DSA第一次作业

袁逸聪 19302010020

1.1快速排序

实现算法(Array.sorts()双基准排序&自己写的单基准快速排序&快排和冒泡的缝合体)

```
/**
 * Write a program to solve the selection  problem: select the biggest k numbers
in the N numbers
 * Let k=N/2.
 * Draw a table showing the running time of your program for various values of N.
 **/

public class Selection {
    //排序从小到大排
    public static void main(String[] args) {
        int timeAll=0;
        for(int episode=0;episode<10;episode++) {


            int N = 200000;
            int K = N / 2;
            int[] intN = new int[N];
            int[] intK = new int[K];

            for (int i = 0; i < N; i++) {
                intN[i] = (int) (Math.random() * N * 20);
            }
            long startTime = System.currentTimeMillis();
            //!!生成了N个随机整数,开始操作!!

            //-------------------------Java自带排序(双基准快排)---我也不知道复杂度咋
写，应该多于单基准快排，但是多利用了cpu减少内存速度限制
//          Arrays.sort(intN);
//          for (int i = 0; i < K; i++) {
//              intK[i] = intN[i];
//          }

            //-------------------------N快排--------------------O(NlogN)
            quickSort(intN);
            for (int i = 0; i < K; i++) {
                intK[i] = intN[i];
            }

            //-------------------------k快排+一个个放进去----------O(N^2)

        //1.将前k个数存入数组k,对k快速排序
        //O(k+klogk)
```

```
//        for (int i = 0; i < K; i++) {
//            intK[i] = intN[i];
//        }
//        quickSort(intK);
//
//        //2.对剩下N-k个数，逐个比较，如果比k[0]小则替换并重新排序。这里的排序只需要处理一个未知未知的数，复杂度为n
//        //O((N-k)k)
//        for (int i = K; i < N; i++) {
//            if (intK[0] < intN[i]) {
//                intK[0] = intN[i];
//                //重新排序此数组
//                oneSort(intK);
//            }
//        }

            //3.此时K中就是前K大的数，循环输出即可
//            System.out.println(Arrays.toString(intN));
//            System.out.println(Arrays.toString(intK));

            long endTime = System.currentTimeMillis();
//            System.out.println("Time = " + (endTime - startTime));
            timeAll+=endTime;
            timeAll-=startTime;
        }
        System.out.println(timeAll/10.0);
    }

    //工具函数区

    //数组第一项大小未知，而后面所有项都排列完毕
    private static void oneSort(int[] array) {
        //System.out.println("OneSort running, new element="+array[0]);
        for (int i = 0; i < array.length - 1; i++) {
            if (array[i] > array[i + 1]) arrayExchange(array, i, i + 1);
            else return;
        }
    }

    //实现单基准快速排序(小左大右)
    //sort工具采用的是双基准快速排序
    private static void quickSort(int[] array) {
        quickSort(array, 0, array.length - 1);
    }

    private static void quickSort(int[] array, int start, int end) {
        //System.out.println("sort running,start" + start + "&end" + end);
        if (start < end) {
            int range = quickPartition(array, start, end);
            quickSort(array, start, Math.max(start, range - 1));
            quickSort(array, Math.min(end, range + 1), end);
        }
    }
```

```
    //快排中的划分
    private static int quickPartition(int[] array, int start, int end) {
        //取array[start]为快排基准
        int standard = array[end];
        //System.out.println(Arrays.toString(array));
        //System.out.println("standard" + standard);
        //想要的效果为，range左边全是小于standard，而右边全是大的。range不仅作为分界线，也保证了"引用的数是大区的"，于是发现了新的小数后可以直接交换
        int range = start;
        for (int i = start; i < end; i++) {
            if (array[i] < standard) {
                //将刚刚比较的数和range代表的大数互换
                arrayExchange(array, i, range);
                //把小的数吃进范围，保证右边是大数
                range++;
            }
        }
        //操作完毕，将基准和range互换
        arrayExchange(array, end, range);
        return range;//所有数分居两侧，并返回standard的位置，作为递归的参数
    }

    //交换数组中的两位
    private static void arrayExchange(int[] array, int a, int b) {
        //System.out.println(Arrays.toString(array) + "exchange:" + a + " " + b);
        int temp = array[a];
        array[a] = array[b];
        array[b] = temp;
        //System.out.println(Arrays.toString(array));
    }
}
```

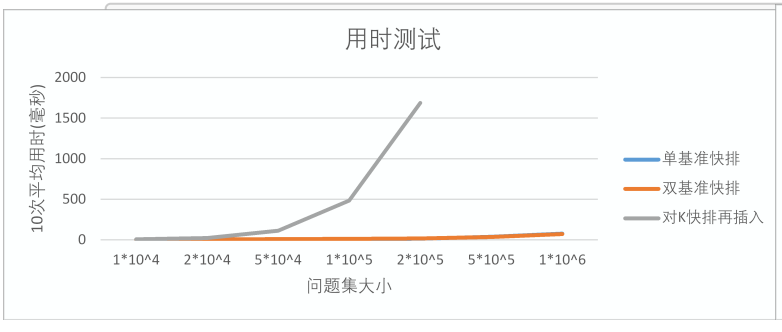
时间复杂度分析

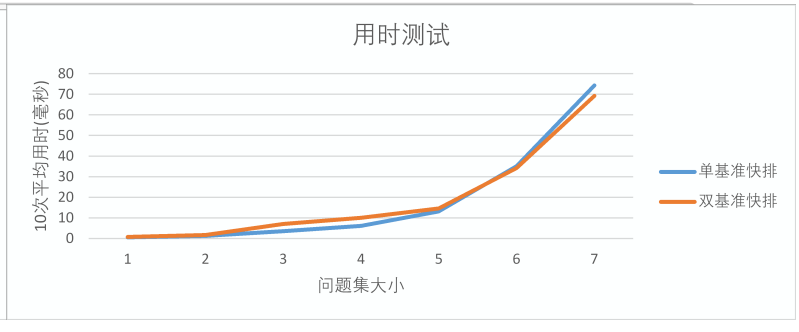
快速排序的时间复杂度应该都是O(NlogN)

双基准快速排序将数组分成3份，更多占用cpu而更少占用内存(更多地重复调用数字，而这些数字会存放在cpu的高速缓存中)。在cpu算力增长快于内存传输速度的当今，比传统快排更高效。

第三种算法本意是避免数组全排序的信息浪费，只对K个数排序，再N中剩余的数逐个比较并插入。如果用二分法，可以在O(logk)中确认插入者的位置,试图做到O(klogk+(N-k)logk)=O(Nlogk)，但由于数组的插入操作需要O(k),实际上的时间复杂度为O(k^2)即O(N^2)

耗时作图





1.6字符串排序输出

```
public class Permute {
    public static void permute(String str) {
        char[] array = str.toCharArray();
        int low = 0;
        for (int i = 0; i < array.length; i++) {
            permute(array, low, i);
        }
    }

    private static void permute(char[] str, int low, int high) {
//        System.out.println("Permute running:"+ Arrays.toString(str)
+low+high+str.length);
        //终止条件：low==str.length-1,说明所有顺序都确定，直接输出
        if (low == str.length - 1) System.out.println(String.valueOf(str));
        else {
            //将str数组中low与high位互换，递归
            char temp = str[low];
            str[low] = str[high];
            str[high] = temp;
            low++;
            for (int i = low; i < str.length; i++) {
                permute(str.clone(), low, i);//此处如果之间str会因引用变量传地址而导
致混乱，必须clone
            }
        }
    }

    public static void main(String[] args) {
        permute("abc");
    }
}
```