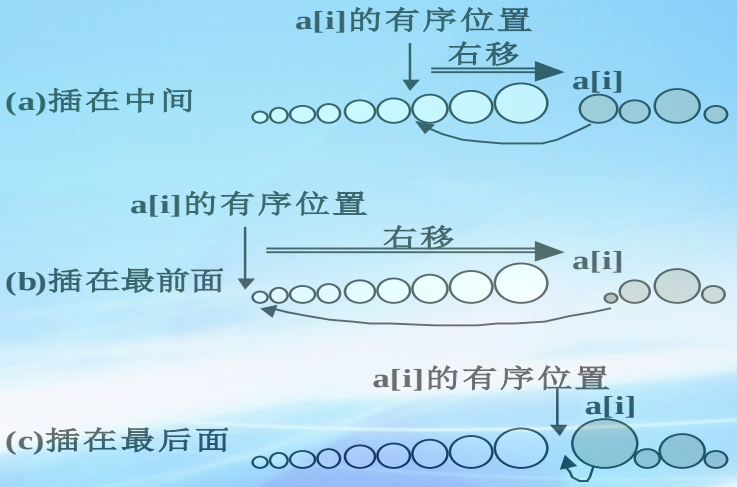
插入排序

《数据结构》

指院网络工程教研中心　陈卫卫



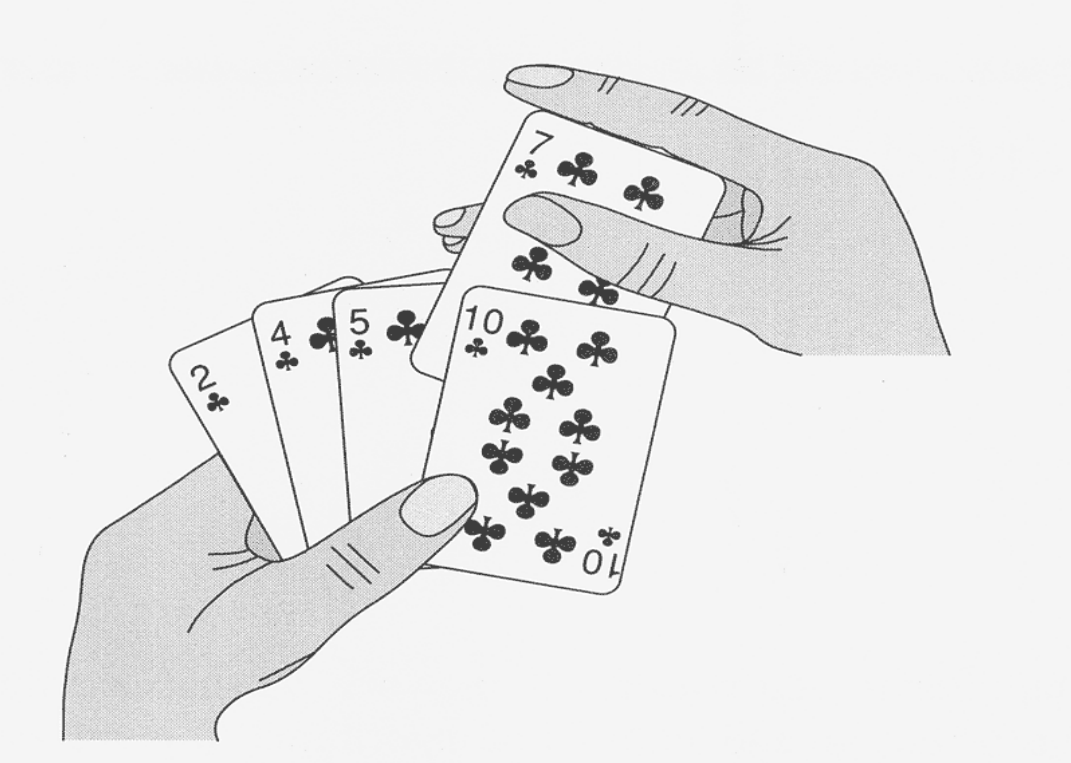
插入排序的基本原理

学习目标和要求

1、理解插入排序的基本原理

2、知道插入排序算法的种类

插入排序的基本原理



插入排序的基本原理

基本操作：有序插入

在有序序列中插入一个元素，保持序列有序

起初，a[0]是长度为1的子序列

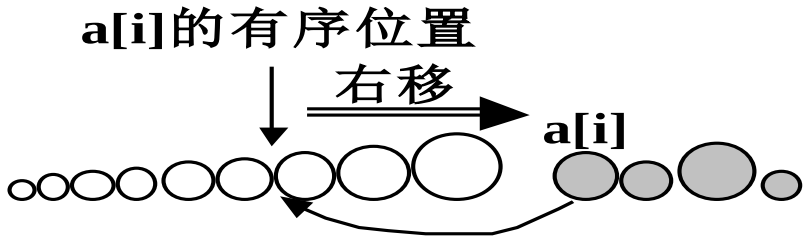
然后，逐一将a[1]至a[n-1]插到有序子序列中

有序插入方法

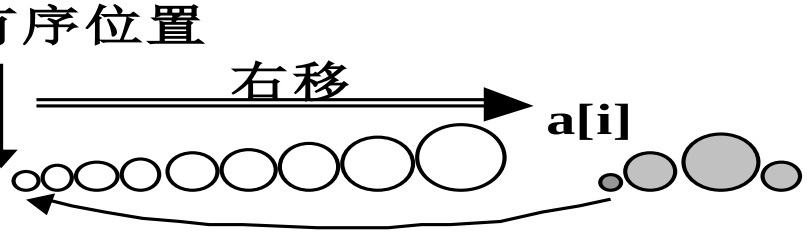
- 在插入a[i]前，数组a的前半段（a[0]~a[i-1]）是有序段，后半段（a[i]~a[n-1]）是停留于输入次序的“无序段”。
- 插入a[i]使a[0]~a[i]有序，也就是要为a[i]找到有序位置j（0≤j≤i），将a[i]插在a[j]的位置上。

