教学内容---第十章

DS/SS

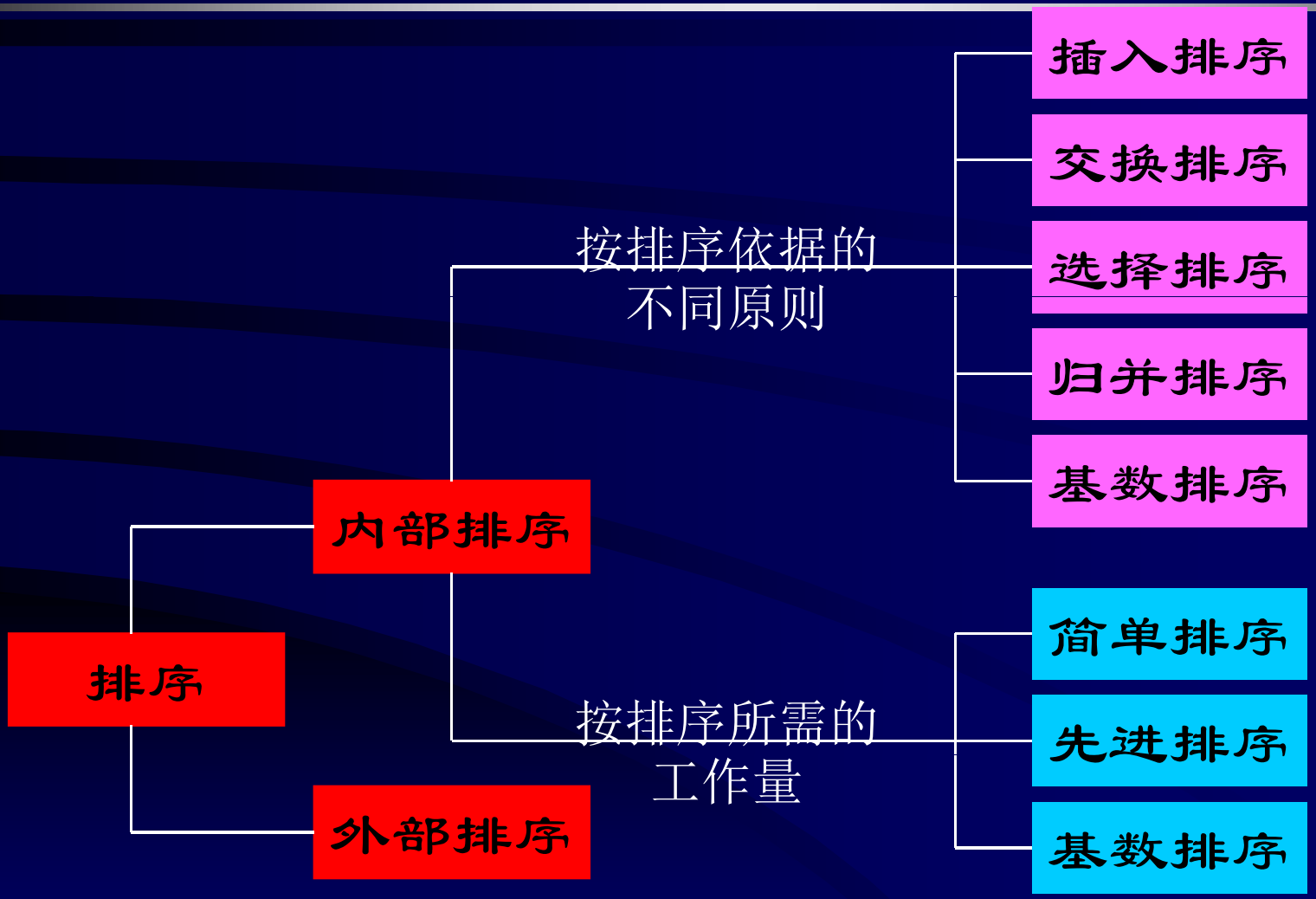
1. 绪论
2. 线性表
3. 栈、队列和串
4. 数组
5. 广义表
6. 树和二叉树
7. 图
8. 动态存储管理
9. 查找
10. 内部排序
11. 外部排序
12. 文件

