2020 年 春 季学期研究生课程考核

实验报告（四）

考 核 科 目：高级算法设计与分析

学生所在院（系）：计算机科学与技术学院

学 生 所 在 学 科：计算机科学与技术

学 生 姓 名：于晟健

学 号：19S003037

学 生 类 别：全日制学术型硕士研究生

考 核 结 果　　阅 卷 人

实验 4　快速排序

4.1 实验目的

（1）掌握快速排序随机算法的设计思想与方法；
（2）熟练使用高级编程语言实现不同的快速排序算法；
（3）利用实验测试给出不同快速排序算法的性能以理解其优缺点；

4.2 实验学时

4 学时。

4.3 实验问题

快速排序是算法导论中的经典算法。在本实验中，给定一个长为 N 的整数数组，要求将数组升序排序。

4.4 实验步骤

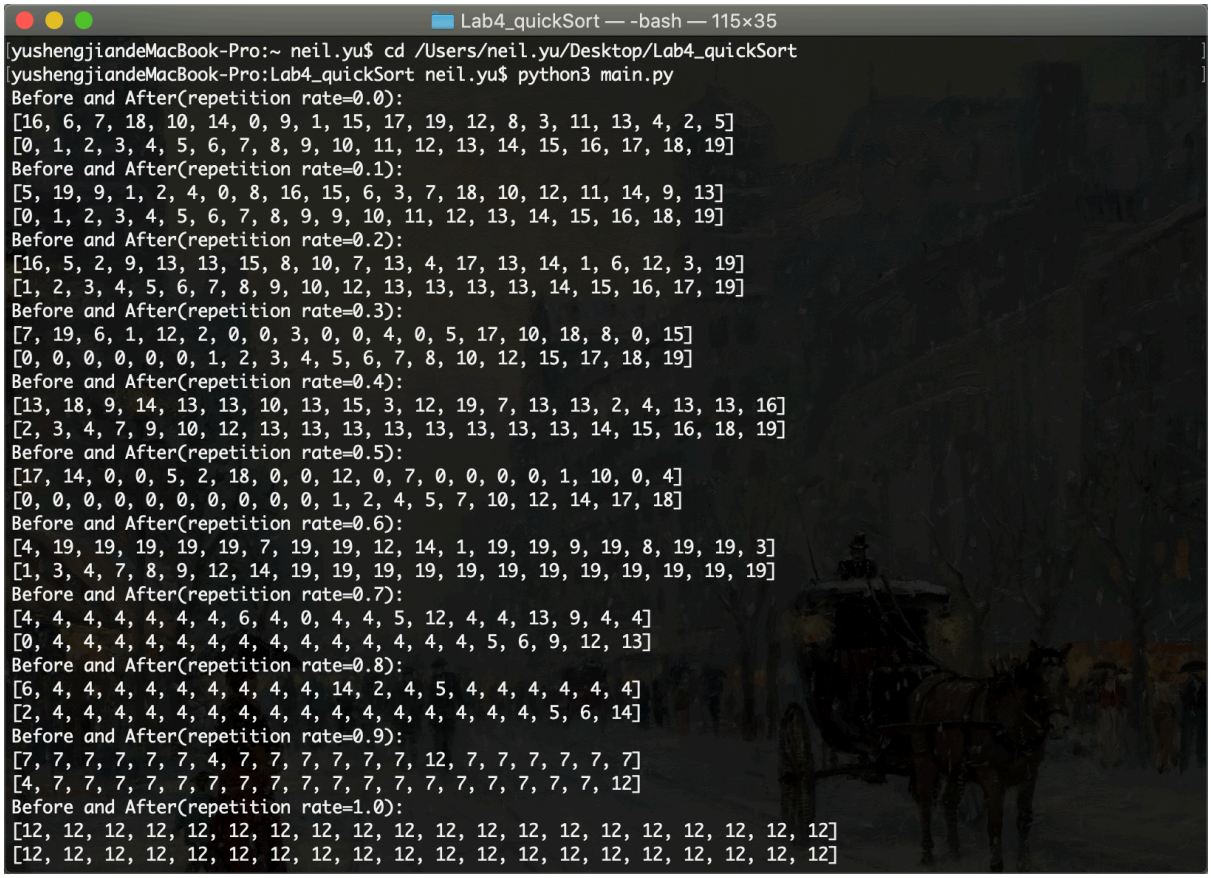
4.4.1 按照算法导论中给出的伪代码实现快速排序

```
QuickSort (A, p, r)
    if p < r
        q = Rand_Partition (A, p, r)
        QuickSort (A, p, q-1)
        QuickSort (A, q+1, r)
Rand_Partition (A, p, r)
    i = Random (p, r)
    exchange A[r] with A[i]
    x = A[r]
    i = p – 1
    for j = p to r – 1
        if A[j] <= x
        i = i + 1
        exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i + 1
```

