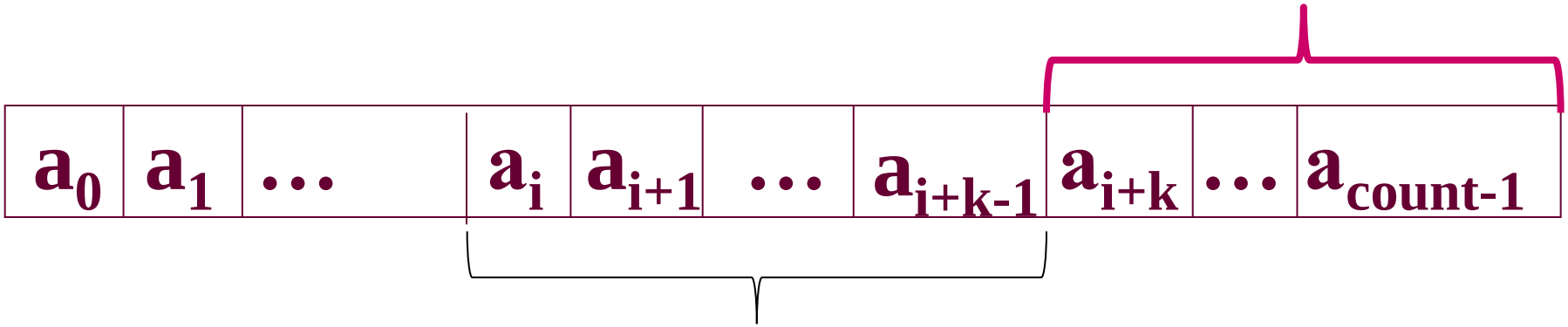
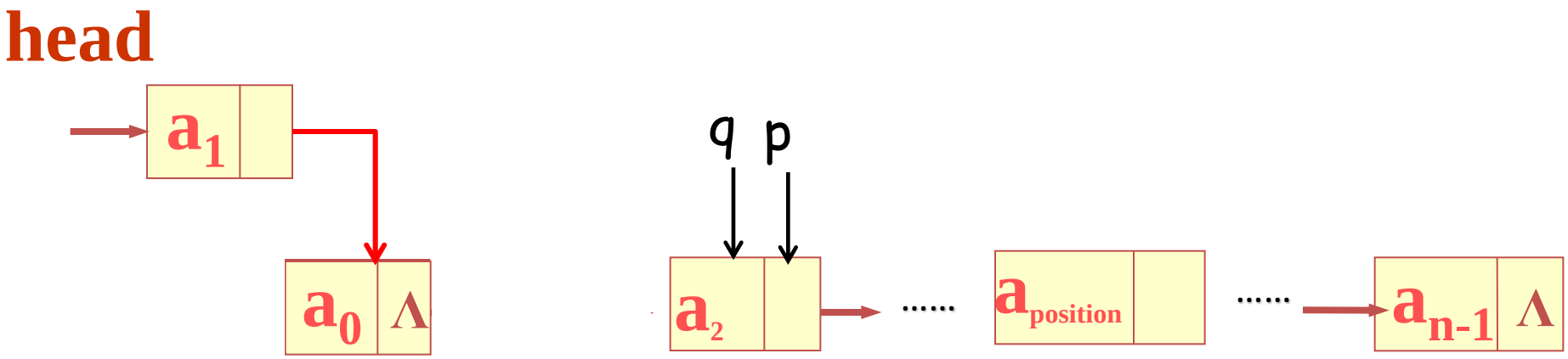