图示

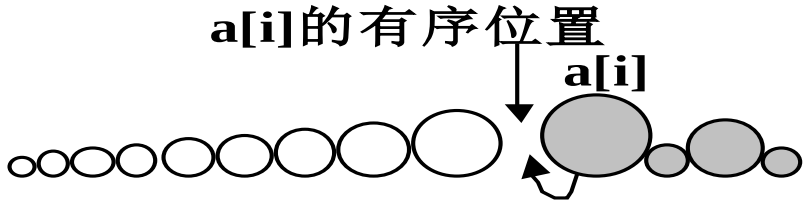
(a)插在中间



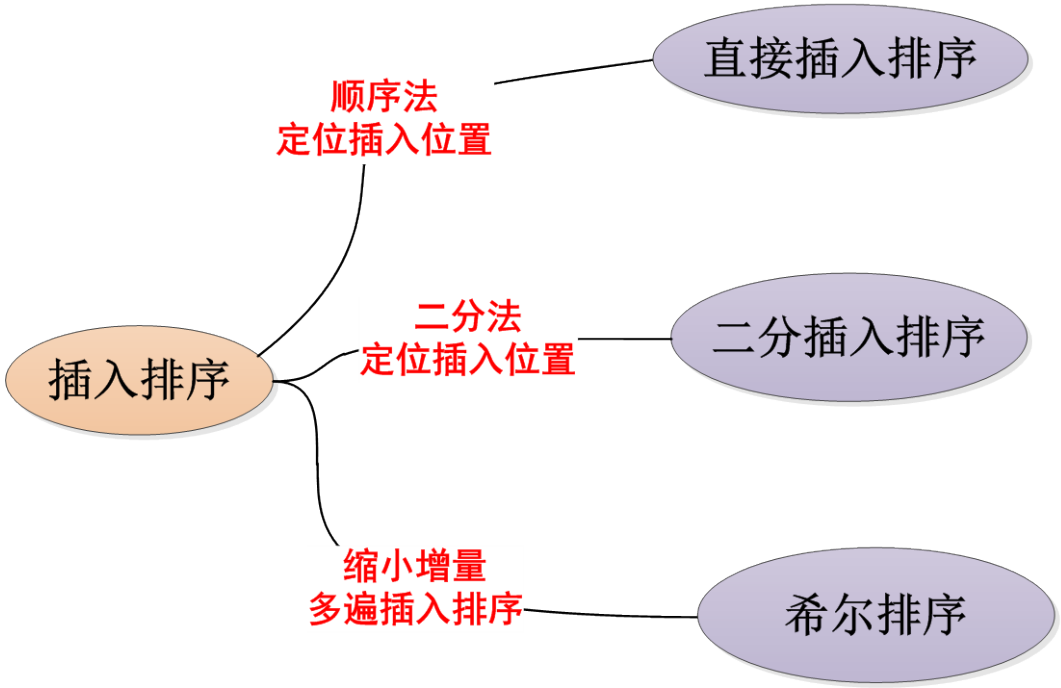
(b)插在最前面



(c)插在最后面



插入排序的种类



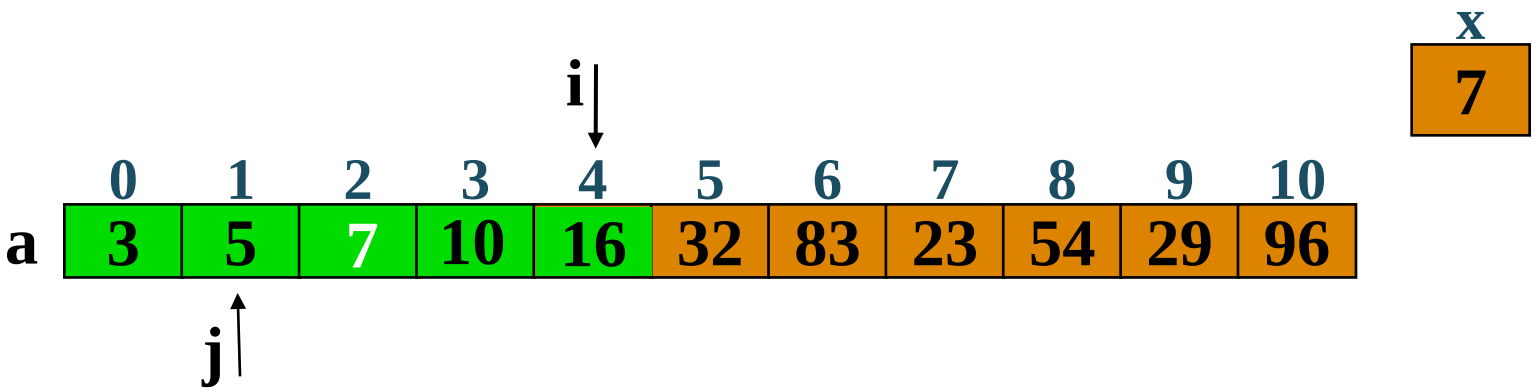
直接插入排序

学习目标和要求

1、编写直接插入排序算法

2、知道算法的时间复杂性和稳定性

1.直接插入排序算法示例



2. x=a[i];

3. for(j=i−1; j>=0&&x<a[j]; j−−) a[j+1]=a[j];

2.简单的直接插入排序

直接插入排序算法之一

```
   void Sinsert_sort(int a[ ],int n)
   { int i,j,x;
1.   for(i=1;i<n; i++)
      {
2.     x=a[i];     //空出a[i]的位置
3.     for(j=i−1; j>=0&&x<a[j]; j− −) a[j+1]=a[j]; //移动，定位
4.     a[j+1]=x;  //将x 插在位置j+1
      }
    }
```

3．性能分析