10.1排序基本概念

- 排序(Sorting)：将一个数据元素的任意序列重新排成一个按照关键字有序的序列的操作。即：

  假设n个元素的序列 $\{R_1,R_2,\ldots,R_n\}$,相应关键字序列为 $\{K_1,K_2,\ldots,K_n\}$。确定1,2,…,n的一种排列$p_1,p_2,\ldots,p_n$,使其相应关键字满足非递减(或非递增)关系$K_{p1}<=K_{p2}<=\ldots<=K_{pn}$，从而得到一个按关键字有序序列 $\{R_{p1},R_{p2},\ldots,R_{pn}\}$，这样一个操作过程，就是排序。
- (排序方法)稳定：设待排元素序列中，若$K_i=K_j$ (1<=i<=n,1<=j<=n, i != j),且排序前R_i领先于R_j(即i<j),经过该方法排序后序列中R_i仍然领先于R_j，则该方法稳定。
- (排序方法)不稳定：设待排元素序列中，若$K_i=K_j$ (1<=i<=n,1<=j<=n, i != j),且排序前R_i领先于R_j(即i<j),经过该方法排序后若可能出现序列中R_j领先于R_i，则该方法不稳定。
- 内部排序：待排序元素存放在内存中进行的排序过程。
- 外部排序：待排序元素个数多，内存中一次不能容纳下，在排序过程中需要对外存进行访问的排序过程。

10.1排序基本概念（续）


排序
内部排序
外部排序
按排序依据的
不同原则
插入排序
交换排序
选择排序
归并排序
基数排序
按排序所需的
工作量
简单排序
先进排序
基数排序

10.1排序基本概念（续）

```
//待排元素一般采用顺序存储
#define  MAXSIZE          20
typedef  int        KeyType;                    //定义关键字类型为整型
typedef  struct {
         KeyType            key                 //关键字项
         infoTypeotherinfo          //元素其他数据项
}ElemType;                                       //元素类型
typedef  struct {
         ElemType           r[MAXSIZE+1]  //r[0]用作“监视哨”
         int                length              //顺序表长度
}sqList;                                         //顺序表类型
```

10.2交换排序

DS/SS

冒泡排序

算法思想:

