unit2

practice1

见书本(中文版P38)

practice2

见书本(中文版P39)

practice3

不相同，不同的编译器可能采取不同的操作生成不同的代码

practice4

略

practice5

1. 8 * 3 = 24 bytes
2. 4 * 10 * 100 = 4000 bytes
3. 8 * 100 * 5 * 20 = 80000 bytes
4. 4 * 10 * 10 * 10 * 5 = 20000 bytes
5. 1 * 2 * 3 * 4 = 24 bytes
6. 4 * 3 * 3 * 3 * 3 = 324 bytes

practice6

设能找到，递归深度为 i ，则 i∈[1,n]; 找不到则递归深度为 n+1
假设概率都是相等的，此时
$S_{rSequentialSearch}(n) = (12 + sizeof(T)) \times \frac{1+2+...+(n+1)}{n+1} = \frac{(12+sizeof(T))\times(n+2)}{2}$
注：暂时没理解$S_p(n)$ 和 $S_{rSequentialSearch}(n)$ 的区别

practice7

见 code/practice7

practice8

序号	操作(程序2-3)
1	y(1) *= 2 value(7) += 2 * 6
2	y(2) *= 2 value(19) += 4 * 5
3	y(4) *= 2 value(39) += 8 * 4
4	y(8) *= 2 value(71) += 16 * 3

序号	操作(程序2-4)
1	value(3) = 3 * 2 + 4
2	value(10) = 10 * 2 + 5
3	value(25) = 25 * 2 + 6
4	value(56) = 56 * 2 + 7

---

practice9

见 code/practice9

---

practice10

选择排序
3265**9**4**8**
3265**8**49
32**6**5**4**89
324**5**689
32**4**5689
3245689
2345689

---

practice11

一次冒泡排序
3265948

一次冒泡排序
2**36**5948
23**65**948
235**69**48
2356**94**8
2356**498**
2365489

practice12

冒泡排序
3265948
2365489
2354689
2345689
2345689
2345689
2345689
2345689

practice13

插入
1246789_
124678_9
12467_89
1246_789
124_6789
12_46789
12346789

practice14

原地重排

原地重排
6 7 8 5 2 0 **3** 1 4 **g** h i f c a **d** b e
3 7 8 **5** 2 0 6 1 4 **d** h i **f** c a g b e
5 7 8 3 2 **0** 6 1 4 **f** h i d c **a** g b e
0 7 8 3 2 5 6 1 4 **a** h i d c f g b e
0 **7** 8 3 2 5 6 **1** 4 a **h** i d c f g **b** e
0 **1** 8 3 2 5 6 7 4 a **b** i d c f g h e
0 1 **8** 3 2 5 6 7 **4** a b **i** d c f g h **e**
0 1 **4** 3 **2** 5 6 7 8 a b **e** d **c** f g h i
0 1 2 3 4 5 6 7 8 a b c d e f g h i

---

practice15

1.

及时终止选择排序
9876543210
0**8**76543**1**9
01**7**6543**2**89
012**6**54**3**789
0123**54**6789
01234**5**6789
0123456789

2. 略

---

practice16

及时终止冒泡排序
4267109853

及时终止冒泡排序
2461078539
2410675389
2104653789
1024536789
0124356789
0123456789
0123456789

practice17

插入排序
4**267109853**
24**67109853**
246**7109853**
2467**109853**
12467**09853**
012467**9853**
0124679**853**
01246789**53**
012456789**3**
0123456789

practice18

$n \quad (0 \sim n-1)$

practice19

$n-1 \quad (n-1 \sim 1)$

practice20

[5,1,2,3,4]

practice21

$numberOfRows \times numberOfColumns$

---

practice22

1. $$\sum_{i=0}^{rows-1}(rows-i-1) = rows \times (rows-1) - \sum_{i=0}^{rows-1} i = \frac{rows \times (rows-1)}{2}$$
2. $$\frac{rows^2}{2} - rows = \frac{rows \times (rows-1)}{2}$$