首先，我们实现图上伪代码所示的算法，代码如下：

```python
def quickSort(data: list, p: int, r: int) -> None:
    if p < r:
        q = Rand_Partition(data, p, r)
        quickSort(data, p, q - 1)
        quickSort(data, q + 1, r)


def Rand_Partition(data: list, p: int, r: int) -> int:
    i = random.randint(p, r)
    data[i], data[r] = data[r], data[i]
    pivot = data[r]
    i = p - 1
    for j in range(p, r):
        if data[j] <= pivot:
            i = i + 1
            data[i], data[j] = data[j], data[i]
    data[i + 1], data[r] = data[r], data[i + 1]

    return i + 1
```

其后，我们在 N=20 的测试样例上进行测试，结果如下：



显然，输出结果是正确的。

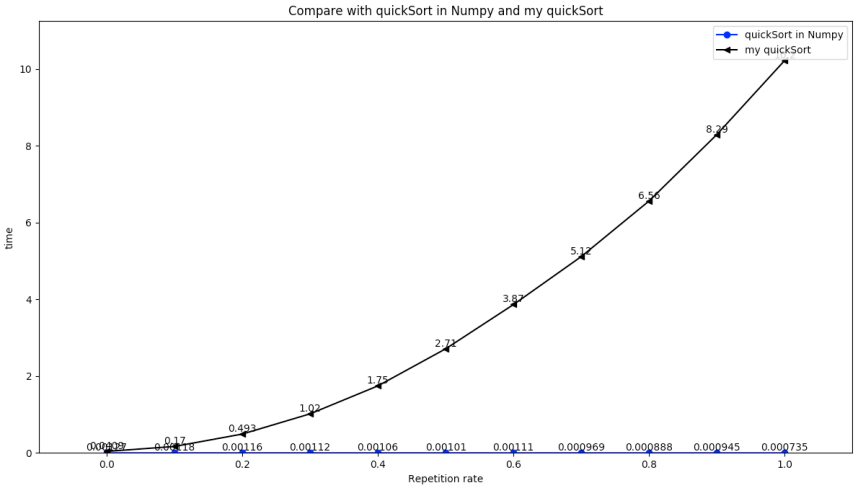
4.4.2 测试算法在不同输入下的表现

生成 11 个大小为 10^6 的整数数据集，第i个子集中存在元素重复 $10^6\times10\times i\%$，i=0,1,2,⋯,10，算法代码如下：

```python
def dataGeneration(N: int, rate: int) -> list:
    """

    :param N: the number of data
    :param rate: 0.1*rate denote the repetition rate of data
    :return:
    """
    if rate == 10:
        return [random.randint(0, N)] * N
    if rate == 0:
        return random.sample(range(0, N), N)

    M = int(N * 0.1 * rate)
    no_repeat = random.sample(range(0, N), N - M + 1)
    # select N-M numbers from no_repeat
    index = random.randint(0, len(no_repeat) - 1)
    repeat = [no_repeat[index]] * (M - 1)
    # disorder the data
    data = no_repeat + repeat
    random.shuffle(data)

    return data
```

4.5 实验结果和算法改进

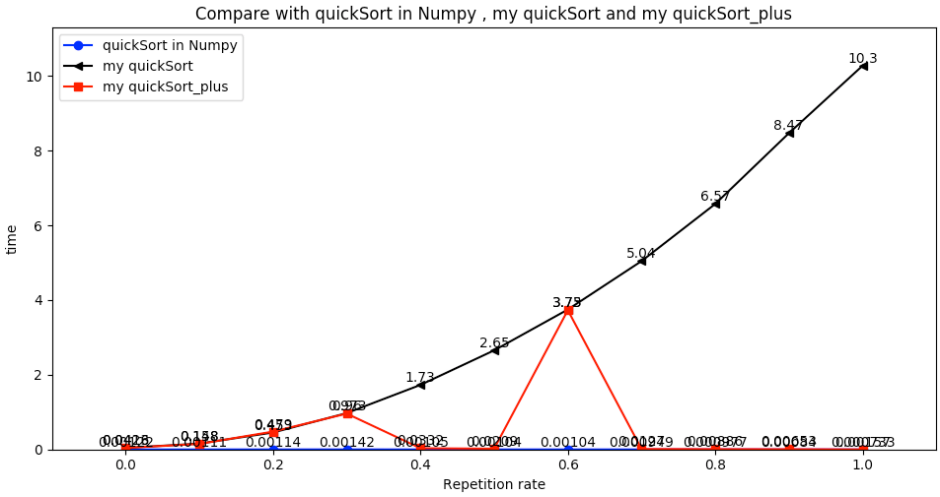
我们在各个实验数据集上运行算法观察实验现象并记录实验结果。当程序在 N=10^6，i=10 的条件下运行的时间过长，故我们展示 N=10^4 时的对比实验结果：



