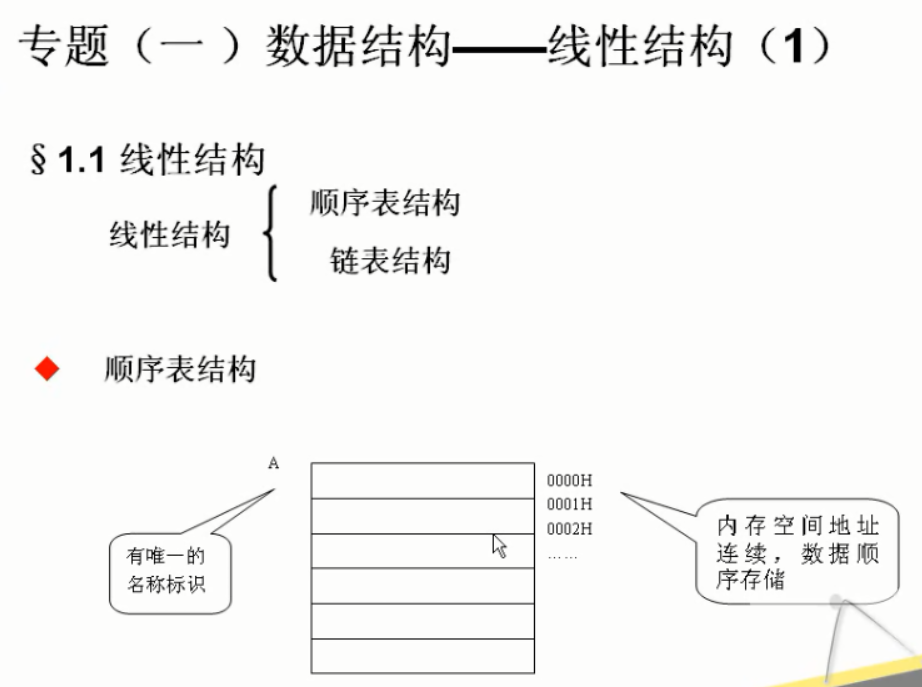
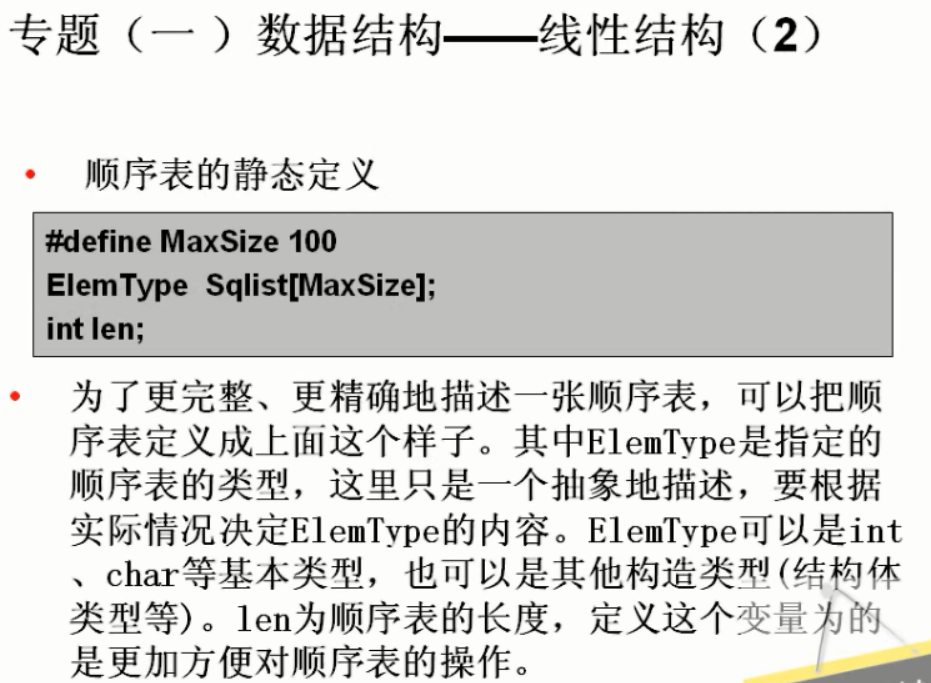
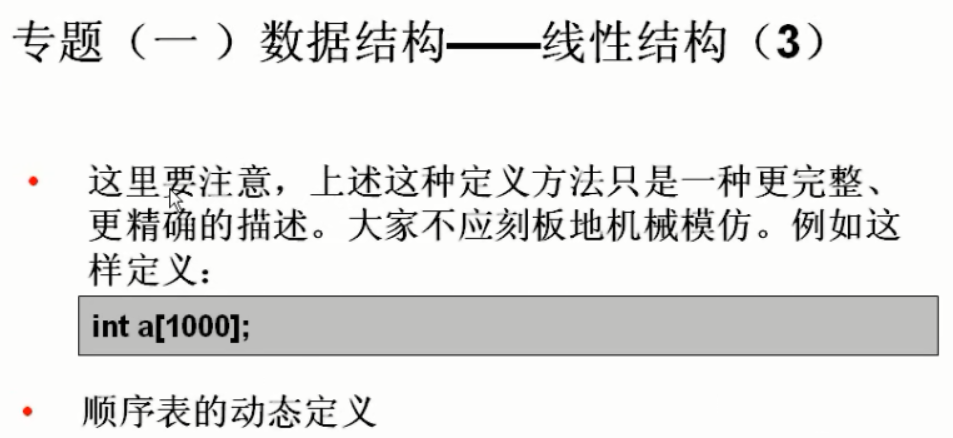
# 线性结构