根据插入a[i]时元素比较次数Ci和移动次数Mi的最小值、最大值、平均值计算总耗费

$$C_{min} = \sum_{i=1}^{n-1} 1 = n-1 \qquad C_{max} = \sum_{i=1}^{n-1} i = \frac{1}{2}(n^2 - n)$$

$$M_{min} = \sum_{i=1}^{n-1} 2 = 2(n-1) \qquad M_{max} = \sum_{i=1}^{n-1} (i+1) = \frac{1}{2}(n^2 + n - 2)$$

$$C_{avg} = \sum_{i=1}^{n-1} \frac{i+1}{2} = \frac{1}{4}(n+2)(n-1)$$

$$M_{avg} = \sum_{i=1}^{n-1} (\frac{i+1}{2} + 1) = \frac{1}{4}(n+6)(n-1)$$

4．时间复杂性结论

- 原始数据越接近有序，排序速度越快
- 最坏情况下（输入数据是逆有序的）

  $T_W(n)=O(n^2)$
- 平均情况下，耗时差不多是最坏情况的一半

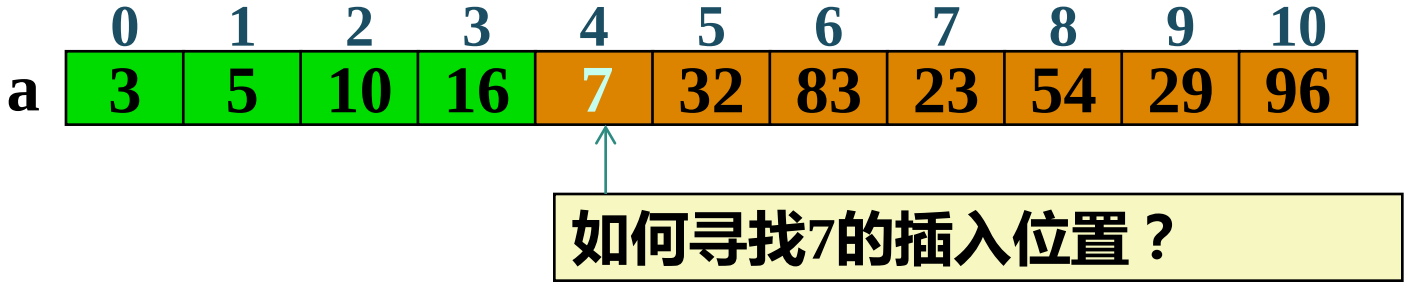
  $T_E(n)=(n^2)$
- 为提高排序速度，要
  - 减少元素的比较次数
  - 减少元素的移动次数