---

practice23

1. $$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = n^3$$
2. $$n \times n \times n = n^3$$

---

practice24

1. $$\sum_{i=0}^{m-1}\sum_{j=0}^{p-1}\sum_{k=0}^{n-1} 1 = m \times p \times n$$
2. $$n \times m \times p$$

---

practice25

$$\sum_{k=0}^{m-1}\sum_{i=k}^{m} 2 = \sum_{k=0}^{m-1} 2(m-k) = m(m+1)$$

---

practice26

1. 程序2-24：2(n-1)
2. 程序2-25：1(最好) 2n(最坏)=1+n+(n-1)
3. 除去两者间相同的前两行和最后一行：
   - 程序 2-24 总步数 = n + 2*n = 3n
   - 程序 2-25 总步数 = n + n = 2n
   - 结论：程序 2-24 消耗时间大概是程序 2-25 的 1.5 倍

---

practice27

1. 不存在：需要 n 次

2. 存在：设在 a[i] 处找到，则需要 n-i 次，假设没有重复的值且每个值的概率都是相同的，平均次数为

$$\frac{1}{n}\sum_{i=0}^{n-1}(n-i)=\frac{n+1}{2}$$

---

practice28

$n \quad (0 \sim n-1)$

---

practice29

见 code/practice29

---

practice30

略

---

practice31

略

---

practice32

1. **rSequentialSearch:**
    1. n=0, stepCount = 2
    2. n>0
        1. 不存在：$stepCount = 3n+2$
        2. 存在：设 a[i]==x，$stepCount_i = 3(n-i)$
            - $stepCount_{avg} = \frac{1}{n}\sum_{i=0}^{n-1}3(n-i) = \frac{5n+1}{2}$

$$stepCount_{avg} = \begin{cases} 2 & n=0 \\ 3n+2 & notFound \\ \frac{5n+1}{2} \end{cases}$$

2. **sequentialSearch:** 假设 n > 0，由代码可知，存不存在的步数一样，因此这里一起讨论
    - 假设不存在重复的值，且找到每个值的概率相同，则 $stepCount_i = i+5, a[i] == x$

$$stepCount_{avg} = \frac{1}{n+1}\sum_{i=0}^{n}(i+5) = n+5, n>0$$

3. **insert**:
    - 假设不存在重复的值，且插入每个位置的概率相同。设插入的位置为 i (n>0)，则

$$stepCount_i = 3 + (n - i + 1) + (n - i) = 2(n - i + 2)$$

$$stepCount_{avg} = \frac{1}{n+1}\sum_{i=0}^{n} 2(n - i + 2) = 2(n + 2) - n = 3n + 4, n > 0$$

经验证，n = 0，满足上式，则

$$stepCount_{avg} = \frac{1}{n+1}\sum_{i=0}^{n} 2(n - i + 2) = 2(n + 2) - n = 3n + 4, n \geq 0$$

---

practice33

1. 略
2. 如果是像以下的方式交换最外层两个 for 循环，则只要保证 p 的行和 m 的列相等即可

```
for (int j = 0; j < p; j++)
    for (int i = 0; i < m; i++)
```

---

practice34

1.

前提：swap = 2 次移动元素，这里忽略使用 t 的过程

排序	次数	举例
selectionSort	$2(n-1)$	[n,1,2,3,...n-1]
insertionSort	$\sum_{i=1}^{n-1}(\sum_{j=0}^{i-1} 1 + 1) = \frac{(n+1)\times(n-1)}{2}$	[n,n-1,n-2,...1]
bubbleSort	$\sum_{i=2}^{n}(\sum_{i=0}^{n-2} 2) = 2(n-1)^2$	[n,n-1,n-2,...1]