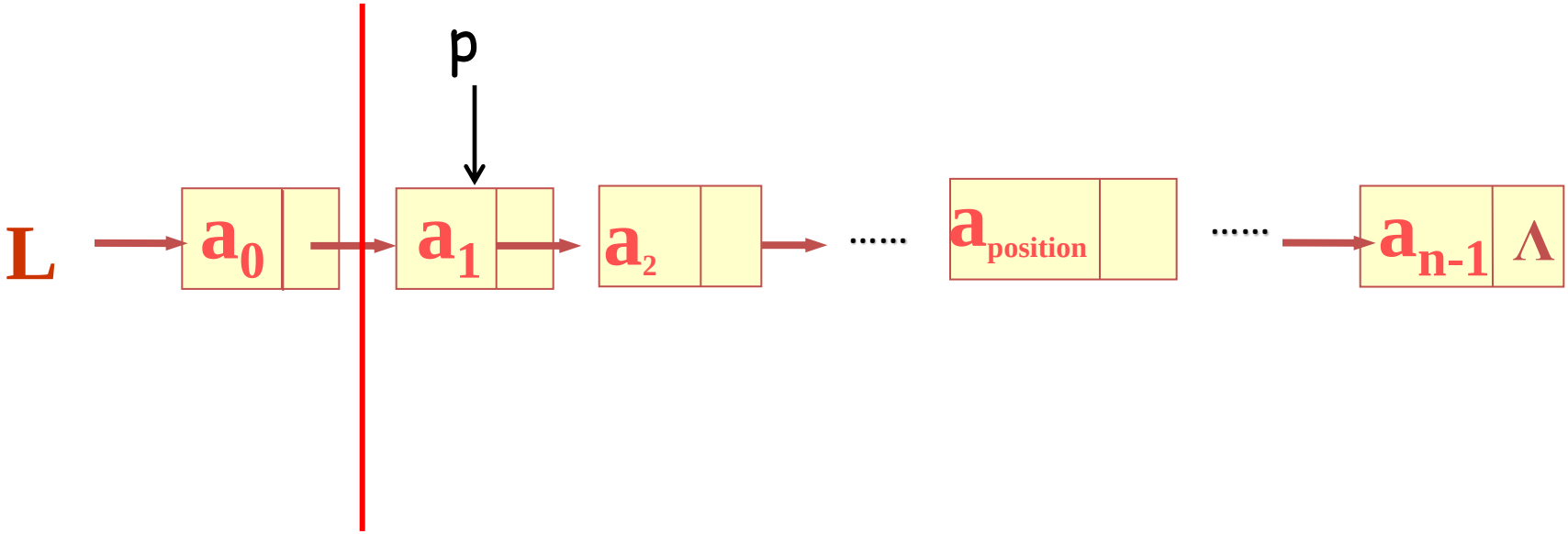
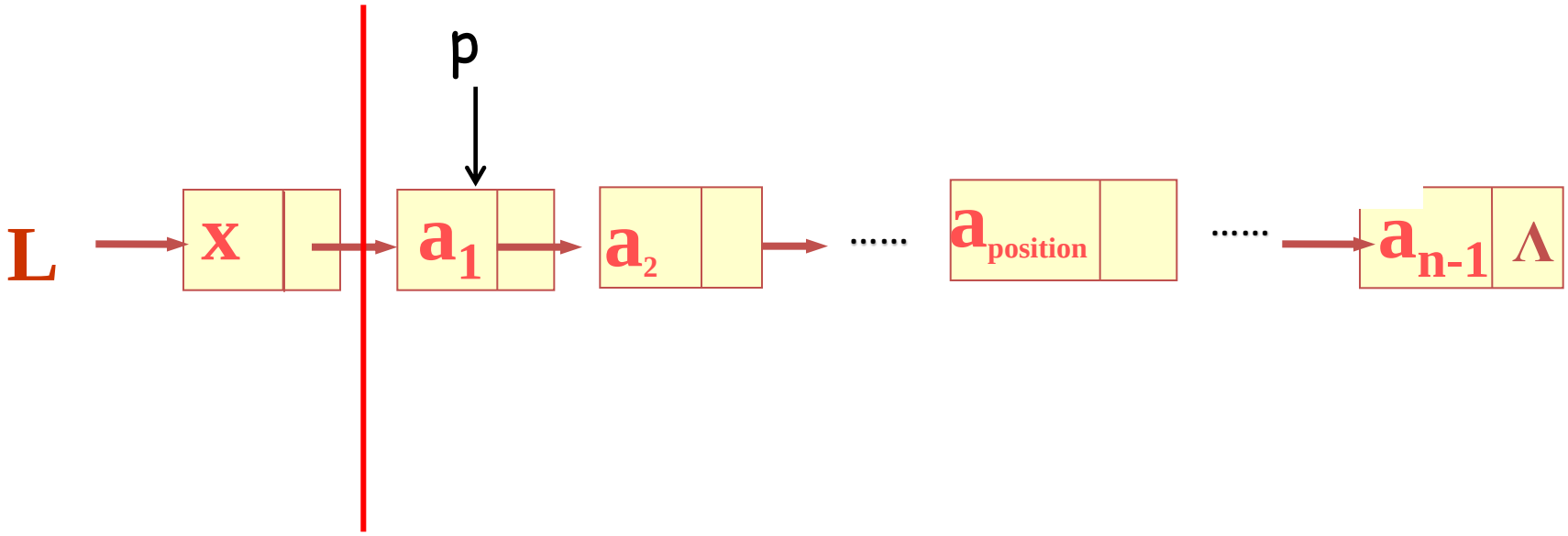
6.4更多算法实现