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√

问题

直接插入排序如何寻找插入位置？



方法：顺序查找，能否改进？

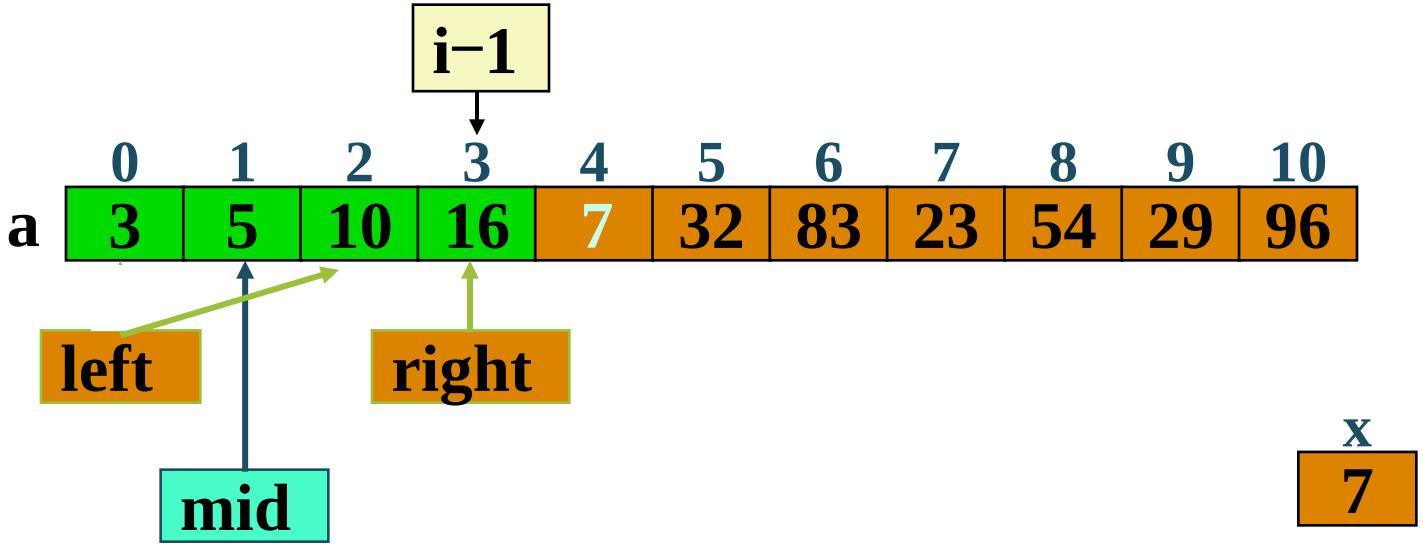
二分插入排序

学习目标和要求

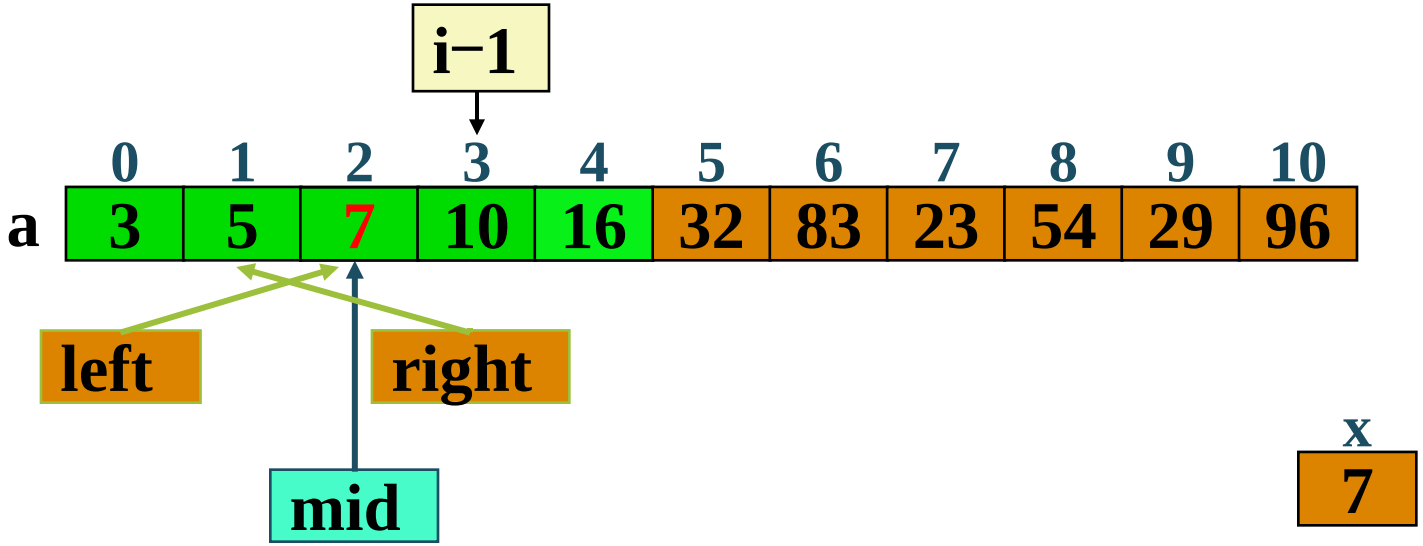
1. 编写二分插入算法

