考题知识点索引：

1. 时间复杂度分析P27-48
2. 二分查找P172、Fib查找P184
3. 排序：选择P255、插入P270、归并P277
4. 栈混洗：P337
5. RPN：P345

第1题 **正误判断（凡交代未尽之处，皆以讲义及示例代码为准）**

1. （ 1 ）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。注：Fibonacci查找是对普通二分查找的一个优化（3比较），目的是减少成功比较与失败比较的次数。

2.（ 1 ），当且仅当。注：上界、下界、恰好的时间复杂度

3.（ 0 ）若借助二分法查找确定每个元素的插入位置，向量的插入排序只需时间时间。注：单次查找是logN，但单次插入则不一定（最好O(1)，最坏O(N)），故插入排序还是O(N^2)的

4.（ 1 ）RPN中各操作数的相对次序，与原中缀表达式完全一致。

5.（ 1 ）对不含括号的中缀表达式求值时，操作法栈的容量可以固定为某一常数。注：运算符有多少级别种类，操作栈就可以只开这么大（例：+，-，\*，/，乘方，共三种不同的运算级别）

6.（ 0 ）无论有序向量或有序列表，最坏情况下均可在时间内完成一次查找。注：列表最坏情况下是O（N）

7.（ 0 ）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时间。注：插入排序对原本就有序的序列排序，时间是O(N)（注意题目中所说的是“对任何输入序列”）

8.（ 0 ）对于同一有序向量，每次折半查找绝不会慢于顺序查找。注：每次找第一个，顺序O(1)，折半O(logN)

第2题 **多重选择**

1.（ C ）共有几种栈混洗方案，可以使字符序列{‘x’,’o’,’o’,’o’,’x’}的输出保持原样？

A．12 B. 10 C. 6 D. 5

2．（ AD ）

若，则下列结论正确的是：注：O为上界，没有明确究竟是多少；故不能使用除法

A. B. C. D.

3．（ B ）对长度为的有向序列做Fibonacci查找。若个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，则失败平均查找长度为：（举例子or习题解析P46）

注：P184：~~平均成功查找长度L=k-2，平均失败查找长度n(k-1)/(n+1) = n(L+1)/(n+1)（这是错的）~~，正确的在后面附图；比较次数至多为k-1.

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

**4．**（ B ）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．（ D？ ）算法g(n)的复杂度为。若算法f(n)中有5条调用g(n)的指令，则f(n)的复杂度为：注：没有说明f算法是否“只”包含g指令

A． B.  C.  D. 不确定

第3题 **估计以下函数F(n)的复杂度（假定int类型字长无限，且递归不会溢出）**

|  |  |
| --- | --- |
| void F(int n) //O( sqrt(n) )  {  for (int i = 0, j = 0; i<n; i+=j, j++);  } | void F(int n) //O(loglogn )  { //同理第一题：改+1为乘2  for (int i = 1, r = 1; i<n; i<<=r, r<<=1);  } |
| void F(int n) //O( nlog(n))  { // 调和级数  for (int i=1; i<n; i++)  for (int j=0; j<n; j+=i);  } | void F(int n) //expected-O(n)  {// 级数求和（注意这里不能只是Log^2n，因为最外层循环已经占了O(n)）  for (int i=1; i<n; i++)  if(0 == rand()%i)  for (int j=1; j<n; j<<=1);  } |
| void F(int n) //O(log\*n )  { // 讲义原题  for (int i=1; i<n; i=1<<i);  } | void F(int n) //O(1.618^(logn)) = O(n^0.694)  { // 转乘法为加法，fibonacii  return (n<4) ? n : F(n>>1)+F(n>>2);  } |
| int F(int n) //O(2^n)  { // 再回头看F，F = G(2, G(2, G(2, …, G(2, 1)))  return (n==0) ? 1 : G(2, F(n-1));  }  int G(int n, int m)  { // 先分析G, G = O(m), G(n, m) = n \* m  return (m==0) ? 0 : n+G(n, m-1);  } | int F(int n) //O(n^2)  {// 再分析F，G((n-1)\*(n-1)) = O((n-1)^2)  return G(G(n-1));  }  int G(int n)  {// 先分析G，G=O(n), G(n) = n\*n  return (n==0) ? 0 : G(n-1)+2\*n-1;  } |

第4题 **分析与计算**

1. 考察如下问题：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已完成整体排序。若采用基于比较的算法（CBA），最坏情况下至少需要做几次比较？为什么？

答：8次。 我们知道对于CBA,我们可以将其涵盖于一棵比较树里边，而树的每一个节点可以代表一次比较运算，树的分支可以代表算法下一步执行的方向。由此可以推算，树高则可以代表比较的次数。

注：类似排序复杂度分析：总情况数为11\*12 = 132. 故二叉树至少要有132个儿子，高度至少为8.（2^8 = 256 > 132）

1. 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，最坏情况需要做k次移动，最好情况则无需移动。从期望的角度来看，无需移动操作的迭代次数平均有多少次？为什么？

假定个元素是等概率独立均匀分布的。

答：logn。调和级数

注：总次数 = 1 + 1/2 + 1/3 + 1.4 + … + 1/n = n log n, 平均为 log n.

1. **现有一长度为15的有序向量A[0…14]，个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 |
|  | 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。

答：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Length | 5 | 4 | 6 | 3 | 6 | 5 | 7 | 2 | 6 | 5 | 7 | 4 | 7 | 6 | 8 |

平均查找长度= 5/128 + 4/128 + 6/32 + 3/8 + 6/8 + 5/32 + 7/16 + 2/16 + 6/128 + 5/64 + 7/16 + 4/4 + 7/16\*3 + 6/128 + 8/64 = 659/128

1. 考察表达式求值算法。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’\n’）至多可能多达？试说明理由，并示范性地画出当时栈中的内容。

答：4 \* 2010 + 1 + 4 = 8045(考虑乘方操作后)、（注意+1，有阶乘操作！）

栈中内容：\0 + \* ^ ( + \* ^ ( + \* ^ … ( + \* ^ !

1. 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中个元素的数值）

#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++)

ListinsertLast(L, i);

ListTraverse(L, visit);

ListReport(L); /\*输出： \*/

ListDestroy(L);

return 0;

}

1980 1981 1985 1994 2010

（具体实现不明，可能为）

1980 1981 1985 1994 2010

（可能为）

1 2 3 4 5

第5题 **基于ADT操作实现算法（如有必要，可增加子函数）**

1、sortOddEvev(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}，则排列序后

\* L = {9，13，7，7，3，6，4，12，2}

\*要求：O(n)时间，**O(1)附加空间**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void sortOddEvev(LList\* L){

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

}

flag1 = 0；

flag2 = n-1;

for (int i=flag1; i<=flag2; i++)

{

if (i%2==0)

{

flag1 = i;

for (int j=flag2; j>flag1; j--)

{

if (j%2==1)

{

flag2 = j;

break；

}

}

//swap(L[flag1],L[flag2]);

}

}

// 思路：从前往后找到第一个偶数A[i]、从后往前找到第一个奇数A[j]，如果i<j，则交换两数，并继续找；否则结束。

2、shift(L,K)

#define LLiST\_TYPE\_ARRAY

#definr LLiST\_ELEM\_TYPE\_iNT

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*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=（n）），**O(1)附加空间**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void shift(LList\* L, int k) { // Assert: L != NULL. 0 < k < Lenth(L)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

}

// 书上例题