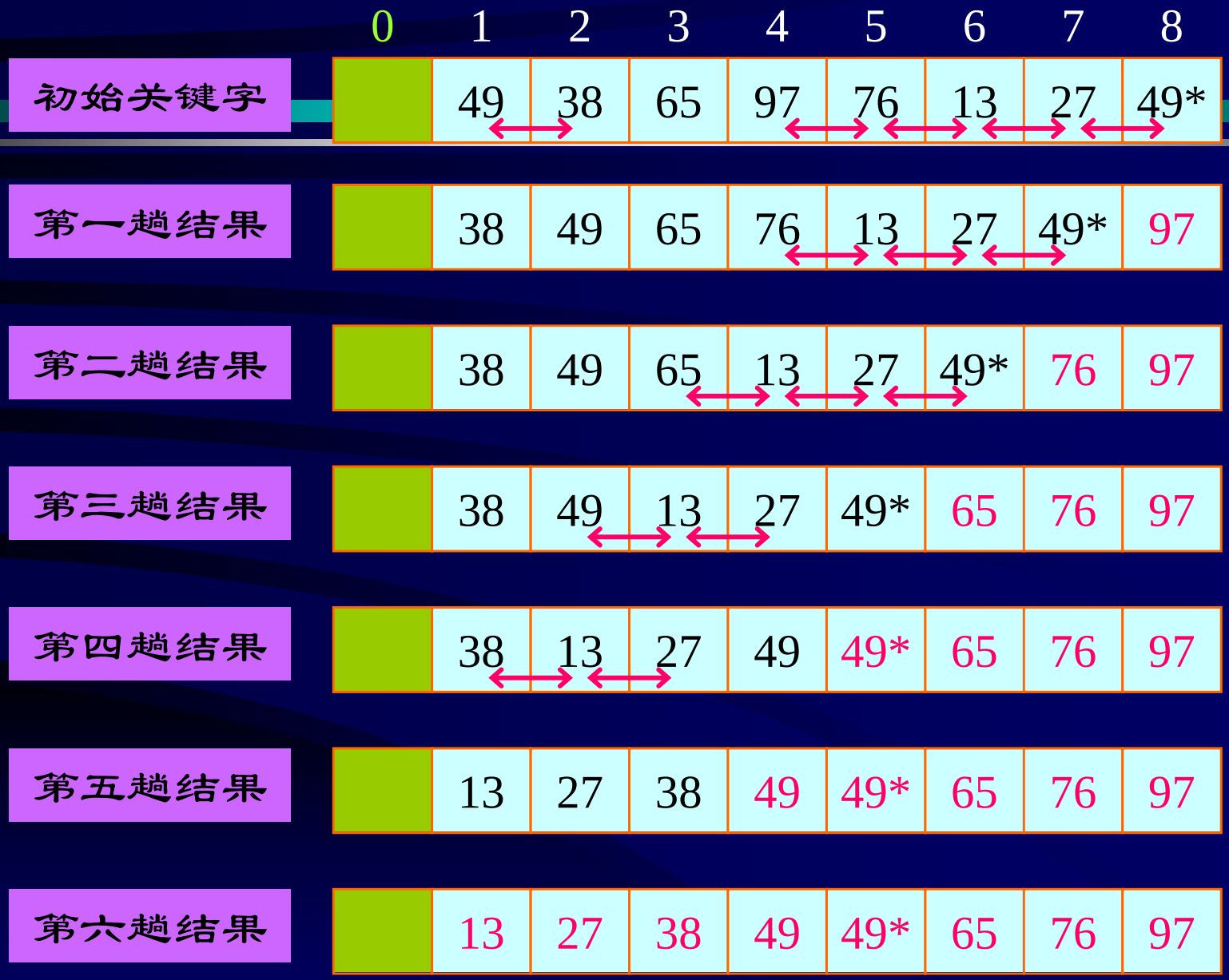
冒泡排序(Bubble Sort)的基本操作是将两个相邻元素关键字比较，若为逆序，则交换两个元素。

第i趟排序过程：从L.r[1]到L.r[n-i+1]依次比较相邻两个元素关键字，并在逆序时交换相邻元素，结果是这n-i+1个元素中关键字最大的元素被交换到第n-i+1的位置上。

整个排序过程共进行k趟(1<=k<n),判别冒泡排序结束的条件是“在一趟排序过程中没有进行过交换元素的操作”。

冒泡排序

DS/SS


0 1 2 3 4 5 6 7 8
初始关键字 49 38 65 97 76 13 27 49*
第一趟结果 38 49 65 76 13 27 49* 97
第二趟结果 38 49 65 13 27 49* 76 97
第三趟结果 38 49 13 27 49* 65 76 97
第四趟结果 38 13 27 49 49* 65 76 97
第五趟结果 13 27 38 49 49* 65 76 97
第六趟结果 13 27 38 49 49* 65 76 97

10.2交换排序（续）

DS/SS

性能分析:

时间复杂度$T(n)=O(n^2)$

- 正序： 比较次数： n-1；
  元素不移动
- 逆序： 比较次数:n(n-1)/2；
  移动次数:n(n-1)/2

空间复杂度$S(n)=O(1)$

稳定性：

冒泡排序方法是稳定的排序方法

适用情况：

*元素初始序列基本有序；

*元素个数较少

10.2交换排序（续）

DS/SS

快速排序

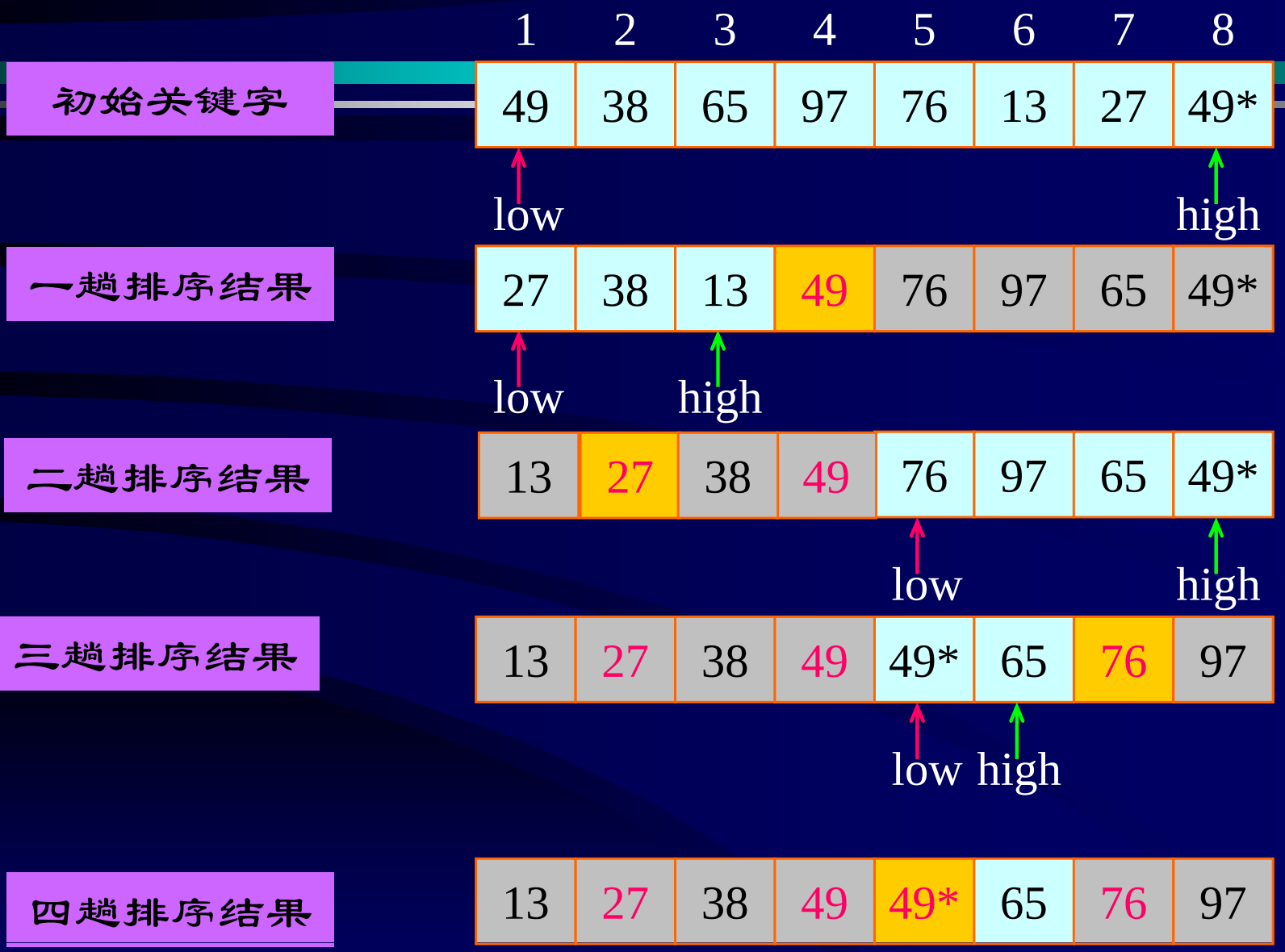
算法思想:

快速排序(Quick Sort)的基本基本思想：通过一趟排序将待排元素分割成独立的两部分,其中一部分元素的关键字均比另一部分元素的关键字小,则可分别对这两部分元素继续进行排序,以达到整个序列有序。

一趟排序(划分)过程：对待排序列{L.r[s],L.r[s+1],…,L.r[t]},任意选取一个元素作为枢轴(pivot),将所有关键字较它小的元素都安置在它的位置之前,将所有关键字较它大的元素都安置在它的位置之后。最后，以该“枢轴”元素最后所落位置作分界线，将序列分割成两个子序列。