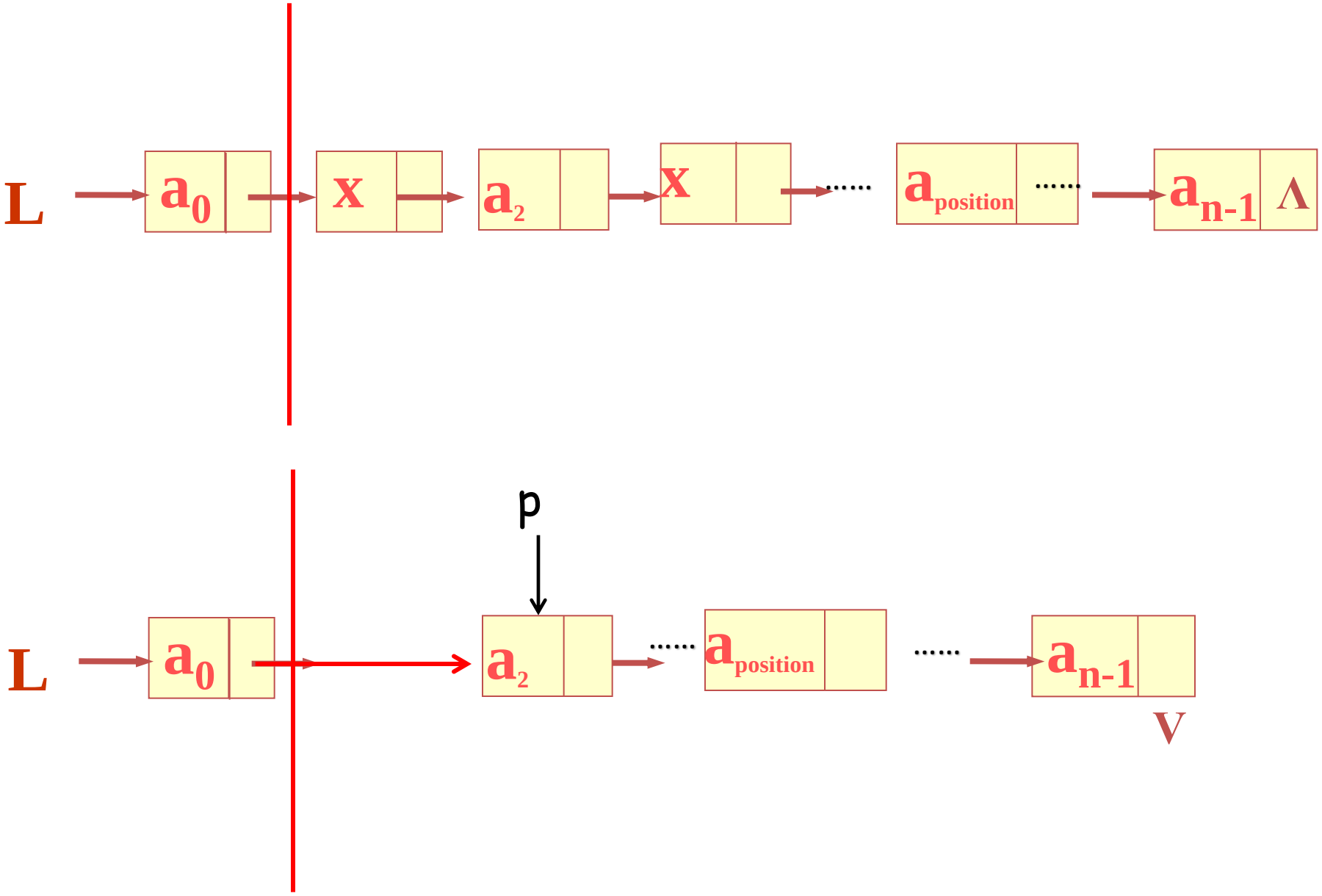

a_0 a_1 … a_i a_{i+1} … a_{i+k-1} a_{i+k} … $a_{count-1}$

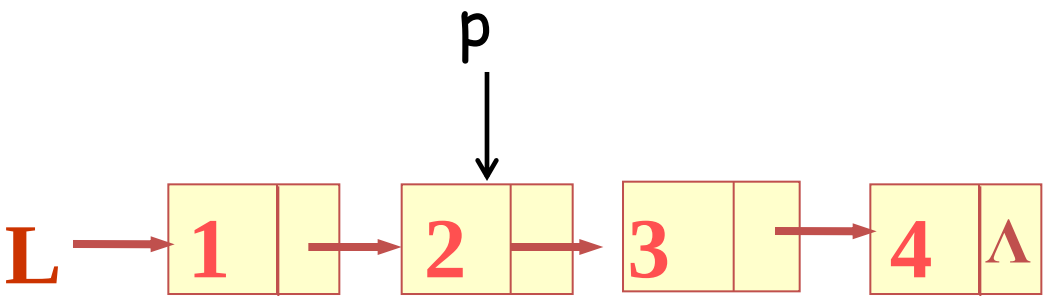