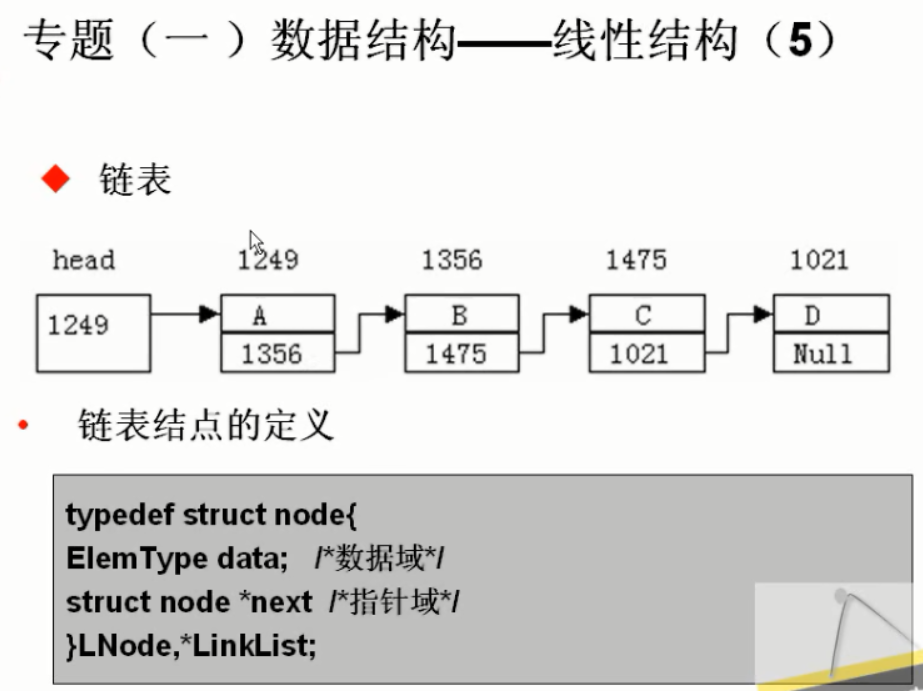


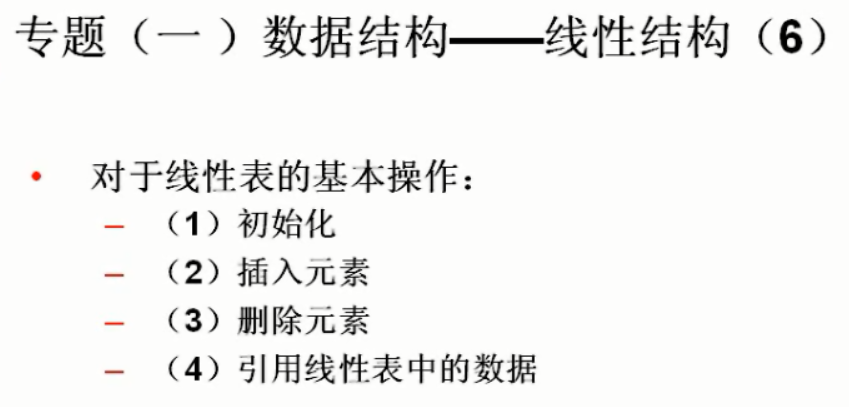






静态与动态的区别：静态是在栈中分配空间的，静态的话，如果函数执行完则系统会自动释放，而动态的是在堆里分配空间，即使函数执行完内存也不会释放，需要手动释放。



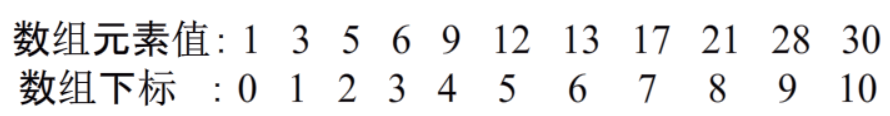


# 查找算法

## 折半查找法

就是分成一半一半地去查找。

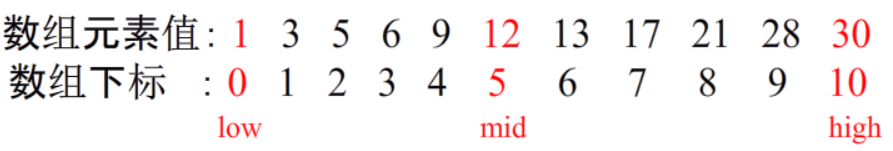
使用折半查找的前提条件是：这个序列必须是有序的，可以是递增，也可以是递减，且只适用于顺序存储结构的序列查找，不适用于链表。



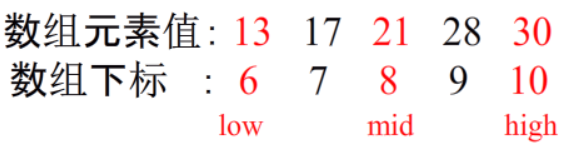
如上面的数组，我们要查找28是否存在于该数组中。

通过下标可知有11个元素，折半就是要找到中间元素的下标，看上图可直观地知道中间元素的下标为5，如何计算出来呢？（0 + 10）/ 2 = 5；也就是用low表示第一个元素下标0，high表示最后一个元素下标10，则mid = (low + high) / 2；如果只有10个元素，那4和5哪个是mid呢，其实哪个都无所谓，根据公式(0 + 9) / 2 = 4；小数会被直接去除，则4为mid，其实5为mid也没什么差别的。

