查找就是在由若干记录组成的集合中找出关键字值与给定值相同的记录。如查找成功，返回找到的记录的信息或者在表中的位置，查找失败就返回一个代表失败的标志。一个查找算法的优劣取决于查找过程中的比较次数，使用平均比较长度(平均比较次数)ASL来衡量查找算法的效率，ASL是和指定值进行比较的关键字的个数的期望值。

$$ASL=\sum_{1}^{n} P_i * C_i$$

其中，Ci表示查找到第i个数据元素时已经比较的次数，Pi是查找表中第i个数据元素的概率。

静态查找和动态查找

根据查找算法是否改变查找表的内容，将查找算法分为静态查找和动态查找。静态查找对查找表查找时，查找成功就返回记录的信息或在查找表中的位置，查找失败就返回一个代表失败的标志，并不对查找表进行插入和删除，或经过一段时间之后再对查找表进行**集中式**的插入和删除操作。动态查找是查找与插入和删除在**同一阶段**进行，例如，在某些问题中，查找成功时，删除查找到的记录，查找失败时，插入被查找的记录。

查找结构

为了提高查找效率，为带查找序列选择合适的数据结构以存储这些数据，这种面向查找的数据结构就称为查找结构。

主要有以下三种数据结构：

1）线性表：适用于静态查找，查找方法有顺序查找和二分查找。

2）树表：适应于动态查找，查找方法是采用二叉排序树进行查找(类似二分查找过程)。

3）哈希表：静态和动态查找均合适，查找方法是哈希技术。

无序查找和有序查找

无序查找要求查找表有序无序均可。有序查找要求查找表必须是有序的。

顺序查找

顺序查找算法是最简单的查找算法，从查找表的一端开始，顺序查找每个记录，直至另一端为止。实现代码如下：

```java
//顺序查找,n是数组长度，a从[0,n-1]
public static int sequenceSearch(int[] a,int value,int n){
    int i=0;
    while (i<n&&a[i]!=value){//比较大小和检查边界同时进行
        ++i;
    }
    if(i<n){
        return i;//查找成功
    }
    else {
        return -1;//查找失败
    }
}
```

查找成功时的平均查找长度是：

$$ASL_{succ}=\frac{1}{n}\sum_{i=1}^{n}i=\frac{1}{n}\times\frac{n(n+1)}{2}=\frac{n+1}{2}$$

查找失败时平均查找长度是（无序表的查找，到最后一个元素才能确定查找失败）：n。

哨兵

为简化边界条件而引入的附加节点（记录）均可称为哨兵，哨兵常用在循环和递归中，用于边界条件的检查。上述顺序查找算法不仅要比较查询表中元素是否与查找值相同，同时还要检测是否越界，因此可以添加哨兵，简化程序，添加哨兵后的代码如下：

```java
//添加哨兵的顺序查找,n是数组最后一个元素的索引，[1,n]是查找表内容，a[0]是哨兵
public static int sequenceSearch2(int[] a,int value,int n){
    int i=n;
    a[0]=value;
    while (a[i]!=value){//只需要比较大小,通过哨兵避免了边界检查(少一条指令,所以查找速度变快了)
        --i;
    }
    if(i>0){
        return i;//查找成功
    }
    else {
        return -1;//查找失败
    }
}
```

哨兵查找成功时的平均查找长度是仍然是（n+1）/2。查找失败时的平均查找长度是n+1(最后需要与哨兵比较，多了一次比较)。在查找表元素个数大于1000时，添加哨兵的查找时间几乎减少了一半。
顺序查找算法对查找表的存储结构没有要求，顺序存储和链式存储均可应用，并且对表中记录的顺序也没有要求，不过，查找效率低下，时间复杂度是O(n)。

折半查找

折半查找要求查找表是有序的，并且是顺序存储，另外，一般应用于静态查找。实现代码如下：

```
1     //折半查找
2     public static int binarySearch(int[] a,int value,int n){
3         /* 边界检查 */
4         int low=0;
5         int high=n-1;
6         //位运算的执行效率优于乘除运算,所以(high-low)/2写成(high-low)>>1会更好
7         //防止溢出.(high+low)/2可能会溢出(当low+high>Integer.Integer.MAX_VALUE时)。
8         int mid = low + ((high-low)>>1);
9         while (low<=high){
10            if(a[mid]==value){
11                return mid;
12            } else if (a[mid]>value){
14                high=mid-1;
15            } else if(a[mid]<value){
17                low=mid+1;
18            }
19            mid=low+(high-low)/2;
20        }
21        return -1;//查找失败
22    }
```

根据折半查找的过程，也就是mid的不断变化，可以生成二叉判定树，通过mid构造这棵判定树，这棵树的左子树节点的值都小于其根节点的值，右子树的值都大于根节点的值，其实，根据折半查找的过程生成的二叉判定树就是一棵二叉排序树。根据已排序的查找表生成二叉查找树(二叉判定树,二叉排序树)的过程如下：

1）对于元素个数为n的查找表，生成的二叉判定树高度(高度从1开始计数)h=⌈log(n+1)⌉，与n个节点的完全二叉树的高度相同。推导过程如下：

对于高度为h的二叉判定树，第一层至第h-1层为满二叉树，所以有：2d-1-1<=n<=2d-1,进而有2d-1<=n+1,n+1<=2d,最后d-1<=log2(n+1)<=d,而d-1和d都是整数，所以有d=⌈log(n+1)⌉。

2）根据二叉判定树得到查找成功时的ASL：

则

$$ASL_{bs}=\frac{1}{n}\sum_{i=1}^{n}C_i$$

$$=\frac{1}{n}(1\times 2^0+2\times 2^1+3\times 2^2+\cdots+h\times 2^{h-1})$$

$$=\frac{1}{n}\sum_{j=1}^{h}(j\cdot 2^{j-1})$$

$$=\frac{1}{n}(h\times 2^h-2^h+1)$$

$$=\frac{n+1}{n}\log_2(n+1)-1$$

当n>50(较大时)，$ASL_{bs}\approx\log_2(n+1)-1$

ASL=1/9*（1+2*2+3*4+4*2）=25/9。

3）查找到外部节点时表示查找失败，外部节点比判定树节点个数多1个，失败时的ASL=1/10*(3*6+4*4)=17/5。

折半查找的最坏性能与平均性能相当接近，其时间复杂度是O(log n)。