算法导论 课程设计

成绩	

题　　目：分治算法

学院班级：1613012

学　　号：16130120191

姓　　名：罗阳豪

主讲教师：覃桂敏

日　　期：2019 年 03 月 21 日

目录

题目一 Exercise 2.3-7

1.1 实验题目

描述一个时间复杂度为θ(nlgn)的算法如下，给定一个含 n 个整数的集合 S 和一个整数 x，确定 S 中是否存在两个元素的总和为 x

1.2 实验目的

加深对于分治算法的理解，掌握构思一个分治算法的流程，理解分治的思想和应用场景

1.3 实验设计与分析

假定整数集合 S 按元素大小升序排列为 S = {d1, d2, ..., dn}，并且其中含有 di, dj（i < j），di + dj = x

很容易发现

若 d1 + dn < x，则 j < n，即 di, dj ∈ {d1, d2, ..., dn-1}

若 d1 + dn > x，则 i > 1，即 di, dj ∈ {d2, d3, ...,dn}

否则 di = d1，dj = dn

所以只要循环地判断 S 的最大和最小元素之和，然后根据结果从 S 中移除最大或最小元素。直到发现 di + dj = x，或发现不存在整数对之和为 x

很容易可以想到，这个算法时间复杂度为 O(n)

1.4 实验环境

Python 3.7.0

1.5 项目测试

功能测试：

输入：

S1 = {9, 7, 3, 8, 3, 4, 12, 123, 5, 12], x1 = 8;

S2 = {2, 4, 5, 1}, x2 = 8; S3 = {}, x3 = 8

输出：

R1 = (True, (3, 5));

R2 = (False, None);

R3 = (False, None)

上述输出结果全部正确

题目二 优先队列

2.1 实验题目

实现优先队列

- MAXIMUM(S) - 返回最大元素
- EXTRACT-MAX(S) - 取出最大元素；（返回并从队列中删除）
- INCREASE-KEY(S, x, k) - 增加某一元素的值
- INSERT(S, x) - 插入一个新元素

2.2 实验目的

理解堆的数据结构，掌握使用堆实现优先队列的方法，掌握大顶堆和最大优先队列的实现

2.3 实验设计与分析

使用大顶堆的性质，可以很容易实现最大优先队列（同理用小顶堆可以实现最小优先队列）。从大顶堆堆顶可以很容易取出队列中最大的元素，同时，维持堆性质（以便下一次取最大元素），队尾插入新元素，都很方便。

2.4 实验环境

Python 3.7.0

2.5 项目测试

功能测试：

执行以下操作：

通过序列{1, 2, 3, 4, 5, 6, 7, 12, 1, 2, 3}初始化最大优先队列

打印优先队列

打印优先队列最大元素（MAXIMUM）

给第 7 个元素增加 10（INCREASE-KEY），打印优先队列

插入值为 24 的元素（INSERT），打印优先队列

循环取出队列最大元素（EXTRACT-MAX）

输出结果

[12, 5, 7, 4, 3, 6, 3, 2, 1, 2, 1]

12

[13, 5, 12, 4, 3, 6, 7, 2, 1, 2, 1]

[24, 5, 13, 4, 3, 12, 7, 2, 1, 2, 1, 6]

24 13 12 7 6 5 4 3 2 2 1 1

上述结果与预期全部一致

由于对于由大顶堆实现的最大优先队列的性能分析的相关的理论已经非常充分，便不再在此做性能测试

题目三 快速排序

3.1 实验题目

实现快速排序，并回答以下问题

1. 在一个含有 n 铬元素且所有元素值相等的列表上执行快速排序将会进行多少次比较？
2. 在一个含有 n 个元素的列表上执行快速排序,最大和最小的比较此时分别是多少，分别举一个例子。

3.2 实验目的

了解快速排序算法，熟悉快速排序算法的实现，掌握快速排序的几种常用优化，熟知快速排序算法的性能和会导致性能问题的特殊情况

3.3 实验设计与分析

快速排序也使用了分治的思想，每一次递归中，确定一个元素的位置，并将无序串分为两个子串，其中左边子串中每一个元素都不大于确定元素，右边子串每个元素都不小于确定元素。

对两个子串递归使用该算法，直到最终子串没有元素

为了克服快排较常出现的一些较坏情况，我还对算法做了以下调整：

1. 从待排序列中随机选择一个元素作为参考元素，避免遇到有序序列或部分有序序列时出现最坏或较坏情况
2. 收集待排序列中与参考元素值相等的元素，使其执行一轮递归后其位置都在参考元素两边，并在下一轮递归中被跳过。避免遇到大量相同元素时出现较坏情况。

3.4 实验环境

Python 3.7.0

3.5 项目测试

由于排序的功能十分简单且是周知的，便不再做功能测试的展示

性能测试：

测试分别对 1 万、10 万、100 万个 64 位浮点数进行排序操作，用时分别是

0.1497s

1.4944s

14.034s

作为对照，语言自带的排序方法对 100 万数据排序用时 0.6225s

可见该算法时间复杂度接近 O(nlgn)，性能比语言内建的排序方法略慢，估计是因为底层优化的问题

题目四 第 k 大元素

4.1 实验题目

给出以下问题的分治算法：给定两个有序且大小分别为 m 和 n 的列表，并且对于每个列表，获取第 i 个元素只需要消耗单位时间。请给出一个时间复杂度为 O(lgm + lgn)的算法，计算这两个列表的联合中的第 k 大元素。（为简单起见，你可以认为这两个列表中的元素是相异的）

4.2 实验目的

加深对于分治算法的理解，掌握构思一个分治算法的流程，理解分治的思想和应用场景

4.3 实验设计与分析

我们首先假定第 k 大元素在序列 m 中，

那么我们使用折半查找的方法

首先判断 m 中第 m/2 的元素是等于、大于还是小于 m、n 中第 k 大元素，然后分别选择输出结果、检查第 m/4 的元素或检查第 3m/4 的元素。直到找到 m、n 中第 k 大元素或发现 m、n 中第 k 大元素不在 m 中，则对 n 做同样操作。相关逻辑伪代码如下

```
THE_KTH_LARGEST(m, n, k):
    l_cursor = 1
    r_cursor = m.length
    while l_cursor < r_cursor:
        cmp_k = COMPARE_TO_K((l_cursor + r_cursor) / 2, m, n, k)
        if cmp_k == 0:
            return m[(l_cursor+r_cursor)/2]
        else if cmp_k < 0:
            r_cursor = (l_cursor + r_cursor) / 2
        else:
            l_cursor = (l_cursor + r_cursor) / 2
    return THE_KTH_LARGEST(n, m, k)
```

该算法依赖方法 COMPARE_TO_K 来计算给定元素与第 k 大元素的大小关系，下面给出时间复杂度为 O(1)的该函数的一个实现的伪代码

```
COMPARE_TO_K(i, m, n, k):
    if m[i] < n[k-i+1]:
        return -1
    else if m[i] > n[k-i+2]:
        return 1
    else:
        return 0
```

该方法利用了 m、n 都是有序数列这一特征，假定 m 中第 i 个元素是 m、n 中第 k 大元素，则他在 n 中应该位于第 k-i+1 与 k-i+2 之间。通过与 n 中这两个数比较，可以确定 m 中第 i 个元素与 m、n 中第 k 大元素的大小关系。

4.4 实验环境

Python 3.7.0

4.5 项目测试

输入：

m = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

n = {2, 3, 4, 6, 10, 20, 100}

k = 8

输出：

5