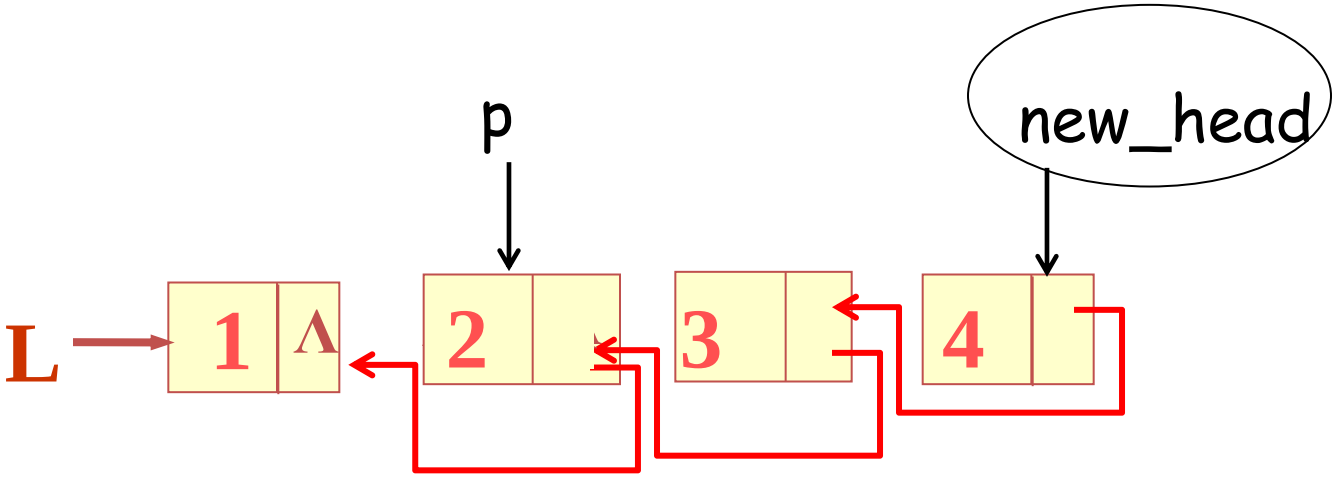

```
p=head->next;
head->next=NULL;
while (p)
{  q=p->next;
   p->next=head;
   head=p;
  p=q;
}
```