整个排序过程共进行的趟数与每一趟排序过程中枢轴元素的选取有关，如果每一趟划分“均匀”，则总趟数可以减到最小。

快速排序


1 2 3 4 5 6 7 8
初始关键字
49 38 65 97 76 13 27 49*
low
high
一趟排序结果
27 38 13 49 76 97 65 49*
low
high
二趟排序结果
13 27 38 49 76 97 65 49*
low
high
三趟排序结果
13 27 38 49 49* 65 76 97
low high
四趟排序结果
13 27 38 49 49* 65 76 97

```
int Partition(SqList  &L ,int low, int high){    //一趟快速排序
  pivotkey = L.r[low].key              //用子表的第一个元素作枢轴元素
  L.r[0] = L.r[low].key;               //将枢轴元素暂存在r[0]位置处
  while (low<high){                    //从表的两端交替地向中间扫描
     while (low<high && L.r[high].key >= pivotkey)  --high;
     L.r[low] = L.r[high];
     while (low<high && L.r[low].key <= pivotkey)  ++low;
     L.r[high] = L.r[low];
   }
  L.r[low] = L.r[0];                   //枢轴元素到位(此时low=high)
  retun low;  }  // Partition
```

```
Void QSort(SqList  &L ,int low, int high){  //完成一次完整的快速排序
  if (low<high) { pivotloc = partition(L, low,high);  //将L.r[low..high]一分为二
                  Qsort(L,low,pivotloc-1); Qsort(L, pivotloc+1, high);  } }  //QSort
```

10.2 交换排序（续）

DS/SS

性能分析:

平均时间复杂度T(n)=O($knlnn$) ,k为常数。

*就平均时间而言，快速排序是目前最好的一种内部排序方法

*若初始序列基本有序，快速排序就蜕变为冒泡排序

*一般选枢轴元素采取“三者取中”法则

一般情况下空间复杂度S(n)=O($\log_2 n$),最坏情况下达到O(n)(若改进算法，在一趟排序之后比较分割所得两部分长度，且先对长度短的子序列元素进行快速排序，则栈最大深度可降为O($\log_2 n$)

稳定性:

快速排序方法是不稳定的排序方法

适用情况:

*元素初始序列不是基本有序;

