简析快速排序中值选取方法对排序效率的影响

地球与空间科学学院　王冠力　1600012626

课件中的快排对应文件 quickSort_raw.py，其中值默认选取的是列表首项。自定义的快排对应文件 quickSort.py，其中中值选取类似于三点选取法，为 alist[mid],alist[midleft],alist[midright]的中位数,mid = (first + last) // 2，midleft = (first + mid) // 2，midright = (mid + last) // 2。测试代码对应文件 quickSort_test.py。

两个快排相比，后者多出了中值选取的开销，但在一定程度上可以减少分裂不均的现象。下面分析一下对比测试的情况。对于如下测试代码：

```
#测试代码：考察中值选取方法随排序效率和稳定性的影响
from quickSort_raw import quickSort_raw
from quickSort import quickSort
import random
from timeit import Timer

#先对比快排调用的时间及其稳定性
print("Testing time consumed on importing:")
for i in range(3):        #..........共三组对比
    t10 = Timer("from __main__ import quickSort_raw").timeit(number=1)
    t20 = Timer("from __main__ import quickSort").timeit(number=1)
    print("Group %d:\n" % (i+1), "t10 = %f\n" % t10, "t20 = %f" % t20, sep='      ')

#再对比排序时间
print("\nTesting time consumed on sorting:")
scale = 1000     #..........用于控制排序的数据规模
for k in range(7):      #..........共七组对比
    raw = tuple([random.randrange(scale) for i in range(scale)])   # .........原始序列用不可变类型存储
    lst1 = list(raw)
    t1 = Timer("quickSort_raw(lst1)", "from __main__ import quickSort_raw, lst1").timeit(number=1)
    lst2 = list(raw)
    t2 = Timer("quickSort(lst2)", "from __main__ import quickSort, lst2").timeit(number=1)
    print("Group %d:\n" % (k + 1), "t1 = %.17f\n" % t1, "t2 = %.17f" % t2, sep='      ')
```

图 1

首先考察了调用快排的时间开销，结果显示这一开销保持在 10^{-5} 数量级上，且在小范围内波动。考虑到后续排序本身占用的时间远远大于 10^{-5} 数量级，可以认为这一因素几乎没有影响。（图 2）

