	0	0	$\Theta(0)$

$$t_{Rank}(n) = \Theta(n^2)$$

(m)

Statement	s/e	Frequency	Total steps
void Insert(T a[], int n, const T& x)	0	0	$\Theta(0)$
{// Insert x into the sorted array a[0:n-1].	0	0	$\Theta(0)$
int i;	0	0	$\Theta(0)$
for (i = n-1; i >= 0 && x < a[i]; i--)	1	$\Omega(1)$, $O(n)$	$\Omega(1)$, $O(n)$
a[i+1] = a[i];	1	$\Omega(1)$, $O(n)$	$\Omega(1)$, $O(n)$
a[i+1] = x;	1	1	$\Theta(1)$
}	0	0	$\Theta(0)$

$$t_{Insert}(n) = \Omega(1), O(n)$$

Statement	s/e	Frequency	Total steps
void InsertionSort(T a[], int n)	0	0	$\Theta(0)$
{	0	0	$\Theta(0)$
for (int i = 1; i < n; i++) {	1	$\Theta(n)$	$\Theta(n)$
T t = a[i];	1	$\Theta(n)$	$\Theta(n)$
Insert(a, i, t);	$\Omega(1)$, $O(i)$	$\Theta(n)$	$\Omega(n)$, $O(n^2)$
}	0	0	$\Theta(0)$
}	0	0	$\Theta(0)$

$$t_{InsertionSort}(n) = \Omega(n), O(n^2)$$

Q.26

（1）
```
for m = 2 to n do{
    for i = 0 to n-m do{
        j = i + m
        w(i, j) = w(i, j-1) + P(i) + Q(j)
        c(i, j)= min_{i<l≤j} { c(i, l-1) + c (l, j) } + w(i, j)
    }
}
```

W(n,n)，P(n)，Q(n)，c(n,n)为算法中使用的数组并已初始化。

答：
```
for m = 2 to n do{                                          Θ (n)
    for i = 0 to n-m do{                                    Θ (n^2)
        j = i + m                                           Θ (n^2)
        w(i, j) = w(i, j-1) + P(i) + Q(j)                   Θ (n^2)
        c(i, j )= min_{i<l≤j} { c(i, l-1) + c (l, j) } + w(i, j)    Θ (n^3)
        c(i, j) = w(i, j) + c(i, k-1) + c(k, j)             Θ (n^2)
    }
}
```

$\min_{i<l\le j}$ { c(i, l-1) + c (l, j) } 的执行时间为 $O(j-i)=O(m)$；

内层 for-循环的执行时间为

$O(m(n-m))$；总的执行时间 t(n)= $O(\sum_{m=2}^{n} m(n-m)) = O(n^3)$

Q.27

```
void sort ( int E[ ], int n)
        {//对数组E中的n个元素进行排序
          if (n > 1) {
            i = n/2;
            j = n-i;
            令A 包含E中的前i 个元素
            令B 包含E中余下的j 个元素
            sort (A , i) ;
            sort (B , j) ;
            merge (A, B, E, i, j) ; //把A和B合并到E
            }
        }
```

其中merge (A, B, E, i, j)的时间复杂度是O(i+j)

答：$t(n)=\begin{cases} d & n\le 1 \\ t(\lfloor n/2 \rfloor + t(\lceil n/2 \rceil + cn & n>1 \end{cases}$

令 $n=2^m$,则

$t(n)=2t(n/2)+cn=2[2t(n/2^2)+c(n/2)]+cn=2^2t(n/2^2)+2cn$

$=\cdots\cdots=2^m t(n/2^m)+mcn=nd+cnlog_2^n$

所以 $t(n)= \Theta(nlog_2^n)$

补充题

1．

$$T(N) \le \begin{cases} 0 & \text{if } N=1 \\ \underbrace{T(\lceil N/2\rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor N/2\rfloor)}_{\text{solve right half}} + \underbrace{cN}_{\text{combine}} & \text{otherwise} \end{cases}$$

$$\Rightarrow T(N) \le cN\lceil \log_2 N\rceil$$

Proof by induction on N.

- Base case: N = 1.
- Define $n_1 = \lfloor n/2\rfloor$, $n_2 = \lceil n/2\rceil$.
- Induction step: assume true for 1, 2, . . . , N – 1.