10.3插入排序

DS/SS

直接插入排序

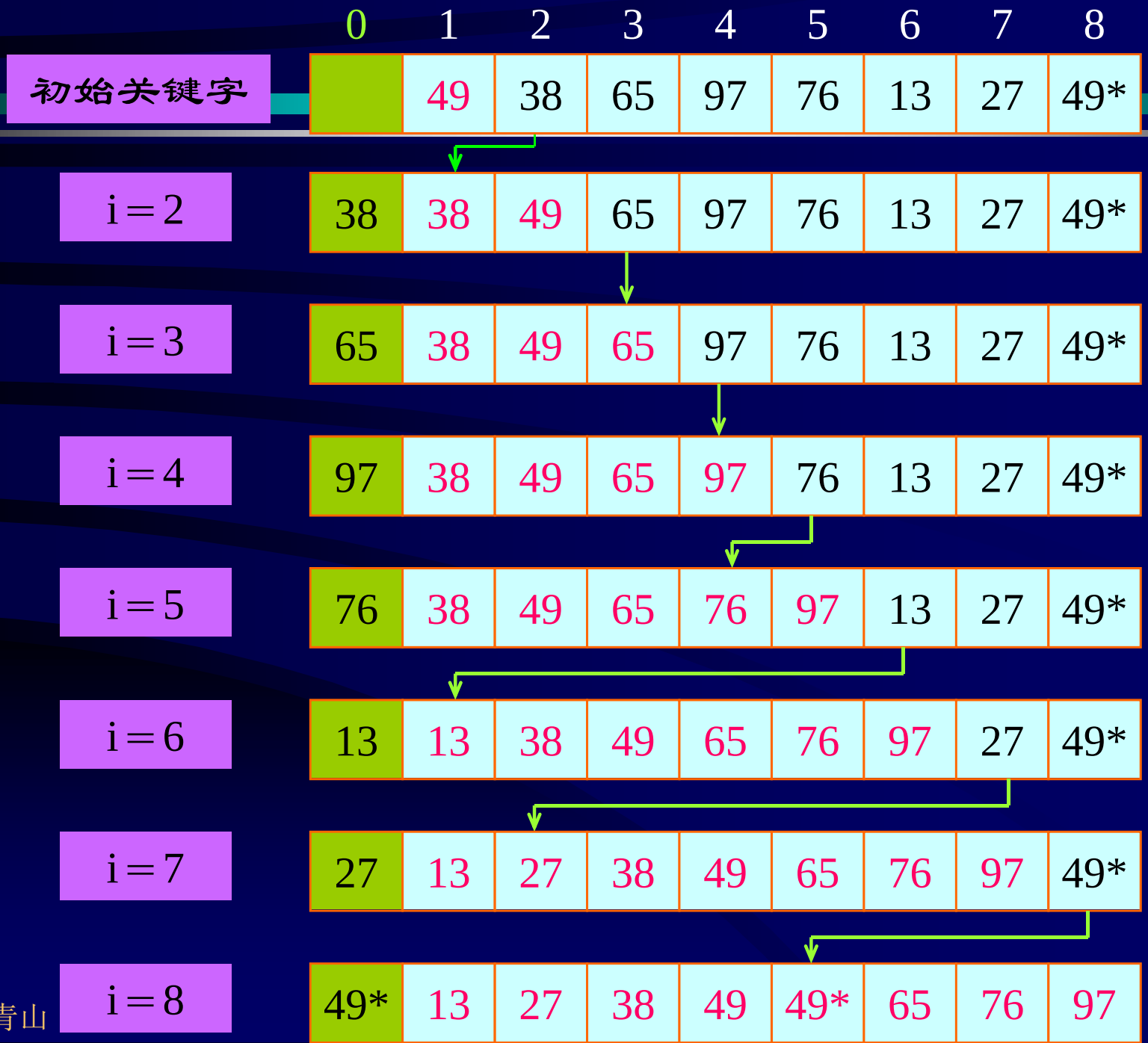
算法思想:

直接插入排序(Straight Insertion Sort)的基本操作是将一个元素插入到已排好序的有序表中,从而得到一个新的、元素数增1的有序表。

第i趟排序过程：在含有i－1个元素的有序子序列r[1..i-1]中插入一个元素r[i]后，变成含有i个元素的有序子序列r[1..i]。

整个排序过程共进行n-1趟(即i从2变化到n):先将序列中的第1个元素看成是一个有序的子序列，然后从第2个元素起逐个进行插入，直至整个序列变成按关键字非递减有序序列为止。

直接插入排序


0 1 2 3 4 5 6 7 8
初始关键字
49 38 65 97 76 13 27 49*
i＝2
38 38 49 65 97 76 13 27 49*
i＝3
65 38 49 65 97 76 13 27 49*
i＝4
97 38 49 65 97 76 13 27 49*
i＝5
76 38 49 65 76 97 13 27 49*
i＝6
13 13 38 49 65 76 97 27 49*
i＝7
27 13 27 38 49 65 76 97 49*
i＝8
49* 13 27 38 49 49* 65 76 97

10.3插入排序（续）

直接插入排序

```
Void InsertSort(SqList  &L ){
    for (i=2; i<=L.length; ++i)
      if LT(L.r[i].key,L.r[i-1].key) {
          L.r[0] = L.r[i];                          //监视哨为当前插入元素
          for (j = i-1; LT(L.r[0].key,L.r[j].key); --j)
             L.r[j+1] = L.r[j];                     //元素后移
          L.r[j+1] = L.r[0];                        插入到正确位置
      }
} //InsertSort
```