2. 知道二分插入排序算法时间性能

1.二分插入排序算法示例之一

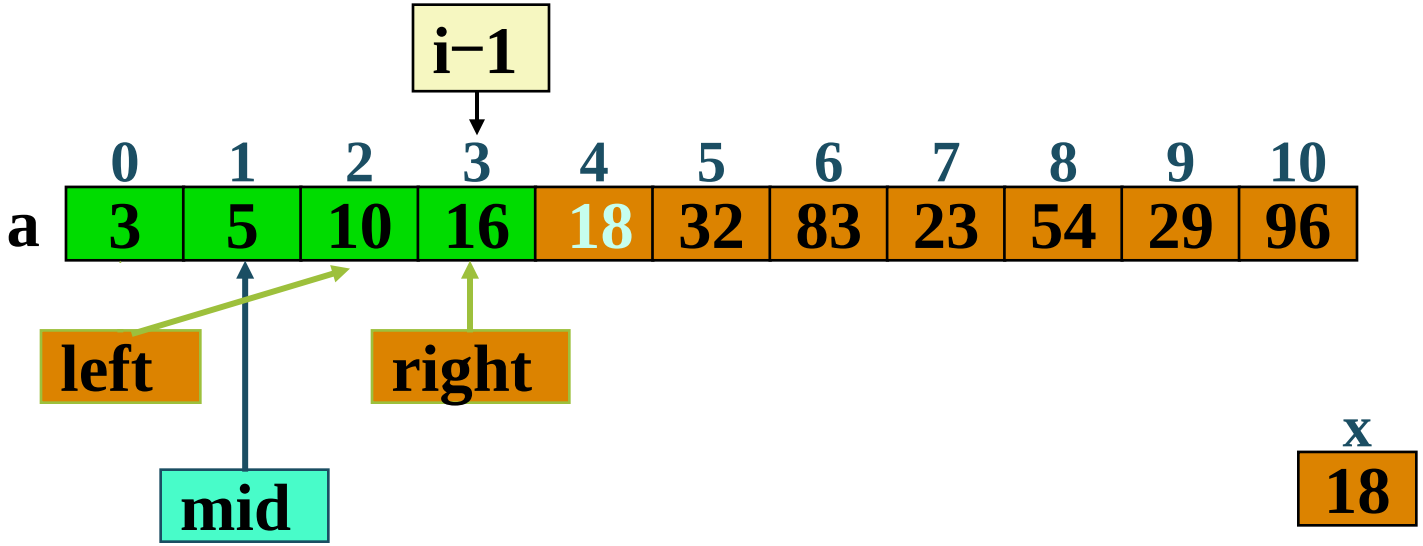


1.二分插入排序算法示例之一



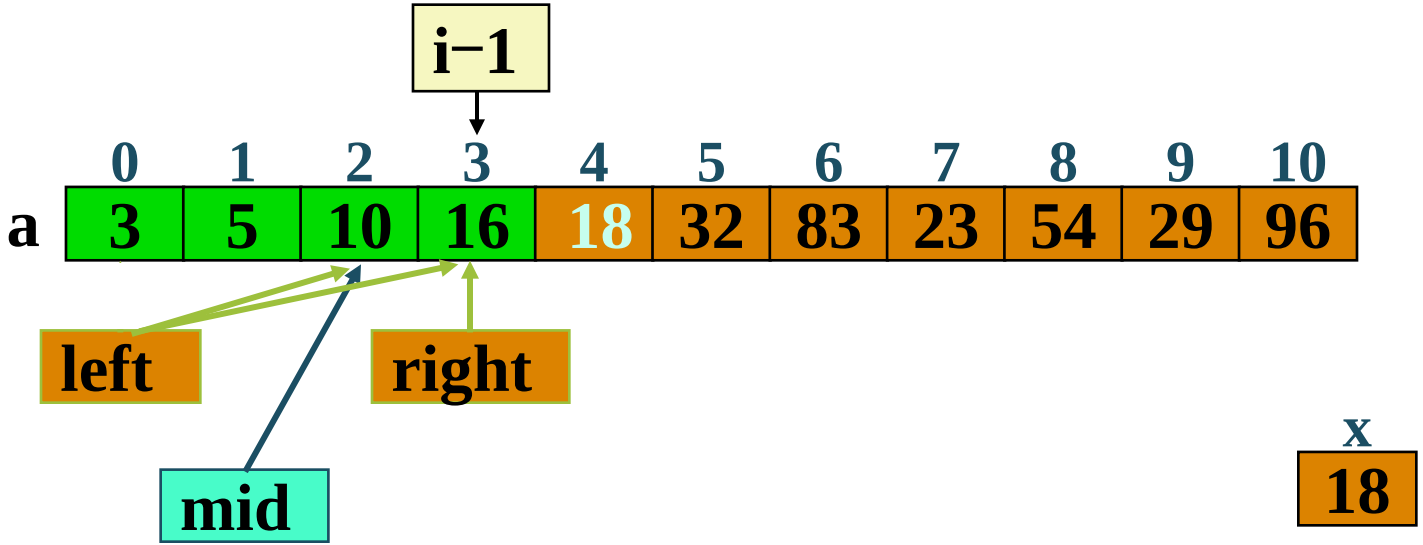

```
8.    for(j=i−1;j>=left;j−−) a[j+1]=a[j];  // 元素右移
9.    a[left]=x;  //元素x就位
```