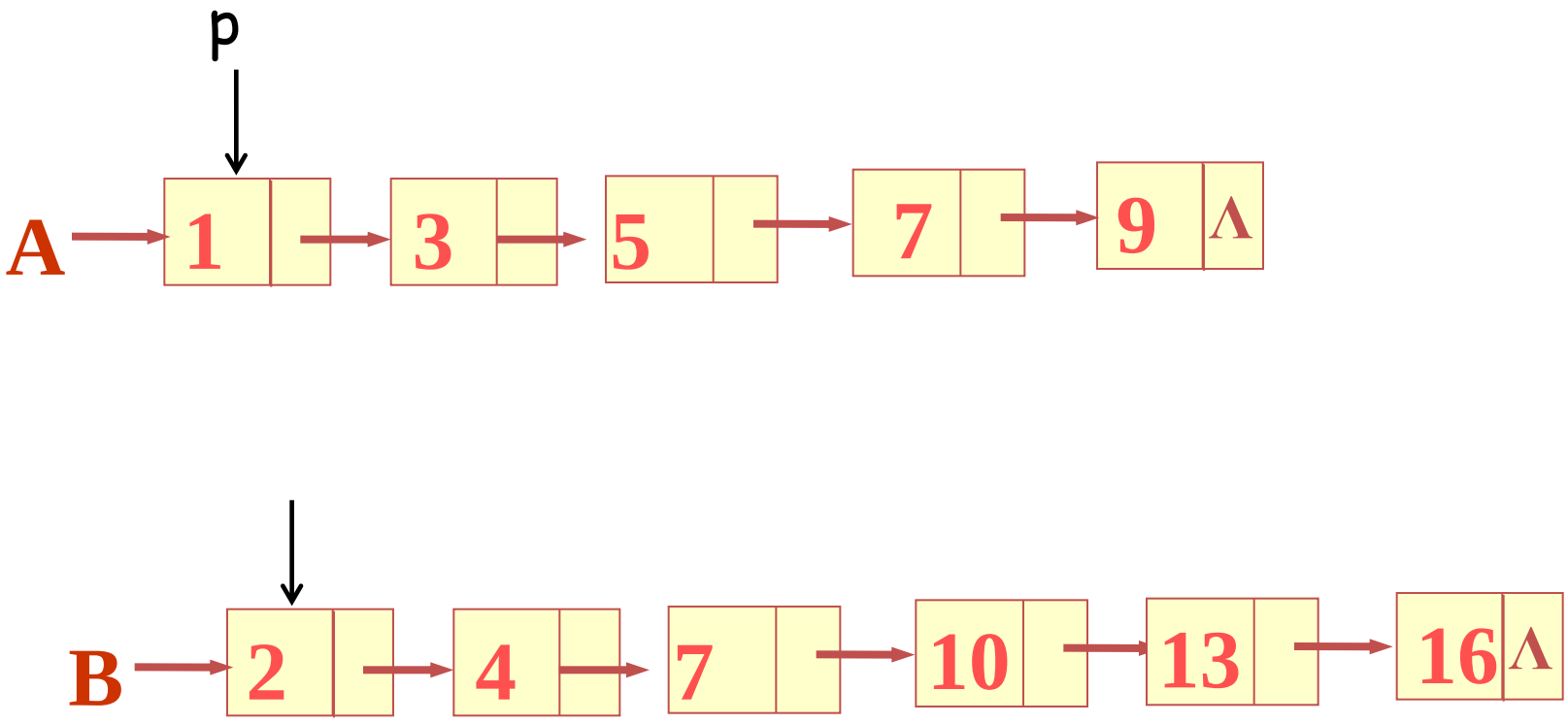

p
L
a_0
a_1
a_2
.....
$a_{position}$
.....
a_{n-1}
Λ


p
L
x
a_1
a_2
.....
$a_{position}$
.....
a_{n-1}
Λ


L
a_0
x
a_2
x
.....
$a_{position}$
.....
a_{n-1}
Λ
L
a_0
p
a_2
.....
$a_{position}$
.....
a_{n-1}
V


p
L
1
2
3
4
Λ


p
new_head
L
1
Λ
2
3
4


p
A
1
3
5
7
9
Λ
B
2
4
7
10
13
16
Λ

1、对不带头结点的单链表，删除其中所有值为x的结点。请设计非递归和递归算法。

2、删除其中重复的结点。

非递归版本

```
template <class List_entry>
Error_code List<List_entry> ::
   removeall( List_entry &x)
{
}
```

