中间几个小问不是很全

1. 是非判断题【20 分，每题 2 分；正确：√，错误：x】

1) 算法的时间效率只与问题规模有关。（ x ）
2) 算法的空间效率是算法执行时需要的内存空间总数。（ x ）
3) 从逻辑关系存储表达上考虑，静态链表是链接存储结构。（ √ ）
4) 顺序表是随机存储结构，因此其检索效率高于链表。（ x ）
5) 线性表的双链表存储是牺牲空间代价，换取找结点前驱运算的方便。（ √ ）
6) 栈和队列是操作受限的线性表，因此其插入、删除运算的效率低。（ x ）
7) 字符串的单字符结点链接存储结构中，存储密度不超过 25%。（ √ ）
8) 递归算法的空间效率分析，是以递归层数进行衡量的。（ √ ）
9) 与广度优先搜索相比，深度优先搜索算法能够更容易求得问题的最优解。（ x ）
10) 字符串的单字符模式匹配中，经典模式匹配的效率不低于 KMP 算法。（ √ ）

2. 填空题【60 分，无特别声明每空 2 分】

1) 四种常用的逻辑关系包括：__集合__、线性、__层状__和__网状__。四种常用的存储结构包括：顺序、链接、__索引__和__散列__。

2) 已知 p 指向某双向链表的某个结点，完成下面的函数，实现 p 与其后面结点交换。

结点结构为：

left	data	right

```
void swap(DoubleNode *p)
{   DoubleNode *q = p->right;
    if (q == NULL) return;
    else
    {   p->right = q->right;              //删除 p 右边的结点
        q->right->left = p ;
        p->left->right = q ;              //把 q 插入到 p 的左边
        q->left = p->left ;
```

3) 完成下列字符串的顺序存储结构下的经典模式匹配算法：

```
#define MAXNUM 1000
typedef struct
{   char c[MAXNUM];   int n; } *pString;   //c 是字符数组，n 是串长。
int  LocSubStr ( pString s,  pString r )   //在目标 s 中匹配模式 r。
{   int i, j, k;
    for ( i = 0;  s->c[i]          ;  i++)
    {   for ( j = i,   k = 0;  s->c[j] == r->c[k] ;   j++,   k++)
        {   if (  !r->c[k]          ) return i;     }
    }
```

5) 对于结点为 PNode {datatype data; PNode link}类型【结构指针类型】，栈顶指针为 top 的链栈 pStack。

压入结点 p 的操作为：__p->link = pStack->top__ ; __pStack->top = p__。

弹出结点的操作为： q=__pStack->top__ ; __pStack->top = pStack->top->link__ ; free(q)。

6) 对于结构为{Datatype q[MAXNUM]; int f, r;}类型【结构指针类型，f、r 为队头和对尾，MAXNUM 为最多元素个数】的循环队列 pQueue，采用少用一个元素空间的方法区别空和满。

入队列的操作为：pQueue->r = __(pQueue->r+1)%MAXNUM__

出队列的操作为：pQueue->f = __(pQueue->f+1)%MAXNUM__

空队列的判断条件是：__pQueue->r == pQueue->f__

满队列的判断条件是：__pQueue->r+1)%MAXNUM == pQueue->f__

3. 简单问答题【20 分，答在本页背面】

为了加快图书检索的效率，通常需要构建“关键词—书号”索引表（词索引表）。简单描述词索引表存储表达方式（考虑不同关键词对应的书的数目差别较大，以及易于后续的快速检索等要求，画出存储结构示意图【10 分】）和词索引表的构建过程（不需要写详细算法，但要写出主要步骤【10 分】）。