lab6 排序算法

姓名：王天一

学号：320200931301

日期：2022年6月22日

1 主要问题

给定一个序列，对其排序。

2 算法思路

实现了三个算法，分别为：

- 冒泡排序
- 快速排序
- 桶排序

2.1 冒泡排序

2.1.1 算法分析

冒泡排序 是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果它们的顺序错误就把它们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

- 步骤1: 比较相邻的元素。如果第一个比第二个大，就交换它们两个；
- 步骤2: 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数；
- 步骤3: 针对所有的元素重复以上的步骤，除了最后一个；
- 步骤4: 重复步骤1~3，直到排序完成。
- 最佳情况：$T(n) = O(n)$
- 最差情况：$T(n) = O(n2)$
- 平均情况：$T(n) = O(n2)$

2.1.2 代码

```
def bubbleSort(alist):
        for passnum in range(len(alist)-1,0,-1):
            for i in range(passnum):
                if alist[i]>alist[i+1]:
                    temp = alist[i]
                    alist[i] = alist[i+1]
                    alist[i+1] = temp
        return alist
```

2.2 快速排序

2.2.1 算法分析

快速排序 的基本思想：通过一趟排序将待排记录分隔成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

快速排序使用分治法来把一个串（list）分为两个子串（sub-lists）。具体算法描述如下：

- 步骤1：从数列中挑出一个元素，称为 “基准”（pivot ）；
- 步骤2：重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；
- 步骤3：递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。
- 最佳情况：$T(n) = O(nlogn)$
- 最差情况：$T(n) = O(n^2)$
- 平均情况：$T(n) = O(nlog_n)$

2.2.2 代码

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
    return alist

def partition(arr,low,high):
    i = ( low-1 )         # 最小元素索引
    pivot = arr[high]

    for j in range(low , high):

        # 当前元素小于或等于 pivot
        if   arr[j] <= pivot:

            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

def quick_sort(lists,i,j):
    if i >= j:
        return list
    pivot = lists[i]
    low = i
    high = j
```

```
    while i < j:
        while i < j and lists[j] >= pivot:
            j -= 1
        lists[i]=lists[j]
        while i < j and lists[i] <=pivot:
            i += 1
        lists[j]=lists[i]
    lists[j] = pivot
    quick_sort(lists,low,i-1)
    quick_sort(lists,i+1,high)
    return lists
```

2.3 桶排序

2.3.1 算法分析

桶排序 是计数排序的升级版。它利用了函数的映射关系，高效与否的关键就在于这个映射函数的确定。

桶排序 (Bucket sort)的工作的原理：

假设输入数据服从均匀分布，将数据分到有限数量的桶里，每个桶再分别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排

- 步骤1：人为设置一个BucketSize，作为每个桶所能放置多少个不同数值（例如当BucketSize==5时，该桶可以存放 {1,2,3,4,5} 这几种数字，但是容量不限，即可以存放100个3）；
- 步骤2：遍历输入数据，并且把数据一个一个放到对应的桶里去；
- 步骤3：对每个不是空的桶进行排序，可以使用其它排序方法，也可以递归使用桶排序；
- 步骤4：从不是空的桶里把排好序的数据拼接起来。

注意，如果递归使用桶排序为各个桶排序，则当桶数量为1时要手动减小BucketSize增加下一循环桶的数量，否则会陷入死循环，导致内存溢出。

桶排序最好情况下使用线性时间O(n)，桶排序的时间复杂度，取决与对各个桶之间数据进行排序的时间复杂度，因为其它部分的时间复杂度都为O(n)。很显然，桶划分的越小，各个桶之间的数据越少，排序所用的时间也会越少。但相应的空间消耗就会增大。

- 最佳情况：$T(n) = O(n + k)$
- 最差情况：$T(n) = O(n + k)$
- 平均情况：$T(n) = O(n^2)$

2.3.2 代码

```
def Bucket_Sort(array, bucketsize):
    minValue = min(array)
    maxValue = max(array)
    res = []
    bucketcount = (maxValue - minValue + 1) // bucketsize
    bucket_lists = [[] for i in range(bucketcount+1)]

```

```
    for i in array:
        bucket_index = (i - minValue) // bucketsize
        bucket_lists[bucket_index].append(i)
    # 桶内排序
    for j in bucket_lists:
        quick_sort(j, 0, len(j)-1)

    for j in bucket_lists:
        if len(j) != 0:
            res.extend(j)
    return res
```

3 测试用例

待排序序列：

```
5 4 3 2 1
```

测试结果：

```
PS C:\Users\wty02> & C:/Users/wty02/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/wty02/Desktop/数据结构实验/Data_Structure_Lab/lab6/lab6.py
原始序列:  [5, 4, 3, 2, 1]
冒泡排序:  [1, 2, 3, 4, 5]
快速排序:  [1, 2, 3, 4, 5]
桶排序:  [1, 2, 3, 4, 5]
```