2021年秋季学期数据与算法第一次作业

2021.10.23

1.分析下列程序的时间复杂度。

(1)
```
int sum(int n)
{
    int s = 0;
        for (int i = 1; i <= n; i++)
        {
            int p = 1;
            for (int j = 1; j <= i; j++)
                p *= j;
            s += p;
        }
    return s;
}
```

(2)
```
int fac(int n)
{
    int p = 1, s = 0;
        for (int i = 1; i <= n; i++)
        {
            p *= i;
            s += p;
        }
    return s;
}
```

(3)
```
int fun(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i = 2 * i)
    {
        int p = 1;
        for (int j = 1; j <= i; j = j++)
        {
            p = p * j;
            sum += p;
        }
    }
    return sum;
}
```

(4)
```
void write(int n)
```

```
{
    if (n != 0)
    {
        write(n - 1);
        cout << n << endl;
    }
    return;
}
```

2. 设数据元素的集合为D={a1,a2,a3,a4,a5,a6}，请分别画出与以下各关系R 对应的数据结构B=(D,R)的结构示意图，并指出它属于哪类结构。

(1) $R=\{(a_3,a_4),(a_4,a_5),(a_1,a_2),(a_2,a_3),(a_5,a_6)\}$

(2) $R=\{(a_3,a_2),(a_2,a_4),(a_3,a_1),(a_2,a_5),(a_2,a_6)\}$

(3) $R=\{(a_{i+1},a_i) \mid i=5,4,3,2,1\}$

(4) $R=\{(a_i,a_j) \mid i>j\}$

(5) R={ }

3. 算法填空。下列给出了将两个有序的线性链表合并为一个有序链表，合并后使原链表为空的函数实现，请补全其中缺失的关键步骤。

注：HA 和 HB 为非递减有序线性链表，合并为非递减有序线性链表 HC

结点数据类型为

```
struct SNode{
    int data;
    SNode *next;
    SNode():data(0),next(NULL){}
};
```

HA.head 是第一个结点

(1) 不带表头结点

```
void MergeList(LinkList& HA, LinkList& HB, LinkList& HC)
{
    //程序以HA链表作为HC的初始链表，通过不断插入HB链表中的元素实现合并
    InitList(HC); //链表HC初始化
    SNode* r, * p = ①, * q = ②;
    HC.head = p;
    if (!q)  //链表HB是空表
    {
        HA.head = NULL;
        return;
    }

    if (!p)   //链表HA是空表
    {
        HC.head = ③;
```

```
        HB.head = NULL;
        return;
    }
    if (p->data <= q->data) //HA链和HB链头节点情况判断
    {
        r = p;
        p = p->next;
    }
    else
    {
        HC.head = q;
        r = ④;
        q = ⑤;
    }
    r->next = ⑥;
    while (p && q)
    {
        if (p->data <= q->data)//HA链未合并的第一个结点的值不大于HB链的第一个未合并结
点的值
        {
            r = p;
            p = p->next;
        }
        else
        {
            r->next = ⑦;
            r = r->next;
            q = ⑧;
            r->next = ⑨;
        }
    }
    if (q) //HB链还有结点
    {
        r->next = ⑩;
    }
    HA.head = NULL;
    HB.head = NULL;
    return;
}
```

(2) 带表头结点

```
void MergeList(LinkList& HA, LinkList& HB, LinkList& HC)
{
    //程序以HA链表作为HC的初始链表，通过不断插入HB链表中的元素实现合并
```

```
    InitList(HC); //链表HC初始化
    SNode* p = ①, * q = ②;
    SNode* r = HC.head;
    r->next = ③;
    while (p && q)
    {
        if (p->data <= q->data)//HA链未合并的第一个结点的值不大于HB链的第一个未合并结
点的值
        {
            r = ④;
            p = p->next;
        }
        else
        {
            r->next = ⑤;
            r = r->next;
            q = ⑥;
            r->next = ⑦;
        }
    }
    if (q) //HB链还有结点
    {
        r->next = ⑧;
    }
    ⑨ = NULL;
    ⑩ = NULL;
    return;
}
```

4.分别实现逆转顺序表和逆转链表的算法，使表中最后一个元素成为第一个元素，倒数第二个元素成为第二个元素，以此类推。(对于链表逆序算法，必须交换结点，而不是结点中的数据)。

(1)

```
class SeqList
{
private:
        int Len;
        int* list;
        SeqList(int n)
        {
            Len = 0;
            list = new int[n]; //分配一个足够大的空间
        }
```

```
public:
        void inverList(SeqList& L)
        {
        //请补充对应函数实现
        }
}
```

(2)

```
void invertLinkList(LinkList& HL)
{
    SNode* p, * q, *r=HL.head;
    if (r == NULL)
        return;
    p = r->next;
    r->next = NULL;
    while (p)
    {
     //LinkList和SNode定义同上一题，链表不带表头
     //请补充对应语句

    }
    HL.head = r;
}
```

5. 如果 3 个元素进栈顺序为 X、Y、Z，试写出所有可能的出栈顺序（可能后来者进栈时，先来者已经出栈）。

6. 在线性链表中查找结点值为 x 的第一个结点，并返回其指针的递归算法。

```
struct NODE {
    ElemType data;
    NODE* next;
    NODE():data(0), next(NULL) {}
};

NODE* list_find(NODE* current, ElemType x)
{
    //请补充对应函数实现
}
```