1.二分插入排序算法示例之二



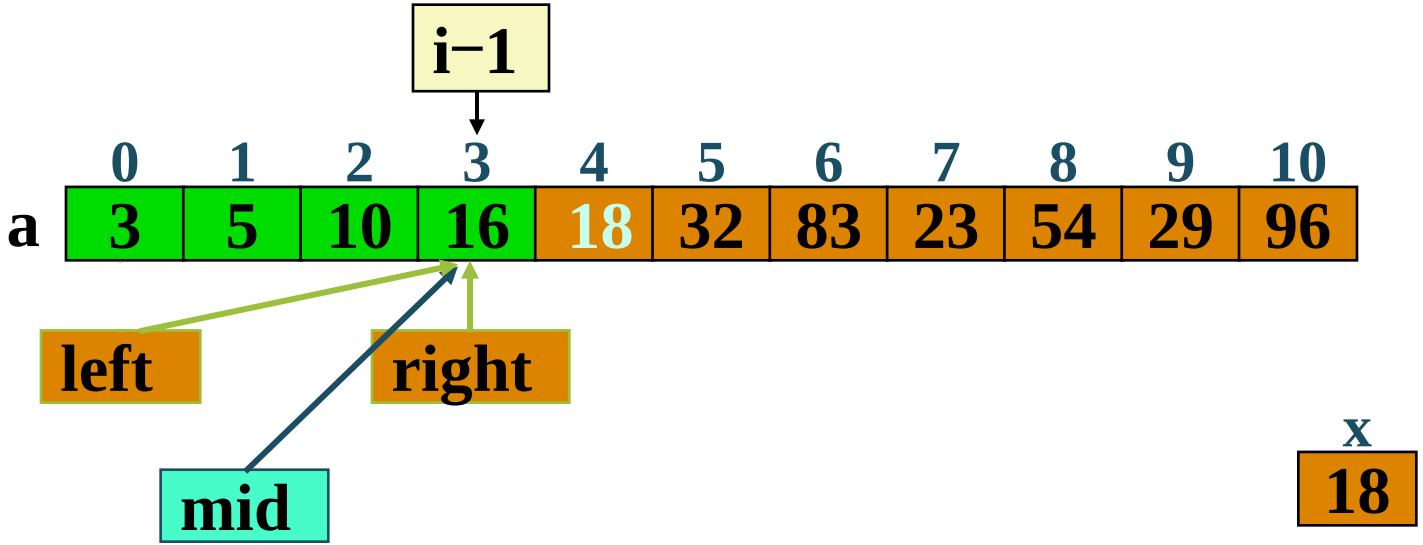
8. for(j=i−1;j>=left;j−−) a[j+1]=a[j]; // 元素右移
9. a[left]=x; //元素x就位

