1.5 计算序列各程序段的时间复杂度（直接给出结果）。

（1）

|  |
| --- |
| for(i = 0; i < n; i++)  for(j = i; j < n; j++) x++; |

(2)

|  |
| --- |
| i = n;  while(i > 1) i=i/2; |

(3)

|  |
| --- |
| for(i = 1; i <= n; i++)  for(j = 1; j <=n; j++)  for(k = 1; k <= n; k++)  x++; |

(4)

|  |
| --- |
| for(i = 1; i <= n; i++)  for(j = 1; j <=n; j++) x++;  for(k = 1; k <= n; k++) x++; |

(5)

|  |
| --- |
| for(i = 1; i < n; i++)  {  j = i;  while(j < n) j \*= 2；  } |

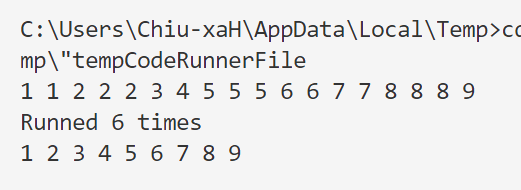
1. **O(n2)**
2. **O(log2n)**
3. **O(n3)**
4. **O(n2)**
5. **~~O(n\*log~~~~2~~~~n)~~ 错误**

2.7 假设顺序表L中的元素按从小到大的次序排列，设计算法以删除表中重复的元素，并要求时间尽可能少。要求：

（1）对顺序表（1,1,2,2,2,3,4,5,5,5,6,6,7,7,8,8,8,9）模拟执行本算法，并统计移动元素的次数。

**次数：~~6次~~ 错误，问的是次数**

**运行结果：**



**代码（C）：**

**void DeWeight(List \*L) {**

**int left = 0,right = 1,same = 0,count = 0;**

**while (right != L->count) {**

**if (L->data[left] == L->data[right]) {**

**same++;**

**} else {**

**if (same > 0) {**

**// 批量右移，减少移动次数**

**for (int i = right; i < L->count; i++) {**

**L->data[i - same] = L->data[i];**

**}**

**count++;**

**L->count -= same; // 更新元素个数**

**right -= same; // 调整 right**

**same = 0;**

**}**

**left = right;**

**}**

**right++;**

**}**

**// 处理末尾情况**

**if (same > 0) {**

**L->count -= same;**

**}**

**printf("Runned %d times\n",count);**

**}**

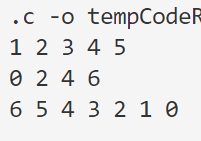
（2）分析算法的时间性能。

**只遍历一次顺序表即完成操作，时间复杂度O(n)**

2.22 设计算法将两个递增有序的带头结点的单链表A、B合并为一个递减有序的带头结点的单链表，并要求算法的时间复杂度为两个表长之和的数量级。

//设计算法两个递增有序的链表合并为一个递减有序的链表，要求时间复杂度为两个表长之和的数量级

**运行结果：**

****

**时间复杂度分析：**

**合并函数MergeUp(): O(m + n)**

**逆置函数Reverse(): O(m + n)**

**最终MergeDown(): O(m + n) + O(m + n) = O(m + n)**

**代码：**

**void MergeDown(LinkList \*L1,LinkList L2) {**

**MergeUp(L1,L2);**

**Reverse(L1);**

**}**

**//设计算法两个递增有序的链表合并为一个递增有序的链表，要求时间复杂度为两个表长之和的数量级**

**void MergeUp(LinkList \*L1,LinkList L2) {**

**LinkList list1 = (\*L1)->next,list2 = L2->next;**

**while(list1 && list2) {**

**if(list1->data > list2->data) {**

**HeadAdd(L1,list2->data);**

**list2 = list2->next;**

**} else if(list1->data < list2->data) {**

**list1 = list1->next;**

**} else {**

**list1 = list1->next;**

**list2 = list2->next;**

**}**

**}**

**while(list2) {**

**TailAdd(L1,list2->data);**

**list2 = list2->next;**

**}**

**}**

**// 逆置**

**Status Reverse(LinkList \*L) {**

**//从第1、2元素开始，最后处理头结点**

**LinkList list = (\*L)->next->next,pre = (\*L)->next,head = \*L;**

**//只有一个元素加头结点，没必要逆置**

**if(list == NULL) {**

**return OK;**

**}**

**pre->next = NULL;**

**while(list) {**

**//暂存初始状态下一个节点**

**LinkList nexts = list->next;**

**list->next = pre;**

**pre = list;**

**list = nexts;**

**}**

**//处理头节点**

**head->next = pre;**

**return OK;**

**}**