10.3插入排序（续）

DS/SS

性能分析：

时间复杂度$T(n)=O(n^2)$

- 正序：比较次数：n-1；
  元素不移动
- 逆序：比较次数：(n+2)(n-1)/2；
  移动次数：(n+4)(n-1)/2

空间复杂度$S(n)=O(1)$

稳定性：

直接插入排序方法是稳定的排序方法

适用情况：

*元素初始序列基本有序；

*元素个数少

10.3插入排序（续）

DS/SS

希尔排序

算法思想：

希尔排序(Shell's Sort)称为“缩小增量排序”，它是从两点出发对直接插入排序进行改进而得到的一种插入排序排序。第一点，元素基本有序时，直接插入排序时间复杂度接近O(n)；第二点，元素个数n很小时，直接插入排序效率也比较高。

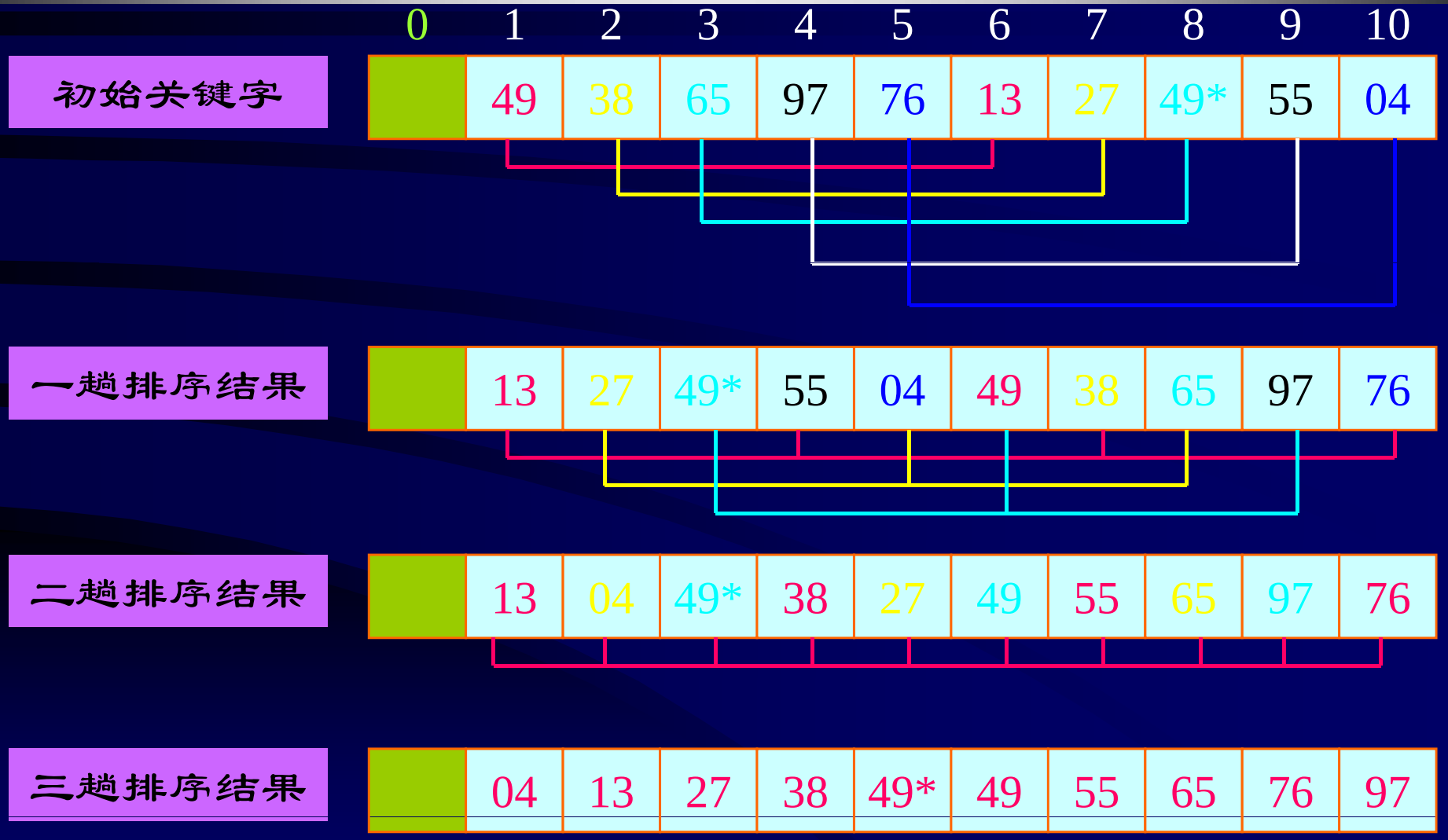
希尔排序的基本思想：先将整个待排元素序列分割成若干子序列分别进行直接插入排序（这些子序列要么元素个数少、要么基本有序，从而直接插入排序效率高），待整个序列中的元素“基本有序”时，再对全体元素进行一次直接插入排序。

