暨南大学本科上机课实验报告

学生姓名：肖健成　　学号：2021103268

学院：网络空间安全　　专业：网络空间安全

课程名称:《算法分析与设计》　　实验名称：

实验类型：验证-设计-综合　　实验时间：2023 年 10 月 20 日晚　　地点：517

上机内容：
(1) 设计利用递归与分治策略设计出二分查找算法:
(2)设计具有数据输入、处理和输出功能的二分查找算法，实现算法的编译和运行，记录实验过程并整理实验结果

算法描述：
二分查找算法（Binary Search）是一种高效的搜索算法，用于在有序数组中查找特定元素的位置。它的原理是将数组划分成两部分，然后确定目标元素位于哪一部分，然后继续在该部分中查找，重复这个过程直到找到目标元素或确定它不存在。
下面是对二分查找算法的描述：
工作原理：
1.需要在有序数组中查找目标元素。
2.选择数组的中间元素作为比较的基准。
3.如果中间元素等于目标元素，搜索成功并返回其位置。
4.如果中间元素大于目标元素，说明目标元素位于基准元素的左侧，将搜索范围缩小到左半部分，然后重复步骤 2。
5.如果中间元素小于目标元素，说明目标元素位于基准元素的右侧，将搜索范围缩小到右半部分，然后重复步骤 2。
6.重复上述过程直到找到目标元素或确定它不存在。
特点：
二分查找算法是一种高效的搜索算法，时间复杂度为 O(log n)，其中 n 是数组的长度。这使得它在大型数据集中非常快速。
1.它要求输入数组必须是有序的，否则无法保证正确的结果。
2.二分查找是一种分治策略的应用，每次都将搜索范围缩小一半。
3.由于每次只需要比较中间元素，它的比较次数较少。
优点：
高效：在有序数组中查找元素的性能非常出色，尤其在大型数据集中。
简单：算法的基本思想非常简单，易于理解和实现。
局限性：
1.数组必须是有序的，如果不是，需要预先进行排序。
2.二分查找只适用于静态数据结构，即一旦构建数组，就不能再插入或删除元素。

3.对于小型数据集，线性查找可能更快。
应用：
二分查找常用于数据库查询、查找特定元素、排名问题等需要快速查找的领域。
它还被广泛用于算法和数据结构领域，作为其他算法的基础组件。
在实现二分查找算法中，我们使用了递归，分治的思想进行实现。

二分查找算法的伪代码

```
BinarySearch(arr, target):
    left = 0
    right = length(arr) - 1

    while left <= right:
        mid = (left + right) / 2

        if arr[mid] == target:
            return mid   # 目标元素已找到，返回索引
        elif arr[mid] < target:
            left = mid + 1   # 目标元素位于右半部分
        else:
            right = mid - 1   # 目标元素位于左半部分

    return -1   # 目标元素不存在于数组中
```

源程序：

实现二分查找算法方法 1：递归实现

```python
def binary_search(arr, target, low, high):
    """
    使用二分查找算法在有序数组中查找目标元素。

    Args:
        arr (list): 有序数组
        target: 要查找的目标元素
        low (int): 当前搜索范围的起始索引
        high (int): 当前搜索范围的结束索引

    Returns:
        int: 目标元素的索引（如果找到），-1（如果不存在）
    """
    if low > high:
        return -1  # 目标元素不在数组中
```

```
    mid = (low + high) // 2  # 计算中间索引

    if arr[mid] == target:
        return mid  # 找到目标元素，返回索引
    elif arr[mid] > target:
        return binary_search(arr, target, low, mid - 1)  # 在左半部
分继续查找
    else:
        return binary_search(arr, target, mid + 1, high)  # 在右半
部分继续查找

# 示例用法
if __name__ == "__main__":
    sorted_list = [1, 3, 5, 7, 9, 11, 13, 15]
    print("原数组:", sorted_list)
    target = 7
    print("目标值:", target)
    result = binary_search(sorted_list, target, 0, len(sorted_list)
- 1)
    if result != -1:
        print(f"目标元素 {target} 在索引 {result} 处找到。")
    else:
        print(f"目标元素 {target} 未找到。")
```

实现二分查找算法方法 2：分治实现

```
def binary_search(arr, target):
    """
    使用二分查找算法在有序数组中查找目标元素。

    Args:
        arr (list): 有序数组
        target: 要查找的目标元素

    Returns:
        int: 目标元素的索引（如果找到），-1（如果不存在）
    """
    left, right = 0, len(arr) - 1  # 初始化左边界和右边界

    while left <= right:
        mid = (left + right) // 2  # 计算中间索引

        if arr[mid] == target:
            return mid  # 找到目标元素，返回索引
        elif arr[mid] < target:
```

```
            left = mid + 1  # 更新左边界，继续在右半部分查找
        else:
            right = mid - 1  # 更新右边界，继续在左半部分查找

    return -1  # 如果找不到目标元素，返回-1

# 示例用法
if __name__ == "__main__":
    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    print("原数组:", arr)
    target = 10
    print("目标值:", target)
    result = binary_search(arr, target)

    if result != -1:
        print(f"目标元素 {target} 在索引 {result} 处找到。")
    else:
        print(f"目标元素 {target} 未找到。")
```