1.二分插入排序算法示例之二



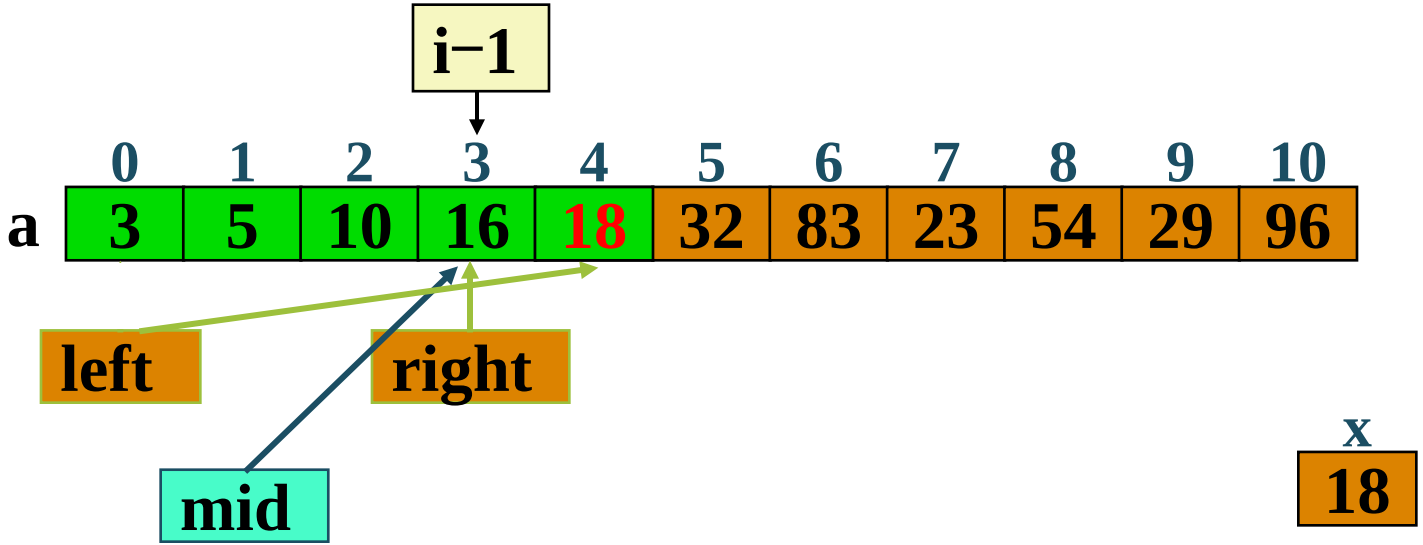

```
8.    for(j=i−1;j>=left;j−−) a[j+1]=a[j];  // 元素右移
9.    a[left]=x;  //元素x就位
```

1.二分插入排序算法示例之二




```
8.    for(j=i−1;j>=left;j−−) a[j+1]=a[j];  // 元素右移
9.    a[left]=x;  //元素x就位
```

1.二分插入排序算法示例之二




```
8.    for(j=i−1;j>=left;j−−) a[j+1]=a[j];  // 元素右移
9.    a[left]=x;  //元素x就位
```

2.二分插入排序算法

```
void  Binsert_sort(int a[ ],int n)
{ int i,j,left,right,mid,x;
1.  for(i=1;i<n;i++)  //准备插入a[i]
2.   {   x=a[i];   //暂存插入元素a[i]
3.       left=0;  right=i−1;  //设置查找段的起点和终点
4.       while(left<=right)     //二分定位
5.         { mid=(left+right)/2;
6.            if(x<a[mid])  right=mid−1;
7.               else   left=mid+1;
           }
8.      for(j=i−1;j>=left;j− − ) a[j+1]=a[j]; //元素右移
9.      a[left]=x;  //元素a[i]就位      }
  }
```

$T(n)=O(n^2)$

问题

可以增大移动的步幅吗？

比较一次，移动1步

比较一次，移动一大步？

学习目标和要求

1. 理解shell排序的核心思想
2. 编程实现shell排序算法
3. 知道算法的时间复杂性和稳定性

希尔排序

Donald. L. Shell设计的排序算法特点：

（1）缩小增量

（2）多遍插入排序

例如，选用增量序列（9，5，3，1），进行4遍插入排序

- h-排序：增量为h的那一遍排序。
- h-子序列：下标间隔为h的元素组成的子序列。
- h-前驱(后继)：同一个h-子序列中的元素之间有h-前驱后继关系。
- h-有序的：对数组a进行h-排序之后，每个h-子序列分别有序。

示例

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

84 92 78 67 21 79 23 57 86 32 46 19 22 90 15 69 42 13 47 38 55 34 18 43 26 94 72 62

第一遍增量为9（9个9-子序列）

	9-排序前	9-排序后
（1）	84 32 47 62	32 47 62 84
（2）	92 46 38	38 46 92
（3）	78 19 55	19 55 78
（4）	67 22 34	22 34 67
（5）	21 90 18	18 21 90
（6）	79 15 43	15 43 79
（7）	23 69 26	23 26 69
（8）	57 42 94	42 57 94
（9）	86 13 72	13 72 86

9-排序后（呈9-有序的）

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	84	92	78	67	21	79	23	57	86	32	46	19	22	90	15	69	42	13	47	38	55	34	18	43	26	94	72	62
h=9	32	38	19	22	18	15	23	42	13	47	46	55	34	21	43	26	57	72	62	92	78	67	90	79	69	94	86	84