递归版本

```
template <class List_entry>
Error_code List<List_entry> ::
   removeall( List_entry &x)
{
   head=recursive_removeall(head,x);
   return success;
}
```

```
template <class List_entry>
Node<List_entry>* List<List_entry> ::recursive_ removeal
(Node<List_entry>  *L, List_entry x)
{//在L为首指针的链表中删除所有值为x的结点，
 //返回删除后生成的新链表的首结点
if (L==NULL) return NULL;
If (L->entry==x)
 {temp=L;p=L->next;delete temp;
return recursive_removeall(p,x);
}
else{
  p=recursive(L->next,x);
L->next=p;
Return L;
}
}
```

```
template <class List_entry>
Node<List_entry>* List<List_entry> ::recursive_ removeall
(Node<List_entry>  *L, List_entry x)
{//在L为首指针的链表中删除所有值为x的结点，
 //返回删除后生成的新链表的首结点
if (L==NULL)
     return NULL;//空表下的删除
else
     if (L->data==x) //首结点需要删除
          {temp=L;
          L=L->next;
          delete(temp);
          return(recursive_ removeall(L,x));}
          else  {
                L->next= recursive_ removeall(L->next,x);
                return L;
                }          }
```

```
template <class List_entry>
void List<List_entry> :: reverse()
{
   head=recursive_reverse(head);
}
```

```
template <class List_entry>
Node<List_entry>*  List<List_entry> ::recursive_
reverse (Node<List_entry>  *head){
If (head==NULL && head->next==NULL)
        return head;
P=head->next;
Temp=recursive(head->next);
p->next=head;
Head->next=NULL;
Return temp;
}
```

```
template <class List_entry>
Node<List_entry>*  List<List_entry> ::recursive_
reverse (Node<List_entry>  *head){
if (head==NULL) return NULL;

p=head->next;head->next=NULL;
if (p==NULL) return head;

new_head=recursive_reverse(p);
p->next=head;
return  new_head;

}
```

对不带头结点的单链表，设计递归算法在指定的位置i上插入x。

```
template <class List_entry>
    Error_code List<List_entry>::insert(int i, const
       List_entry x){
    return recursive_insert(head,i,x);
    }
```

```
template <class List_entry>
Error_code List<List_entry> :: recursive_ insert (Node<List_entry>*    &L, int
    i,List_entry x){
  if ((L==NULL && i>0)|| (i<0))
            return range_error;
  if (i==0){
            Node<List_entry>* newNode=new Node<List_entry>(x,L);
    L= newNode;}
    else{
    recursive_insert(L->next, i-1,x,);
    }
    return success;
 }
```

将一个链表分裂成奇偶两个链表，偶数位序的结点留在原表，奇数位序的形成一个新表。编写递归和非递归算法。

```
template <class List_entry>
    Node<List_entry>  *List<List_entry>::divide(){
    Int i=0;if (head==NULL || head->next==NULL)
        return NULL;
    odd=head->next;
    La=head;lb=odd;p=odd->next;
    while (p){
    If (i==0){
    la->next=p;i=1;la=la->next;
    }
    Else
    {
    lb->next=p;i=0;lb=lb->next;
    }
    p=p->next;

    }
    la->next=NULL;
    lb->next=NULL;
    Return odd;
    }
```

```
template <class List_entry>
    Node<List_entry>  *List<List_entry>::divide(){
    Return Recursive_divide(head);
    }
template <class List_entry>
    Node<List_entry>  *List<List_entry>::Recursive_divide (Node<List_entry>  *
       &head){
    If
    Odd=head->next;
    P=odd->next;
    Head->next=p;
    Odd->next= Recursive_divide(p);
    Reteun odd;
    }
```