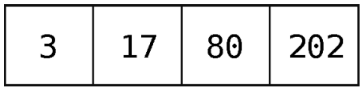
2 算法

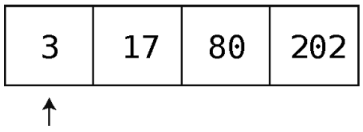
2.1 有序数组

1. 有序数组的值要求保持**有序**
2. 一个有序数组插入的例子 [3,17,80,202] 插入75 需要找到插入的位置

先回顾一下原始的数组。

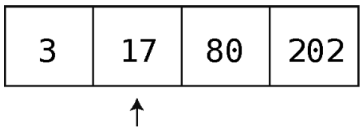


第 1 步：检查索引 0 的值，看 75 应该在它的左边还是右边。



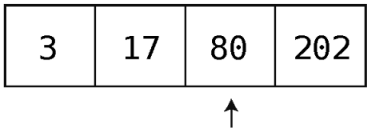
因为 75 大于 3，所以 75 应该在它右边的某个位置。而具体的位置，目前还是不能确定，于是，再检查下一个格子。

第 2 步：检查下一格的值。



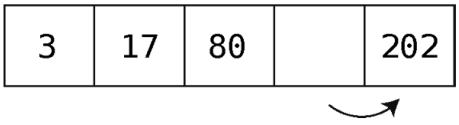
因为 75 大于 17，所以继续。

第 3 步：检查下一格的值。

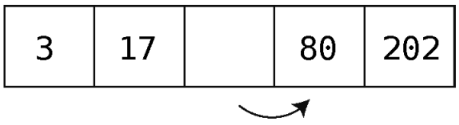


这次是 80，大于 75。因为这是第一次遇到大于 75 的值，可想而知，必须把 75 放在 80 的左侧以使整个数组维持有序。但要在这里插入 75，还得先将它的位置空出来。

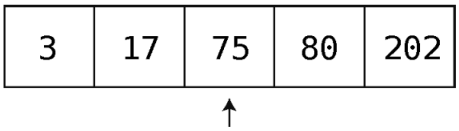
第 4 步：将最后一个值右移。



第 5 步：将倒数第二个值右移。



第 6 步：终于可以把 75 插入到正确的位置上了。



可以看到，往有序数组中插入新值，需要先做一次查找以确定插入的位置。这是它跟常规数组的关键区别（在性能方面）之一。

虽然插入的性能比不上常规数组，但在查找方面，有序数组却有着特殊优势。

2.2 查找有序数组

1. 线性查找：从左到右，逐个查找
2. 以下是用 Ruby 语言实现的有序数组**线性查找**。

```
def linear_search(array, value)
```

```
  # 遍历数组的每一个元素
  array.each do |element|

    # 如果这个元素等于我们要找的值，则将其返回
    if element == value
      return value

    # 如果这个值大于我们要找的值，则提早退出循环
    elsif element > value
      break
    end
  end

  # 如果没找到，则返回空值
  return nil
end
```

2.3 二分查找

1. 有序数组相比常规数组的一大优势就是它除了可以用线性查找，还可以用二分查找。常规数组由于无序，不能使用二分查找。
2. 二分查找的 Ruby 实现

```
def binary_search(array, value)

  # 首先，设定下界和上界，以限定所查之值可能出现的区域。
  # 在开始时，以数组的第一个元素为下界，以最后一个元素为上界

  lower_bound = 0
  upper_bound = array.length - 1

  # 循环检查上界和下界之间的最中间的元素
  while lower_bound <= upper_bound do

    # 如此找出最中间的格子之索引
    #（无须担心商是不是整数，因为 Ruby 总是把两个整数相除所得的小数部分去掉）

    midpoint = (upper_bound + lower_bound) / 2

    # 获取该中间格子的值

    value_at_midpoint = array[midpoint]

    # 如果该值正是我们想查的，那就完事了。
    # 否则，看你是要往上找还是往下找，来调整下界或上界
```

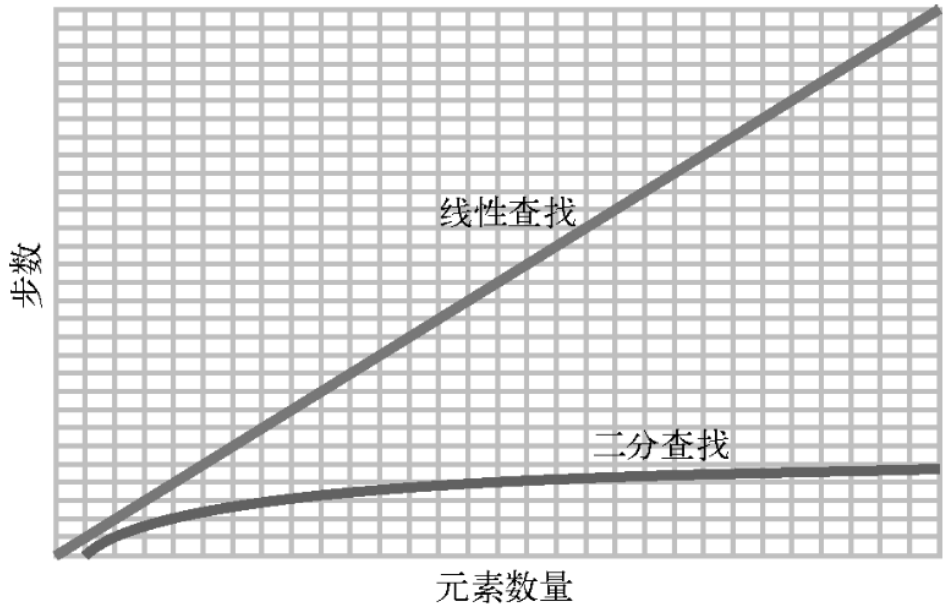
```
    if value < value_at_midpoint
      upper_bound = midpoint - 1
    elsif value > value_at_midpoint
      lower_bound = midpoint + 1
    elsif value == value_at_midpoint
      return midpoint
    end
  end

  # 当下界超越上界，便知数组里并没有我们所要找的值

  return nil
end
```

2.4 二分查找与线性查找

1. 对于线性查找：对多步数=元素个数
2. 对于二分查找：数组长度翻倍，二分查找增加1步
3. 线性查找VS二分查找



4. 有序数组并不是所有操作都比常规数组要快。插入操作相对慢一点，查找相对快一点。