子序列的构成不是简单地“逐段分割”，而是将相隔某个“增量”的元素组成一个子序列。这样便于每一趟排序都均匀展开。

排序的趟数就是增量序列的长度。

希尔排序（增量序列为：5、3、1）

DS/SS



10.3插入排序（续）

```
Void ShellInsert(SqList  &L ,int dk){    //一趟希尔排序，前后元素位置增量为dk
   for (i=dk+1; i<=L.length; ++i)
     if LT(L.r[i].key,L.r[i-dk].key) {//将L.r[i]插入有序增量子表
        L.r[0] = L.r[i];                          //L.r[0]为暂存单元
        for (j = i-dk; j>0&&LT(L.r[0].key,L.r[j].key); j-=dk)
           L.r[j+dk] = L.r[j];                    //元素后移
        L.[j+dk] = L.r[0];                        插入到正确位置
     }
} //ShellInsert
```

```
Void ShellSort(SqList  &L ,int dlta[], int t){    //按增量序列dlta[0..t-1]作希尔排序
   for (k=0; k<t; ++k)   ShellInsert(L,dlta[k]);
} //ShellSort
```

10.3插入排序（续）

DS/SS

性能分析:

时间复杂度T(n) =

- $O(n^{1.5})$：增量序列dlta[k]=$2^{t-k+1}-1$
- $O(n^{1.3})$：平均情况
- $O(n(\log_2 n)^2)$：n趋于无穷大时

空间复杂度S(n)=O(1)

稳定性：

希尔排序方法是不稳定的排序方法

增量序列选择：

*增量序列中的值没有除1之外的公因子；

*最后一个增量值必须等于1

10.5归并排序

DS/SS

2-路归并排序

算法思想:

归并排序(Merging Sort)的基本操作是“归并”：将两个或两个以上的有序表组合成一个新的有序表。

2-路归并过程：假设初始序列含n个元素，看成n个子序列，两两归并，得到“n/2取上整”个长度为2或为1的有序子序列；再两两归并，如此重复，直至得到一个长度为n的有序序列为止。

整个排序过程共进行$\log_2 n$趟。每一趟归并调用“(n/(2h))取上整”次核心归并算法，将SR[1..n]中前后相邻长度为h的有序段进行两两归并，得到前后相邻、长度为2h的有序段，并存放在TR[1..n]中。交替进行。

10.5归并排序（续）

DS/SS

```
void Merge(RcdType  SR[] , RcdType  &TR[] , int i, int m, int n){  //核心归并算法
    //将有序的SR[i..m]和SR[m+1..n]归并为有序的TR[i..n]
    for (j =m+1, k =i; i<=m&& j<=n; ++k) {  //将SR中元素由小到大地并入TR
        if LQ(SR[i].key, SR[j].key)   TR[k] = SR[i++];
        else  TR[k] = SR[j++];
    }
    if(j<=m) TR[k..n] = SR[i..m];                  //将剩余的SR[i..m]复制到TR
    if(j<=n) TR[k..n] = SR[j..n];                  //将剩余的SR[j..n]复制到TR
}  // Merge
```

10.5归并排序（续）

2-路归并排序

性能分析:

时间复杂度$T(n)= O(n\log_2 n)$

空间复杂度$S(n)=O(n)$ (每一趟交替进行)

稳定性：

归并排序方法是稳定的排序方法

适用情况：

*元素个数较多（辅助外排序情况下常用）

*n较大时，归并排序所需时间较堆排序省，但辅助空间多