2. 见附录

---

practice35

同一个程序在最坏的情况下，所需时间和内存一定最大
证明：

1. 如果该程序的时间与内存和实例特征有关，则最坏情况下，比较、移动的次数增加，递归深度增加，导致最终的时间和内存增加
2. 如果时间与内存和实例特征无关，那么不存在最好、最坏情况，或者说最好、最坏情况下时间、内存都是相同的，即时间、内存都是最小或最大的

---

practice36

1.

$$
\begin{aligned}
t(n) &= 2 + t(n-1) \\
&= 4 + t(n-2) \\
&\vdots \\
&= 2n + t(0) \\
&= 2n + 2, n \geq 0
\end{aligned}
$$

2.

$$
\begin{aligned}
t(n) &= 1 + t(n-2) \\
&= 2 + t(n-4) \\
&\vdots \\
&= \begin{cases} \frac{n-1}{2} + 1 & mod(n,2) == 1 \\ \frac{n}{2} & mod(n,2) == 0 \end{cases}
\end{aligned}
$$

3.

$$
\begin{aligned}
t(n) &= 2n + t(n-1) \\
&= 2(n + (n-1)) + t(n-2) \\
&\vdots \\
&= 2(n + (n-1) + \cdots + 1) + t(0) \\
&= \frac{n(n+1)}{2}, n \geq 0
\end{aligned}
$$

4.

$$
\begin{aligned}
t(n) &= 2 * t(n-1) \\
&= 4 * t(n-2) \\
&\vdots \\
&= 2^n * t(0) \\
&= 2^n, n \geq 0
\end{aligned}
$$

5.

$$
\begin{aligned}
t(n) &= 3 * t(n-1) \\
&= 9 * t(n-2) \\
&\vdots \\
&= 3^n * t(0) \\
&= 3^n, n \geq 0
\end{aligned}
$$

附录

原地重排的计数排序

```
template<class T>
void rank(T a[], int n, int r[])
{
    for (int i = 0; i < n; i++)
        r[i] = 0;

    for (int i = 1; i < n; i++)
        for (int j = 0; j < i; j++)
            if (a[j] <= a[i]) r[i]++;
            else r[j]++;
}

template<class T>
void rearrange(T a[], int n, int r[])
{
    for (int i = 0; i < n; i++)
        while (i != r[i])
        {
            int t = r[i];
            swap(a[i], a[t]);
            swap(r[i], r[t]);
        }
}
```

及时终止的选择排序

```
template<typename T>
void selectionSort(T a[], int n)
{
    bool sorted = false;
    for (int size = n; !sorted && size > 1; size--)
    {
        int indexOfMax = 0;
        sorted = true; // 假设数组有序
        for (int i = 1; i < size; i++)
            // 如果当前元素比前部分的 max 还要大，则到该元素为止，都是有序的
            if (a[i] > a[indexOfMax]) indexOfMax = i;
                // 当出现 a[i] < a[indexOfMax] 时，数组无序部分出现
            else sorted = false;
        swap(a[indexOfMax], a[size - 1]);
```

```
    }
}
```

及时终止的冒泡排序

```
template<typename T>
bool bubble(T a[], int n)
{
    bool sorted = true;
    for (int i = 0; i < n - 1; i++)
        if (a[i] > a[i + 1])
        {
            swap(a[i], a[i + 1]);
            sorted = false;
        }

    return sorted;
}

template<typename T>
void bubbleSort(T a[], int n)
{
    for (int i = n; i > 1 && !bubble(a, i); i--) {}
}
```

插入排序

```
template<typename T>
void insert(T a[], int n, const T &x)
{
    int i;
    for (i = n - 1; i >= 0 && a[i] > x; i--)
        a[i + 1] = a[i];
    a[i + 1] = x;
}

template<typename T>
void insertionSort(T a[], int n)
{
    for (int i = 1; i < n; i++)
    {
        T t = a[i]; // 因为 insert 中对 x 进行了引用，如果直接传递 a[i]，会导致在
a[i+1]=a[i] 的时候修改掉 x 的值，导致数组的值被改变
        insert(a, i, t);
```

```
    }
}
```