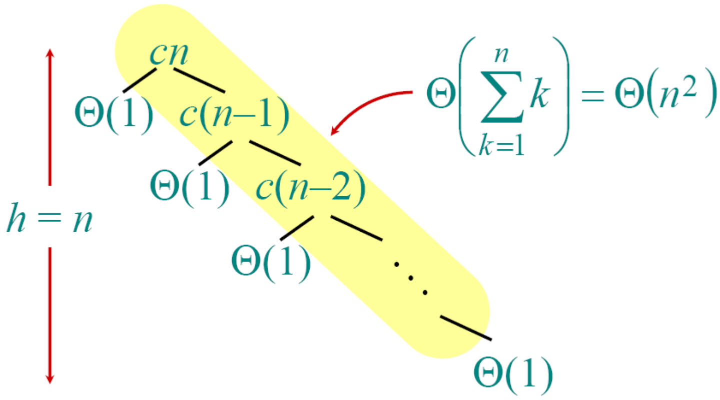
$$\begin{aligned} T(N) &\le T(n_1) + T(n_2) + cn \\ &\le cn_1\lceil \log_2 n_1\rceil + cn_2\lceil \log_2 n_2\rceil + cn \\ &\le cn_1\lceil \log_2 n_2\rceil + cn_2\lceil \log_2 n_2\rceil + cn \\ &= cn\lceil \log_2 n_2\rceil + cn \\ &\le cn(\lceil \log_2 n\rceil - 1) + cn \\ &= cn\lceil \log_2 n\rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2\rceil \\ &\le \left\lceil 2^{\lceil \log_2 n\rceil}/2 \right\rceil \\ \Rightarrow \log_2 n_2 &\le \lceil \log_2 n\rceil - 1 \end{aligned}$$

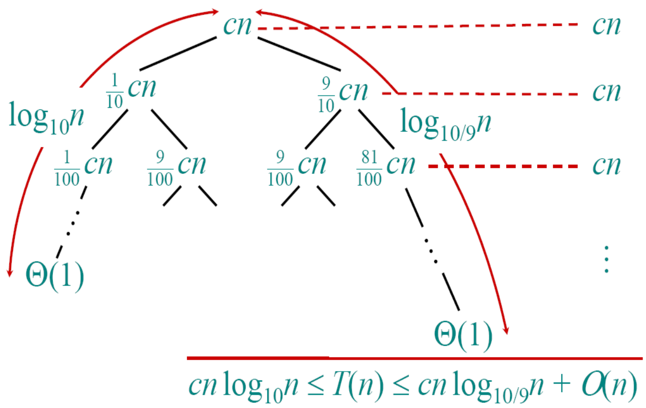
7

2．Case 1: $\log_b a=1$，$n^{\log a}$ 是 $n^{1/2}$ 的上界（例如 ε 取 1/3）。所以 T(n)=Θ(n).

3. Θ(n^2)



4. Θ(nlogn)



5. 应用Master's方法求解书中157页练习Q37的递归式

（a）Case 1： Θ($n^{\log 10}$)

（b）Case 3： Θ(n^5)

（c）Case 2：$\Theta(n^3\log n)$

（d）Case 2 ：$\Theta(n^3\log^3 n)$

(e)case 3:a=9,b=2, $f(n)=n^2 2^n$,任取 c

$t(n)=\Theta(n^2 2^n)$

(f)case 3: a=3,b=8, $f(n)=n^2 2^n\log n$,任取 c

$t(n)=\Theta(n^2 2^n\log n)$

(g)case 1:a=128,b=2,f(n)=6n

$t(n)=\Theta(n^7)$

(h)case3,$\log_b a=7$,$f(n)=n^8=\Omega(n^{7+\varepsilon})$,

$8(n/2)^8<cn^8$,$t(n)=\Theta(n^8)$

(i)case 3,$t(n)=\Theta(2^n/n)$

(j)case 1,$t(n)=\Theta(n^7)$

6．练习14(a)

解：因每次将较大的段进栈，留下较小的段继续分划，所以，至多$\lceil\log_2 n\rceil$次分划留下的段长度为1，所以进栈的段至多$\lceil\log_2 n\rceil$。

7．分析当r取3时是否能在O(n)时间内求解选择问题?分析r=7时选择算法的时间复杂度。

解：当r＝3 时，找不到满足α ＋1/3<1的正数α 使得n-2*[(1/2)(n/3)]< αn，[ ]表示向上取整。所以不能确定是否能在O(n)时间内用本节介绍的分治法求解。

当r=7时，可证明：$T(n)\le T(n/7)+T(5n/6)+cn$，当n>1时成立。用归纳法可证明：$T(n)\le 42cn$成立。

8．(123 页练习 4) 证明按普通 2×2 分块矩阵乘法得到的分治法算法的时间复杂度为 $\Theta(n^3)$.

解. $t(n)=8t(n/2)+cn^2=8[8t(n/2^2)+c(n/2)^2]+cn^2$

$=8^2t(n/2^2)+c(8/4)n^2+cn^2$

$=8^2[8t(n/2^3)+c(n/2^2)^2]+c(8/4)n^2+cn^2$

$=8^3t(n/2^3)+c(8/4)^2n^2+c(8/4)n^2+cn^2$

=……

$=8^kt(1)+cn^2(1+2+\cdots+2^{k-1})$

$=8^kt(1)+cn^2(2^k-1)=(2^k)^3+cn^2(n-1)=n^3+cn^2(n-1)=\Theta(n^3)$

9．分析书中递归式(14.7).

解 设 $n=4^k$，即 $2^k\times2^k$ 棋盘的方格数目。

$t(k)=d \qquad k=0$

$t(k)=4t(k-1)+c \qquad k>0$

$t(k)=4t(k-1)+c$

$=4[4t(k-2)+c]+c$

$=\ldots\ldots$

$=4^kt(0)+c(1+4+4^2+\ldots+4^{k-1})$