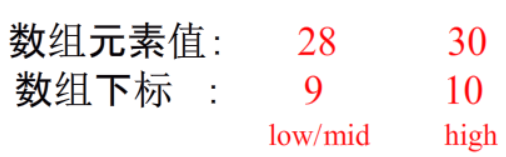
折半查找法的关键点就在于找到mid，如上图的mid如下：



找到mid 为5，则可以进行查找了，28 > arr[mid]，则28只有可能出现在mid + 1 ~ high这个范围，不可能出现在low ~ mid - 1这个范围，这样我们就锁定了mid + 1 ~ high这一半了，接着把这一半再折半，low = mid + 1 = 6; mid = (6 + 10) / 2 = 8;如下：

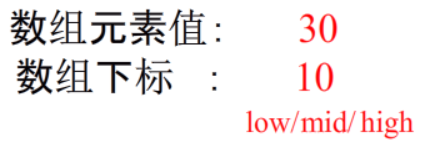


28 > arr[mid]，则28在mid + 1 ~ high位置，继续折半，low = mid + 1 = 9; mid = (9 + 10) / 2 = 9; 此时low和mid都为9，如下：



1. = arr[mid]，元素位置已找到，元素28在下标为9的位置已找到。

假设我们要查找的是29，则29 > arr[mid]，则还需要继续折半，low = mid + 1 = 10; mid = (10 + 10) / 2 = 10;此时low、mid、high都为10，如下：