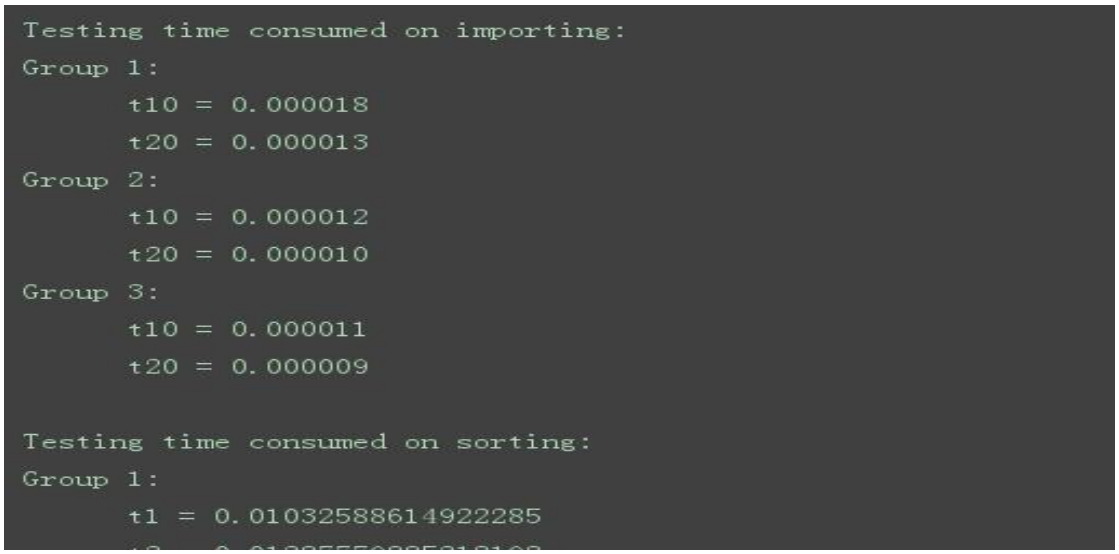

图 2

接下来，对于规模为 1000 的同一随机数列表，分别用两个快排进行排序并对比时间；共进行了七组。结果如下（图 3、图 4）：

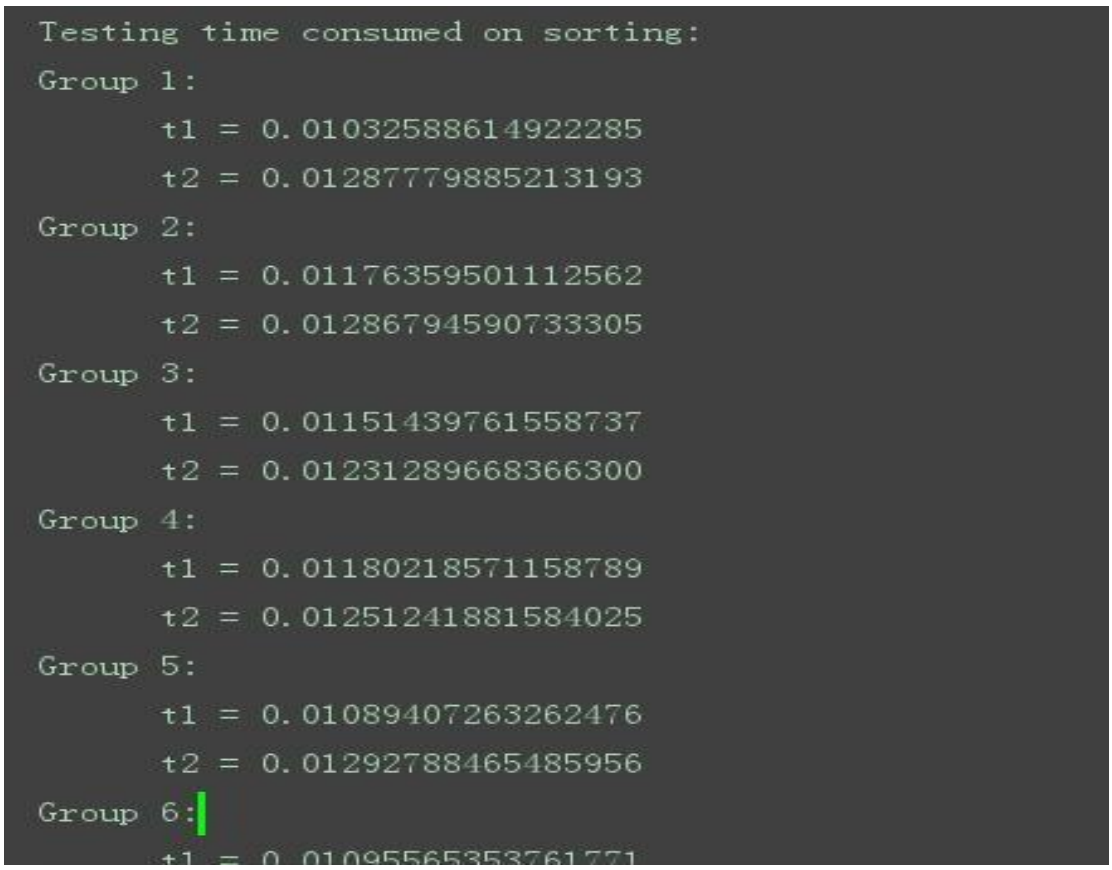

图 3

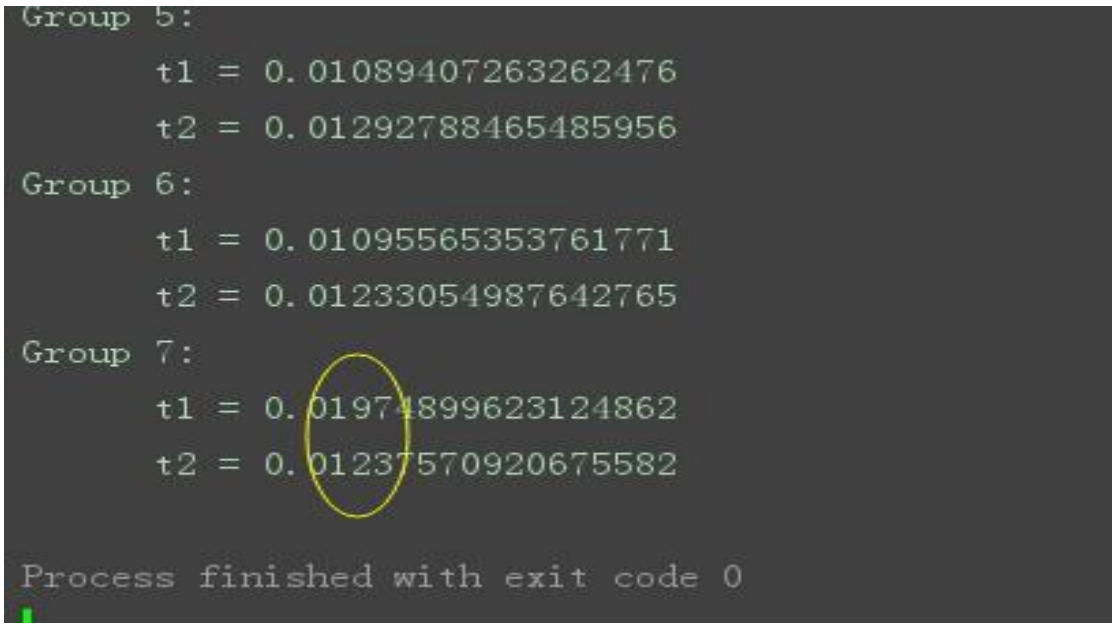

图 4

t1 代表中值为首项的快排所用的时间；t2 代表自定义中值选取的快排所用的时间。七组对比中前六组均是 t1<t2，大概相差 0.001-0.002 s，说明中值设为首项的快排在多数情况下更快一些（没有额外的中值选取上的时间开销）；但是对于第七组数据，t1>t2，且二者相差 0.007s，说明对于这一组的待排序列表，采用自定义的中值选取方法有效地避免了分裂不均的情况，大大提高了排序效率。

我们再来看一个更极端的例子：

```
    #再对比排序时间
    print("\nTesting time consumed on sorting:")
    scale = 1000     #.....用于控制排序的数据规模
    for k in range(1):      #.....共七组对比
        raw = tuple([1000 - i for i in range(1000)])
        print(raw)
        #raw = tuple([random.randrange(scale) for i in range(scale)])   #.....原始序列用不可变类型存储
