

二分查找

二分查找是指在一个递增或递减的数组结构中，查找一个元素是否存在的方式。

比如一个递增的数组，先找到数组中间位置，如果比数组中间位置数值比目标大，证明目标在中间位置左边，再依次继续查找左边的中间位置，实现每次减半的目标，称为二分查找。

比如查找一个数组 `[1, 2, 3, 6, 10, 15, 16, 20, 23, 43, 45, 50, 55, 60, 70]` 中是否有 `45`，就可以使用二分查找方法。

```
// 二分查找，循环
function binarySearch1(arr: number[], target: number): number {
  let len = arr.length;
  if (!len) {
    return -1;
  }
  let startIndex = 0;
  let endIndex = len - 1;
  // 二分法，时间复杂度 O(logn)
  while(startIndex <= endIndex) {
    let midIndex = Math.floor((startIndex + endIndex) / 2);
    let midValue = arr[midIndex];
    if (target > midValue) {
      // 目标元素大于中间位置的值，在右边继续查找
      startIndex = midIndex + 1;
    } else if (target === midValue) {
      return midIndex;
    } else {
      endIndex = midIndex - 1;
    }
  }
  return -1;
}

let arr = [];
for (let i = 0; i < 100 * 10000; i++) {
  arr.push(i);
}

console.time("耗时")
binarySearch1(arr, 450000); // 0.13ms
console.timeEnd("耗时")

// indexOf 方法
let arr2 = [];
for (let i = 0; i < 100 * 10000; i++) {
  arr2.push(i);
}

console.time("耗时2")
arr2.indexOf(450000); // 0.84ms 时间复杂度 O(n)
console.timeEnd("耗时2")
```

凡有序，必二分。凡是有序数据结构操作，使用二分查找算法。凡二分，必包含 $O(\log n)$ 。二分查找算法时间复杂度是 $O(\log n)$