30240184，2010 年 4 月 9 日　　姓名：　　学号：

题号	1	2	3	4	5	∑
得分						

1. 正误判断（凡交待未尽之处，皆以讲义及示例代码为准）　2 ×8

1) （　　）$f(n) = \mathcal{O}(g(n))$，当且仅当$g(n) = \Omega(f(n))$。

2) （　　）若借助二分查找确定每个元素的插入位置，向量的插入排序只需$\mathcal{O}(n\log n)$时间。

3) （　　）无论有序向量或有序列表，最坏情况下均可在$\mathcal{O}(\log n)$时间内完成一次查找。

4) （　　）对有序向量做Fibonacci查找，就最坏情况而言，成功查找所需的比较次数与失败查找相等。

5) （　　）只要是采用基于比较的排序算法，对任何输入序列都至少需要运行$\Omega(n\log n)$时间。

6) （　　）对不含括号的中缀表达式求值时，操作符栈的容量可以固定为某一常数。

7) （　　）对于同一有序向量，每次折半查找绝不会慢于顺序查找。

8) （　　）RPN中各操作数的相对次序，与原中缀表达式完全一致。

2. 多重选择　3 ×5

1) （　　）若$f(n) = \mathcal{O}(n^2)$且$g(n) = \mathcal{O}(n)$，则以下结论正确的是：

A. $f(n) + g(n) = O(n^2)$　B. $f(n)/g(n) = O(n)$　C. $g(n) = O(f(n))$　D. $f(n)*g(n) = O(n^3)$

2) （　　）算法$g(n)$的复杂度为$\Theta(n)$。若算法$f(n)$中有 5 条调用$g(n)$的指令，则$f(n)$的复杂度为：

A. $\Theta(n)$　B. $\mathcal{O}(n)$　C. $\Omega(n)$　D.不确定

3) （　　）共有几种栈混洗方案，可使字符序列{'x', 'o', 'o', 'o', 'x'}的输出保持原样？

A. 12　B. 10　C. 6　D. 5

4) （　　）对长度为$Fib(12) - 1 = 143$ 的有序向量做Fibonacci查找，比较操作的次数至多为：

A. 12　B. 11　C. 10　D. 9

5) （　　）对长度为$n = Fib(k) - 1$ 的有序向量做Fibonacci查找。若各元素的数值等概率独立均匀分布，且平均成功查找长度为L，则平均失败查找长度为：

A. n(L-1)/(n-1)　B. n(L+1)/(n+1)　C. (n-1)L/n　D. (n+1)L/n

3. 给出函数F(n)复杂度的紧界（假定int字长无限，移位属基本操作，且递归不会溢出）　3 ×8

`void F(int n)　//O(　　)` `{ for (int i=1,r=1; i<n; i<<=r,r<<=1); }`	`void F(int n)　//O(　　)` `{　for (int i=0, j=0; i<n; i+=j, j++); }`
`void F(int n)　//O(　　)` `{ for (int i=1; i<n; i = 1<<i); }`	`int F(int n)　//O(　　)` `{ return (n<4) ? n : F(n>>1) + F(n>>2); }`

`int F(int n)　　//O(　　)`
`{　return (n == 0) ? 1 : G(2, F(n-1)); }`
`int G(int n, int m)`
`{ return (m == 0) ? 0 : n + G(n,m-1); }`
	`int F(int n)　　//O(　　)`
	`{　return G(G(n-1)); }`
	`int G(int n)`
	`{ return (n==0) ? 0 : G(n-1) + 2*n - 1; }`

`void F(int n) {　　//O(　　)`
`　for (int i=1; i<n; i++)`
`　for (int j=0; j<n; j+=i);`
`}`
	`void F(int n) {　　//expected-O(　　)`
	`　for (int i=n; 0<i; i--)`
	`　　if (0 == rand()%i)`
	`　　for (int j=0; j<n; j++);`
	`}`

4. 分析与计算　　5 ×5

1) 考查如下问题：任给 12 个互异的整数，且其中 10 个已组织为一个有序序列，现需要插入剩余的两个以完成整体排序。若采用基于比较的算法（CBA），最坏情况下至少需做几次比较？为什么？

2) 向量的插入排序由n次迭代完成，逐次插入各元素。为插入第k个元素，最坏情况需做k次移动，最好时则无需移动。从期望的角度来看，无需移动操作的迭代平均有多少次？为什么？
假定各元素是等概率独立均匀分布的。

3) 现有长度为 15 的有序向量A[0..14]，各元素被成功查找的概率如下：

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
P_i (Σ=1)	1/128	1/128	1/32	1/8	1/8	1/32	1/16	1/16	1/128	1/64	1/16	1/4	3/16	1/128	1/64

若采用二分查找算法，试计算该结构的平均成功查找长度。

4) 考查表达式求值算法。算法执行过程中的某时刻，若操作符栈中的括号多达 2010 个，则此时栈的规模（含栈底的'\0'）至多可能多大？试说明理由，并示意性地画出当时栈中的内容。

5) 阅读以下程序，试给出其中ListReport()一句的输出结果（即当时序列L中各元素的数值）

```
#define  LLIST_ELEM_TYPE_INT  //节点数据域为int型
LValueType  visit(LValueType e) {
   static int lemda = 1980;
   lemda += e*e;
   return lemda;
}
```

```
int   main(int argc, char* argv[]) {
   LList* L = ListInit(-1);
   for (int i=0; i<5; i++)
      ListInsertFirst(L, i);
   ListTraverse(L, visit);
   ListReport(L);     /*输出：                                        */
   ListDestroy(L);
   return 0;
}
```