        lst1 = list(raw)
        print("lst1, before sorting:\n", lst1)
        t1 = Timer("quickSort_raw(lst1)", "from __main__ import quickSort_raw, lst1").timeit(number=1
        print("lst1, after sorting:\n", lst1)
        lst2 = list(raw)
        print("lst2, before sorting:\n", lst2)
        t2 = Timer("quickSort(lst2)", "from __main__ import quickSort, lst2").timeit(number=1)
        print("lst2, after sorting:\n", lst2)
        print("Group %d:\n" % (k + 1), "t1 = %.17f\n" % t1, "t2 = %.17f" % t2, sep='      ')
```

图 5

取定 raw = （1000，999，998，…，1）（图 5）；这时两个快排所用的时间对比如下（图 6）：

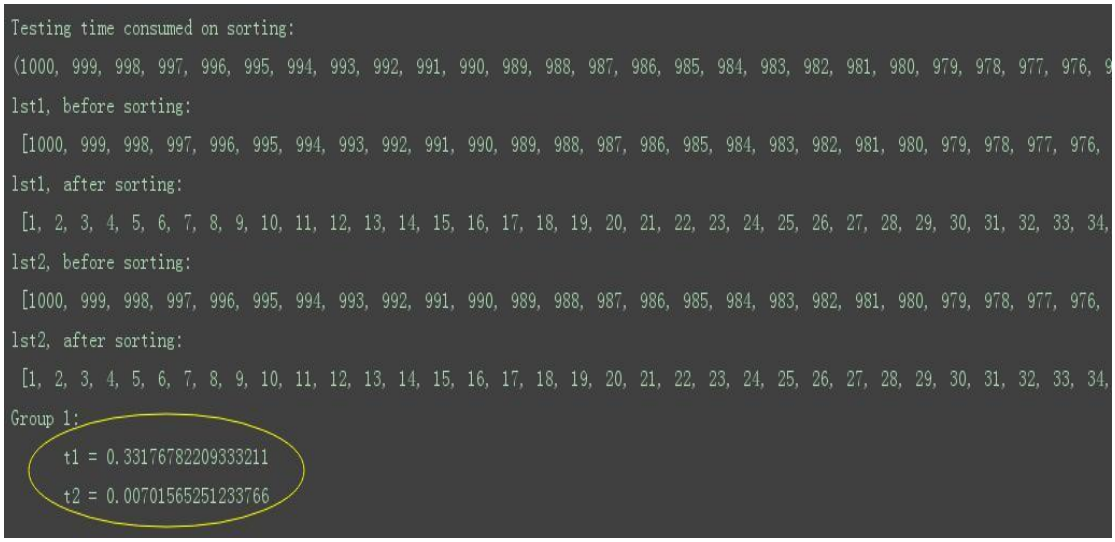

图 6

t1 >> t2。可见对于某些特殊序列，中值选取的方法会在很大程度上影响排序的效率。

综上所述，适当调整中值选取的方法，虽然会有不可避免的额外时间开销，但能够有效地减少分裂不均的情况，使排序算法的整体效率得到提升。