第一遍增量为9（9个9-子序列）

9-排序前				9-排序后			
（1）	84	32	47	62	32	47	62	84
（2）	92	46	38		38	46	92	
（3）	78	19	55		19	55	78	
（4）	67	22	34		22	34	67	
（5）	21	90	18		18	21	90	
（6）	79	15	43		15	43	79	
（7）	23	69	26		23	26	69	
（8）	57	42	94		42	57	94	
（9）	86	13	72		13	72	86	

9-排序后（呈9-有序的）

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	84	92	78	67	21	79	23	57	86	32	46	19	22	90	15	69	42	13	47	38	55	34	18	43	26	94	72	62
h=9	32	38	19	22	18	15	23	42	13	47	46	55	34	21	43	26	57	72	62	92	78	67	90	79	69	94	86	84
	15	23	19	13	18	26	38	34	21	43	32	55	42	22	47	46	57	72	62	69	78	67	84	79	92	94	86	90

第二遍增量为5（5个5-子序列）

	5-排序前	5-排序后
（1）	32 15 46 26 78 94	15 26 32 46 78 94
（2）	38 23 55 57 67 86	23 38 55 57 67 86
（3）	19 42 34 72 90 84	19 34 42 72 84 90
（4）	22 13 21 62 79	13 21 22 62 79
（5）	18 47 43 92 69	18 43 47 69 92

5-排序后（呈5-有序的）

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	84	92	78	67	21	79	23	57	86	32	46	19	22	90	15	69	42	13	47	38	55	34	18	43	26	94	72	62
h=9	32	38	19	22	18	15	23	42	13	47	46	55	34	21	43	26	57	72	62	92	78	67	90	79	69	94	86	84
h=5	15	23	19	13	18	26	38	34	21	43	32	55	42	22	47	46	57	72	62	69	78	67	84	79	92	94	86	90
	13	18	19	15	22	21	38	23	26	42	32	47	43	34	55	46	57	72	62	69	78	67	84	79	90	94	86	92

第三遍增量为3（3个3-子序列）

	3-排序前										3-排序后									
（1）	15	13	38	43	42	46	62	67	92	90	13	15	38	42	43	46	62	67	90	92
（2）	23	18	34	32	22	57	69	84	94		18	22	23	32	34	57	69	84	94	
（3）	19	26	21	55	47	72	78	79	86		19	21	26	47	55	72	78	79	86	

1-排序后（有序的）

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	84	92	78	67	21	79	23	57	86	32	46	19	22	90	15	69	42	13	47	38	55	34	18	43	26	94	72	62
h=9	32	38	19	22	18	15	23	42	13	47	46	55	34	21	43	26	57	72	62	92	78	67	90	79	69	94	86	84
h=5	15	23	19	13	18	26	38	34	21	43	32	55	42	22	47	46	57	72	62	69	78	67	84	79	92	94	86	90
h=3	13	18	19	15	22	21	38	23	26	42	32	47	43	34	55	46	57	72	62	69	78	67	84	79	90	94	86	92
h=1	13	15	18	19	21	22	23	26	32	34	38	42	43	46	47	55	57	62	67	69	72	78	79	84	86	90	92	94

增量序列的选择

d[0]>d[1]>…>d[t-1]　而且　d[t-1]=1

$$1 \le d[i] \le \sqrt{n}$$

希尔排序算法

```
void Shell_sort(int a[ ],int d[ ],int n,int t)
  { int  i,j,h,k,x;
1.  for(h=0;h<t;h++)
2.   {   k=d[h];   //当前增量为k
3.       for(i=k;i<n;i++)
4.      {  x=a[i];
5.         for(j=i-k;j>=0&&x<a[j];j-=k;)
6.            a[j+k]=a[j];
7.          a[j+k]=x;  //x就位
        }
      }
   }
```

```
//直接插入排序算法
void Sinsert_sort(int a[ ],int n)
  { int i,j,x;
1.  for(i=1;i<n; i++)
2. {  x=a[i];     //空出a[i]的位置
3.    for(j=i-1; j>=0&&x<a[j]; j--)
4.       a[j+1]=a[j];
5.    a[j+1]=x;  //x就位
    }
  }
```

算法的时间复杂度

$T(n)=O(n^{3/2})$

（1）…9，5，3，1　（形如2^k+1）

（2）…12，6，3，2，1　（形如$2^p\times3^q$）

（3）…15，7，3，1　（形如2^k-1）

（4）…11，5，3，1（形如$(2^k-(-1)^k)/3$）

（5）…40，13，4，1　（形如$(3^k-1)/2$）

$T(n)=O(n(\log n)^2)$

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
希尔排序	与增量序列有关：$O(n^{3/2})$,$O(n(\log n)^2)$			×