5. 基于ADT操作实现算法（如有必要，可增加子函数）　　10 ×2

1) sortOddEven(L)

```
#define  LLIST_TYPE_ARRAY  //基于向量实现序列
#define  LLIST_ELEM_TYPE_INT  //节点数据域为int型
/*************************************************************************
 * 输入：基于向量实现的序列L
 * 功能：移动L中元素，使得所有奇数集中于前端，所有偶数都集中于后端
 * 输出：无
 * 实例：L = {2, 13, 7, 4, 6, 3, 7, 12, 9}，则排序后
 *       L = {9, 13, 7, 7, 3, 6, 4, 12, 2}
 * 要求：O(n)时间，O(1)附加空间
*************************************************************************/
void  sortOddEven(LList* L) {




}
```

2) shift(L, k)

```
#define  LLIST_TYPE_ARRAY  //基于向量实现序列
#define  LLIST_ELEM_TYPE_INT  //节点数据域为int型
/*************************************************************************
 * 输入：基于向量实现的序列L
 * 功能：将L中各元素循环左移k位
 * 输出：无
 * 实例：L = {1, ..., k, k+1, ..., n}，则左移后
 *              L = {k+1, ..., n, 1, ..., k}
 * 要求：𝑂(n)时间（注意：最坏情况下k=Ω(n)），𝑂(1)附加空间
 *************************************************************************/
void  shift(LList* L, int k) {  //Assert: L != NULL, 0 < k < Length(L)




}
```

```
// ********************************************************
// Queue.h
// ********************************************************
Queue*       QueueInit(int id); //创建队列
Status       QueueDestroy(Queue* Q); //销毁队列
bool         QueueEmpty(Queue* Q); //判断是否队空
int          QueueSize(Queue* Q); //获取队列的当前规模
QValueType   Front(Queue* Q);   //获取队首节点
QValueType   Enqueue(Queue* Q, QValueType e); //入队
QValueType   Dequeue(Queue* Q); //出队
/* 此处忽略其余ADT操作 */
// End of Queue.h
```

```
// ********************************************************
// RPN.h
// ********************************************************
   const char pri[N_OPTR][N_OPTR] = { // 运算符优先等级 [栈顶] [当前]
   //  |------------------------ 当前运算符 ----------------------|
   //  +     -     *     /     ^     !     (     )     \0
      '>',  '>',  '<',  '<',  '<',  '<',  '<',  '>',  '>'   // +     --
   ,  '>',  '>',  '<',  '<',  '<',  '<',  '<',  '>',  '>'   // -     |
   ,  '>',  '>',  '>',  '>',  '<',  '<',  '<',  '>',  '>'   // *     栈
   ,  '>',  '>',  '>',  '>',  '<',  '<',  '<',  '>',  '>'   // /     顶
   ,  '>',  '>',  '>',  '>',  '>',  '<',  '<',  '>',  '>'   // ^     运
   ,  '>',  '>',  '>',  '>',  '>',  '>',  '<',  '>',  '>'   // !     算
   ,  '<',  '<',  '<',  '<',  '<',  '<',  '<',  '=',  ' '   // (     符
   ,  '>',  '>',  '>',  '>',  '>',  '>',  ' ',  '>',  '>'   // )     |
   ,  '<',  '<',  '<',  '<',  '<',  '<',  '<',  ' ',  '='   // \0    --
};
```

n	0	1	2	3	4	5	6	7	8	9	10	11	12
Fib(n)	0	1	1	2	3	5	8	13	21	34	55	89	144
Fib(n) - 1		0	0	1	2	4	7	12	20	33	54	88	143
2^n	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
2^n - 1	0	1	3	7	15	31	63	127	255	511	1023	2047	4095

反面还有

```
// ********************************************************
// LinearList.h
// ********************************************************
LList*       ListInit(int id); //创建序列
Status       ListDestroy(LList* L); //销毁序列
bool         ListEmpty(LList* L); //判断序列是否为空
int          ListLength(LList* L); //获取序列的当前规模
Position     ListFirst(LList* L); //获取序列的首节点（若无返回NULL）
Position     ListLast(LList* L); //获取序列的末节点（若无返回NULL）
Position     ListPrev(Position v); //获取前驱（若无返回NULL）
Position     ListNext(Position v); //获取后继（若无返回NULL）
LValueType   ListGetValue(Position p); //获取节点的数据域
LValueType   ListGetValue(LList* L, Rank i); //获取节点的数据域
LValueType   ListSetValue(LList*  L, Rank  i, LValueType  e); //设置节点的数据域
LValueType   ListSetValue(Position   p, LValueType  e); //设置节点的数据域
Position     ListInsert(LList* L, Rank i, LValueType e); //插入节点
Position     ListInsertFirst(LList* L, LValueType e); //作为首节点插入
Position     ListInsertLast(LList* L, LValueType e); //作为末节点插入
LValueType   ListDelete(LList* L, Position p); //删除节点p，返回其数据域
LValueType   ListDelete(LList* L, Rank i); //删除秩为i的节点，返回其数据域
LValueType   ListDeleteFirst(LList* L); //删除首节点，返回其数据域
LValueType   ListDeleteLast(LList* L); //删除末节点，返回其数据域
void         ListTraverse(LList* L, LValueType (*visit)(LValueType)); //遍历
/* 此处忽略其余ADT操作 */
// End of LinearList.h
```

```
// ********************************************************
// Stack.h
// ********************************************************
Stack*       StackInit(int id); //创建栈
Status       StackDestroy(Stack* S); //销毁栈
bool         StackEmpty(Stack* S); //判断是否栈空
int          StackSize(Stack* S); //获取栈的当前规模
SValueType   Top(Stack* S); //获取栈顶节点
SValueType   Push(Stack* S, SValueType e); //入栈
SValueType   Pop(Stack* S); //出栈
/* 此处忽略其余ADT操作 */
// End of Stack.h
```

反面还有