1. < arr[mid]，则说明元素29可能在low ~ mid - 1之间，low为10不用变，调整high和mid的值，high = mid - 1 = 9; mid = (10 + 9）/ 2 = 9; 此时high（9）的值小于了low（10）的值，说明查找失败。

折半查找的具体实现代码如下：

**public** **static** **int** binarySearch(**int**[] arr, **int** e) {

**int** low = 0, high = arr.length - 1, mid;

**while** (low <= high) {

mid = (low + high) / 2;

**if** (e == arr[mid]) { // 找到了

**return** mid;

} **else** **if** (e < arr[mid]) { // e 在前一半，调整high

high = mid - 1;

} **else** { //e > arrP[mid]，e在后一半，调整low

low = mid + 1;

}

}

**return** -1; // 没找到返回-1

}

**public** **static** **void** main(String[] args) {

**int** arr[] = {1, 3, 5, 6, 9, 12, 13, 17, 21, 28, 30};

**int** e = 28;

**int** index = *binarySearch*(arr, e);

System.***out***.printf("元素%d在数组中的位置为：%d\n", e, index);

e = 29;

index = *binarySearch*(arr, e);

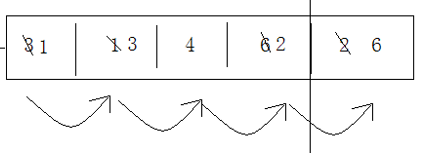
System.***out***.printf("元素%d在数组中的位置为：%d\n", e, index);

}

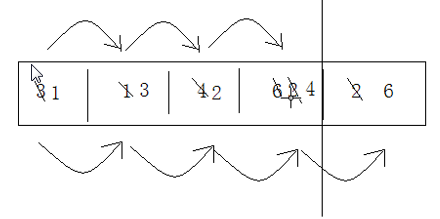
结果为：



## 冒泡排序法



相邻的两个元素进行比较，如果后面一个元素比前一个元素小，则交换位置，如上图，脚标0和脚标1比，比完后，脚标1和脚标2比，一直比到数组最后，这时最大值就确定在数组的最后了，继续这样的操作



可发现第二大的数也确定了，就这样一直比，大的数就像冒泡一样一直放上窜，所以叫冒泡排序法，代码如下：

思路：观察上图，可发现，每比较一次就确定一个大数，这个数就不用比较了，也就是说每次比较会少一个数，那么这个就是递减，我们可以内循环条件跟着外层递减即可达到效果，外循环控制比较的次数，看图可以知道，如果有6个数，那么需要5次确定排序，如果有4个需要3次确定，也就是arr.length-1

**import** java.util.Arrays;

**public** **class** Main {

**public** **static** **void** swap(**int**[] arr,**int** x,**int** y) {

**int** tem = arr[y];

arr[y] = arr[x];

arr[x] = tem;

}

**public** **static** **void** bubbleSort(**int**[] arr) {

**for**(**int** i = arr.length -1;i>0;i--){

**for**(**int** j=0;j<i;j++) {

**if**(arr[j]>arr[j+1]) *swap*(arr, j, j+1);

}

}

}

**public** **static** **void** main(String[] args) {

**int** arr[] = {5, 6, 1, 2, 3, 4};

*bubbleSort*(arr);

System.***out***.println(Arrays.*toString*(arr));

}

}

如果整个数组本来顺序就是OK的，或者在排序的时候一两轮就排好了，如**int** arr[] = {5, 6, 1, 2, 3, 4};这个数组只需要两轮比较就能把5、6排到最后，剩下的1、2、3、4的顺序就不用再排了，但是按照上面的代码还是得走完所有循环，这有点浪费，改进如下：

**public** **static** **void** bubbleSort(**int**[] arr) {

**for**(**int** i = arr.length -1;i>0;i--){

**boolean** hasSwap = **false**; // 是否有交换过

**for**(**int** j=0;j<i;j++) {

**if**(arr[j]>arr[j+1]) {

*swap*(arr, j, j+1);

hasSwap = **true**;

}

}

**if** (!hasSwap) {

**break**; // 如果没有交换过，则说明顺序是OK的，不需要再排了

}

}

}

## 快速排序法