设计具有数据输入、处理和输出功能的二分查找算法

```
def is_sorted(arr):
    """
    检查数组是否按升序排列。

    Args:
        arr (list): 待检查的数组

    Returns:
        bool: 如果数组按升序排列，返回 True；否则返回 False。
    """
    for i in range(1, len(arr)):
        if arr[i] < arr[i - 1]:
            return False
    return True

def binary_search(arr, target):
    """
    使用二分查找算法在有序数组中查找目标元素。

    Args:
        arr (list): 有序数组
        target: 要查找的目标元素

    Returns:
```

```
        int: 目标元素的索引（如果找到），-1（如果不存在）
    """
    left, right = 0, len(arr) - 1  # 初始化左边界和右边界

    while left <= right:
        mid = (left + right) // 2  # 计算中间索引

        if arr[mid] == target:
            return mid  # 找到目标元素，返回索引
        elif arr[mid] < target:
            left = mid + 1  # 更新左边界，继续在右半部分查找
        else:
            right = mid - 1  # 更新右边界，继续在左半部分查找

    return -1  # 如果找不到目标元素，返回-1

# 输入数据
arr = []
n = int(input("请输入数组的长度: "))
for i in range(n):
    element = int(input(f"请输入第 {i+1} 个元素: "))
    arr.append(element)

# 检查数组是否升序
if not is_sorted(arr):
    print("输入的数组不是升序数组，无法进行查找。")
else:
    target = int(input("请输入要查找的目标元素: "))

    # 调用二分查找算法
    result = binary_search(arr, target)
    print("原数组:", arr)
    print("目标值:", target)
    # 输出结果
    if result != -1:
        print(f"目标元素 {target} 在索引 {result} 处找到。")
    else:
        print(f"目标元素 {target} 未找到。")
```

复杂度分析：
二分查找算法
时间复杂度分析：
最坏情况：在最坏情况下，二分查找算法需要进行 O(log n)次比较，其中 n 是数组的长度。这是因为每次都将搜索范围缩小为当前范围的一半，直到找到目标元素或搜索范围为空。
平均情况：在平均情况下，二分查找也需要进行 O(log n)次比较。这是因为二分查找将搜索范围减半，因此搜索的深度是以 2 为底的对数。
空间复杂度分析：
二分查找算法的空间复杂度非常低，为 O(1)，因为它只需要少量额外的存储空间来保存左右边界和中间索引等变量。不会随着问题规模的增加而增加额外的内存消耗。

程序的运行结果及截图：
实现二分查找算法方法 1：递归实现

```
原数组: [1, 3, 5, 7, 9, 11, 13, 15]
目标值: 7
目标元素 7 在索引 3 处找到。
```

实现二分查找算法方法 2：分治实现

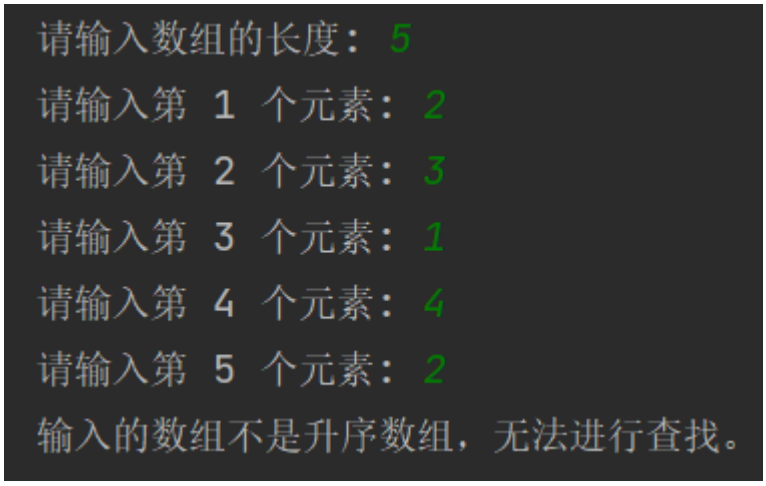
```
原数组: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
目标值: 10
目标元素 10 在索引 9 处找到。
```

设计具有数据输入、处理和输出功能的二分查找算法
正常运行结果

```
请输入数组的长度： 5
请输入第 1 个元素： 1
请输入第 2 个元素： 2
请输入第 3 个元素： 3
请输入第 4 个元素： 4
请输入第 5 个元素： 5
请输入要查找的目标元素： 1
原数组：[1, 2, 3, 4, 5]
目标值：1
目标元素 1 在索引 0 处找到。
```

```
请输入数组的长度： 5
请输入第 1 个元素： 1
请输入第 2 个元素： 2
请输入第 3 个元素： 3
请输入第 4 个元素： 4
请输入第 5 个元素： 5
请输入要查找的目标元素： 6
原数组：[1, 2, 3, 4, 5]
目标值：6
目标元素 6 未找到。
```

非升序数组



个人总结：
算法原理：
二分查找算法基于分治策略，通过将搜索范围逐步减半来查找目标元素。
1.首先，确定数组的左边界和右边界，然后计算中间索引。
2.比较中间元素与目标元素，如果相等则返回中间索引；如果小于目标元素，则更新左边界；如果大于目标元素，则更新右边界。
3.重复上述过程，直到找到目标元素或搜索范围为空。
时间复杂度：
最坏情况和平均情况下，二分查找的时间复杂度均为 O(log n)，其中 n 是数组的长度。这是因为每次比较都将搜索范围减半，搜索的深度是以 2 为底的对数。
空间复杂度：
二分查找的空间复杂度非常低，为 O(1)。它只需要常数级别的额外内存来保存辅助变量。
稳定性：
二分查找是一种稳定的查找算法，即在有多个相同值的情况下，总是返回最左侧的目标元素的索引。
适用性：
二分查找适用于有序数组，无论是升序还是降序排列。
它通常用于解决在大型数据集中查找目标元素的问题，因为它的时间复杂度相对较低。
注意事项：
数组必须是有序的，否则二分查找不适用。
二分查找返回的是目标元素的索引，如果目标元素可能存在多次，则返回最左侧的索引。
总之，二分查找算法是一种非常高效的查找算法，适用于有序数组中查找目标元素。它的时间复杂度和空间复杂度都非常低，使其成为解决大型数据集中查找问题的理想选择。