当 i=10 时所有元素均重复，算法的时间复杂度最差为 $O(n^2)$。因此，我们将二切分改进为三切分以解决元素重复率过高导致的算法效率过低问题，算法代码如下：

```python
def quickSort_plus(data: list, p: int, r: int) -> None:
    if p < r:
        a, b = Trisection(data, p, r)
        quickSort(data, p, a - 1)
        quickSort(data, b + 1, r)


def Trisection(data: list, p: int, r: int) -> tuple:
    """

    :param data:
    :param p:
    :param r:
    :return: the scope of the pivot
    """
    i = random.randint(p, r)
    data[i], data[p] = data[p], data[i]
    pivot = data[p]

    i, j, k = p + 1, r, p
    while i <= j:
        if data[i] < pivot:
            data[i], data[k] = data[k], data[i]
            i, k = i + 1, k + 1
        elif data[i] > pivot:
            data[j], data[i] = data[i], data[j]
            j -= 1
        else:
            i += 1

    return k, j
```

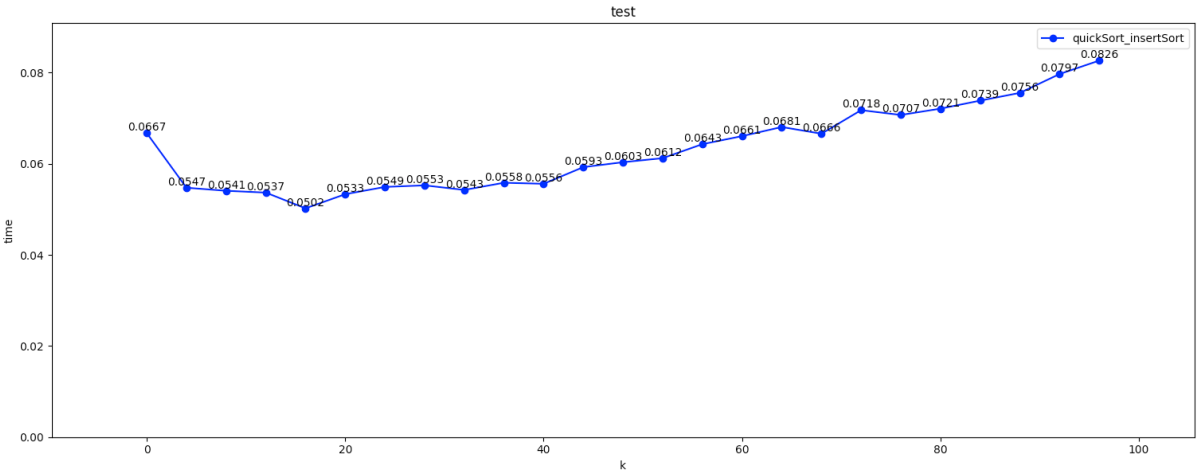
我们再进一步观察对比实验结果：



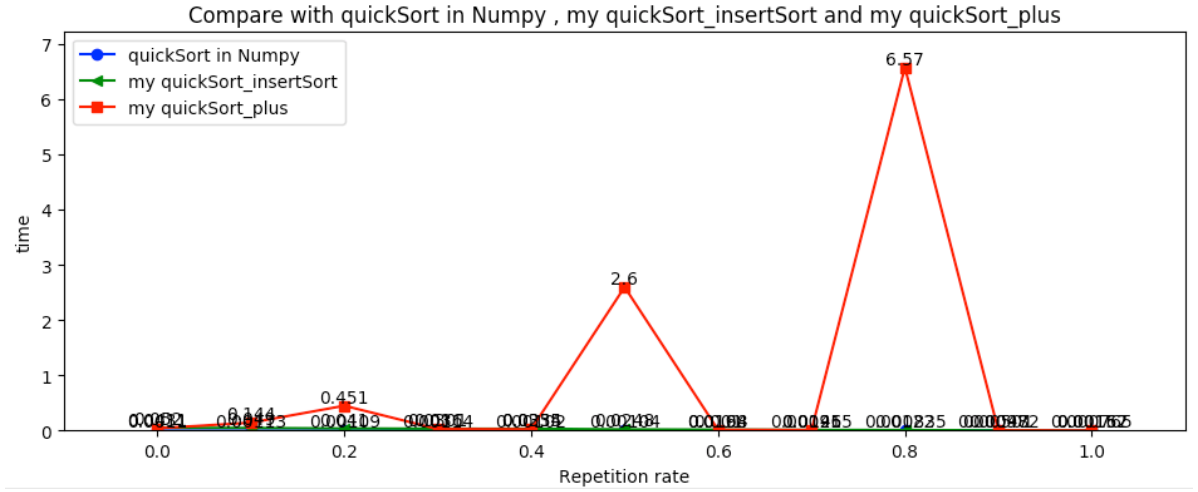
