数据结构与算法第一次书面作业

电子工程系　钟宏涛　201701120

第一题：

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; //循环j结束时p=i!。
        s+=p;
    }
    return s;
}
```

功能：求$\sum_{k=1}^{n} k!$。

复杂度分析：sum 函数执行了$\sum_{k=1}^{n} k=\frac{n(n+1)}{2}$，所以算法的复杂度为 O($\frac{n^2}{2}$)=O($n^2$)。

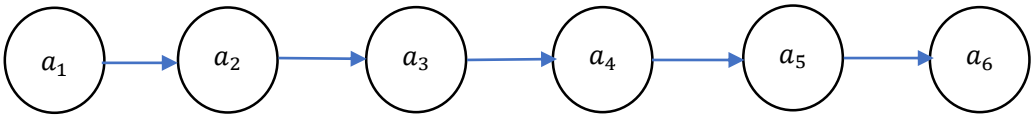
2)
```
int fac(int n)
{   int p=1,s=0;
    for(int i=1; i<=n; i++)
    {    p*=i; //每次执行都得到i!。
        s+=p;
    }
    return s;
}
```

功能：求$\sum_{k=1}^{n} k!$。

复杂度分析：fac 函数执行了 n 次，所以算法复杂度为 O(n)。

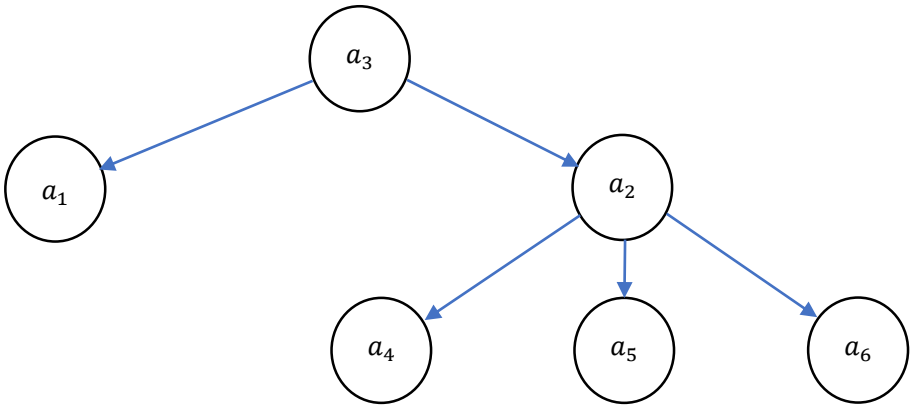
第二题：

1)



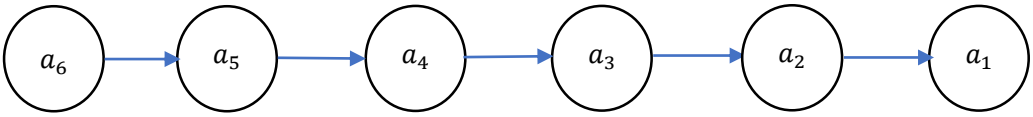
由图可知，这是一个线性结构。

2)



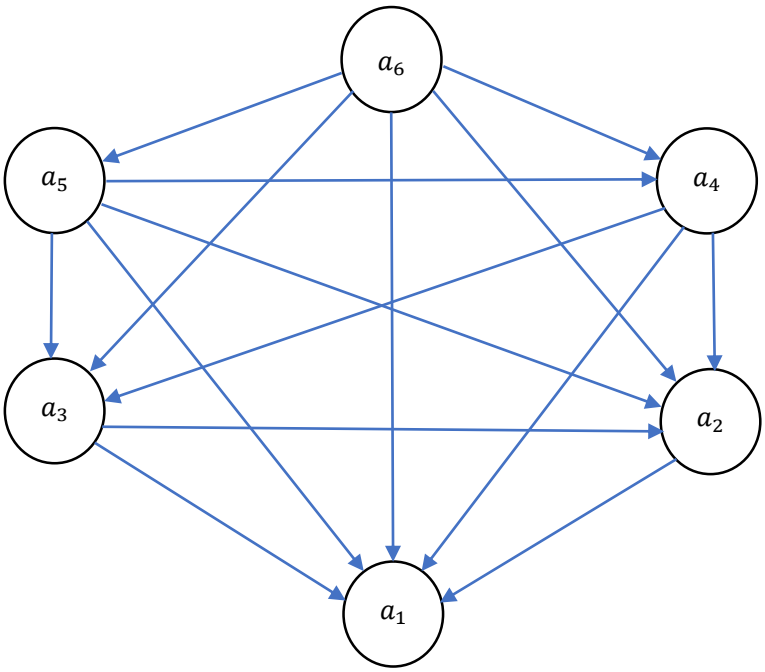
由图可知，这是一个树结构。

3)



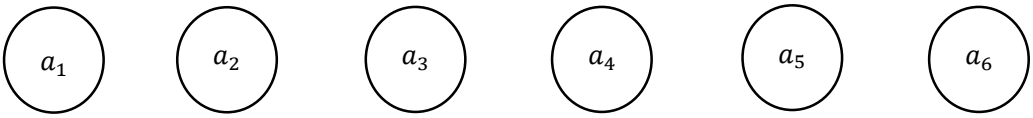
由图可知，这是一个线性结构。

4)



由图可知，这是一个图结构。

5)



由图可知，这是一些孤立的点，因此这是一个集合结构。

第三题：

1) 不带表头