快速排序有一个缺点就是对于小规模的数据集性能不是很好。可能有人认为可以忽略这个缺点不计，因为大多数排序都只要考虑大规模的适应性就行了。但是快速排序算法使用了分治技术，最终来说大的数据集都要分为小的数据集来进行处理，所以快排分解到最后几层性能不是很好，所以我们就可以使用扬长避短的策略去优化快排：即先使用快排对数据集进行排序，此时数据集已经达到了基本有序的状态；然后当分区的规模达到一定小时，改用插入排序（因为插入排序在对基本有序的数据集排序有着接近线性的复杂度）性能比较好。算法代码如下：

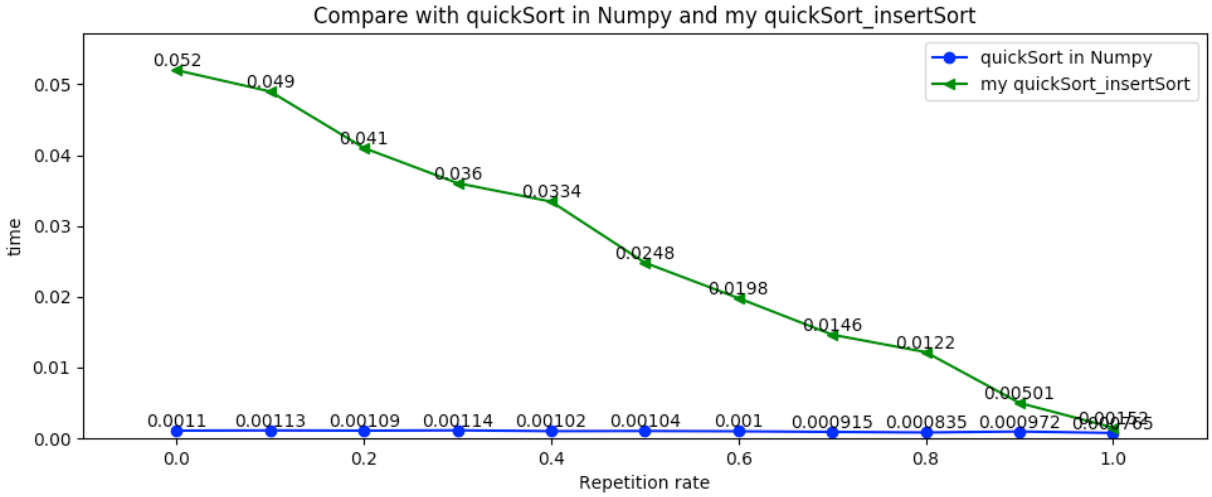
```python
def quickSort_insertSort(data: list, p: int, r: int, k: int) -> None:
    if r - p > k:
        a, b = Trisection(data, p, r)
        quickSort_insertSort(data, p, a - 1,k)
        quickSort_insertSort(data, b + 1, r,k)
    else:
        insertSort(data, p, r)


def insertSort(data: list, p: int, r: int) -> None:
    for i in range(p + 1, r + 1):
        for j in range(i, p, -1):
            if data[j] < data[j - 1]:
                data[j], data[j - 1] = data[j - 1], data[j]
            else:
                break
```

我们对这个分区规模的阈值 k 进行实验有如下对比结果：



即当 k=16 时，算法具有更好地表现，故取 k=16 得到如下对比实验结果：





显然，此时的算法执行时间优于之前的改进方案。

4.6 实验心得

通过本次实验，对快速排序算法的双切分和三切分方法有了更深入的了解。同时，了解了快速排序在大数据集上的优势，并利用其他排序补足小数据集上的劣势。