```
#include <iostream>
using namespace std;
struct SNode
{   int data;
    SNode *next;
    SNode():data(0),next(NULL){}
};
void mergeList(LinkList &HA,LinkList &HB,LinkList &HC)
{   int i,j,s,Is_same=0;//Is_same 用来判断是否前后两次插入是否来源于同一个数组。
    SNode *pin_A=HA.head,*pin_B=HB.head,pin_C;
```

```
HC.head=NULL;//初始化 HC。
if(pin_A==NULL&&pin_B==NULL)
   return;
else if(pin_A!=NULL&&pin_B==NULL)
{   HC.head=pin_A;
    HA.head=nullptr;
    return;
}
else if(pin_A==NULL&&pin_B!=NULL)
{   HC.head=pin_B;
    HB.head=nullptr;
    return;
}
else
{   if(pin_A->data<=pin_B->data)
    {   HC.head=pin_A;
        pin_C=HC.head;
        s=1;//这里表头的位置记为 0。
        while(1)
        {   if(pin_C->next==NULL)
            {   pin_C->next=pin_B;
              break;
            }
            else
            {   pin_C=pin_C->next;
                if(pin_C->data>=pin_B->data)
                {   if(ls_same==0)
                    { pin_C.insert(s,pin_B->data);//insert(int position,int e)。第一个表示位置，第二个表示插入节点数据的值。插入位置与数组定义的位置一样。
                      ls_same=1;
                    }
                    else//如果前一个插入也来源于于 HB 链表，那么我们需要比较即将插入属于 HB 的结点的 data 与后一个属于 HA 的结点的的 data。
                    {   if(pin_C->next->data>=pin_B->data)
                            pin_C.insert(s,pin_B->data);
                        else
                            ls_same=0;
                    }
                    if(pin_B->next==NULL)
                      break;
                    else
                      pin_B=pin_B->next;
                    s=s+1;
                }
```

```
                }
            }
        }
        else
        {   HC.head=pin_B;
            pin_C=HC.head;
            s=1;
            while(1)
            {   if(pin_C->next==NULL)
                {   pin_C->next=pin_A;
                  break;
                }
                else
                {   pin_C=pin_C->next;
                    if(pin_C->data>=pin_A->data)
                    {   if(ls_same==0)
                        { pin_C.insert(s,pin_A->data);
                          ls_same=1;
                        }
                        else
                        {   if(pin_C->next->data>=pin_A->data)
                                pin_C.insert(s,pin_A->data);
                             else
                                ls_same=0;
                        }
                        if(pin_A->next==NULL)
                          break;
                        else
                          pin_A=pin_A->next;
                        s=s+1;
                    }
                }
            }
        }
    }
    HA.head=NULL;
    HB.head=NULL;
}
```

2)带表头

```
struct SNode
{   int data;
    SNode *next;
    SNode():data(0),next(NULL){}
};
void mergeList(LinkList &HA,LinkList &HB,LinkList &HC)
{   SNode *pin_A=HA.head,*pin_B=HB.head,pin_C=HC.head;
    if(pin_A->next==NULL&&pin_B->next==NULL)
        return;
    else if(pin_A->next!=NULL&&pin_B->next==NULL)
  {   pin_C->next=pin_A->next;
      pin_A->next=nullptr;
      return;
  }
  else if(pin_A->next==NULL&&pin_B->next!=NULL)
  {   pin_C->next=pin_B->next;
      pin_B->next=nullptr;
      return;
  }
  else
  {   pin_A=pin_A->next;
      pin_B=pin_B->next;
      pin_C->next=pin_A;
      while(pin_A!=NULL)
      {   if(pin_A->data<=pin_B->data)
          {   pin_C=pin_C->next;
              pin_A=pin_A->next;
          }
          else
          {   pin_C->next=pin_B;
              pin_C->next->next=pin_A;
              pin_B=pin_B->next;
              pin_C=pin_C->next;
          }
      }
      if(pin_B!=NULL)
        pin_C->next=pin_B;
      HA.head=NULL;
      HB.head=NULL;
  }
}
```

第四题：

1) 顺序表

```
class SeqList
{   public:
        int length;
        int *list;
    SeqList(int n)
  {   length=0;
        list=new int[n]; //分配一个足够大的空间
  }
};
void inverseSeqList(SeqList &L)
{   int i;
    for(i=0;i<int((L.length)/2);i++)
      L.list[i]=L.list[L.length-i-1]
}
```

2) 链表

```
truct SNode
{   int data;
    SNode *next;
    SNode():data(0),next(NULL){}
};
void inverseLinkList(LinkList &HL)
{   if(HL.head==NULL)
      return;
    else
  {   SNode *p=HL.head,*q=HL.head,*w;
        p=p->next;
        q->next=NULL;
        while(p->next!=NULL)
      {   w=p->next;
          p->next=q;
          q=p;
          p=w;
      }
      HL.head=p;
  }
}
```

第五题：（注：加左括号表示进栈，加右括号表示出栈。）

1) [x][y][z]
2) [x[y]][z]
3) [x][y[z]]
4) [x[y][z]]
5) [x[y[z]]]

第六题：

```
struct NODE
{    ElemType data;
     NODE *next;
     NODE():data(0),next(NULL){}
};
NODE *list_find(NODE *current,ElemType x)
{  NODE *p=head;
   if(p==NULL)
    return NULL;
   else
   { if(p->data==x)
       return p;
       else
          return list_find(p->next,x)
   }
}
```

第七题：

```
void writ(int n)
{    if(n!=0)
     {    writ(n-1);//深层递归，然后从 write(1)往上执行。
          cout<<n<<endl;
          return;
     }
}
```

功能：从 1 到 n 依次输出。

复杂度分析：write 函数执行了 n 次，因此复杂度为 O(n)。