暴力美学之——排序

一、知识点

- 蛮力算法定义
- 选择排序
- 冒泡排序
- 算法效率评估

二、实验原理

2.1 蛮力算法

蛮力算法是一种直接解决问题的方法，通常基于问题描述或者概念定义直接设计算法。

比如，计算a^n，最直接的方法就是把数字a乘以n次，这就是直接基于指数计算概念定义来进行的。

再比如，在一个乱序列表里查找概率列表是否包含K，最直接的方法就是遍历列表的每个元素，并把每个元素和K比较，这就是基于问题描述来进行的。

蛮力算法虽然不够巧妙，但是却是一种几乎什么问题都能解决的一般性方法。

2.2 选择排序

选择排序的设计思想就是蛮力算法。

选择排序算法流程：

扫描整个待排序列表（长度为n），找到最小的元素，把这个元素和列表第一个元素交换；从第二个元素开始扫描列表，找到剩余$n-1$个元素中最小的元素，把这个元素和列表第二个元素交换；从第三个元素开始.....

这样遍历$n-1$次以后，该列表就完成了排序。

伪代码如下：

> $SelectionSort(A[0...N-1])$
> //选择排序
> //输入：一个可排序数组$A[0...n-1]$
> //输出：排序好的升序数组
> $for\ i \leftarrow 0\ to\ n-2\ do$
> $min \leftarrow i$
> $for\ j \leftarrow i+1\ to\ n-1\ do$
> $if\ A[j] < a[min]\ min \leftarrow j$
> $swap\ A[i]\ and\ A[j]$

下面的动图给出一个选择排序的实例：

15	21	1	25	12	6	8	3	5	19	10	18

COMPARES | 0 |

下面分析算法负责度：

$C(n) = \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} 1 = \frac{(n-1)n}{2} \in O(n^2)$

> 最差效率、平均效率和最优效率都是$O(n^2)$，想想，为什么？

注意，选择排序需要大概$\frac{n^2}{2}$次比较和至多$n-1$次交换；由于算法复杂度是$O(n^2)$，所以选择排序效率并不高，那么，为什么还有使用选择排序呢？因为选择排序虽然需要$O(n^2)$次读操作，但是，仅仅需要$O(n)$次写操作，当写操作很慢而读操作很快的时候，选择排序可以近似看成线性，效率还是很好的！

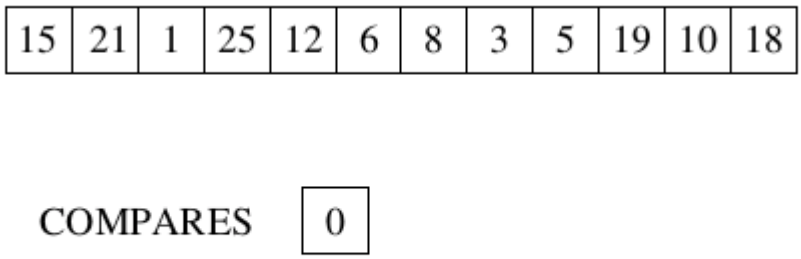
2.3 冒泡排序

冒泡排序是蛮力法在排序算法中的另一个应用。冒泡算法比较列表中的相邻元素，如果两个相邻元素顺序不对，则交换这2个元素。这样，一次遍历之后，最大的元素就到了列表的最后一个位置。这样遍历$n-1$次，列表就完成了排序。

伪代码如下：

> $BubbleSort(A[0...n-1])$
> //冒泡排序
> //输入：可排序数组$A[0...n-1]$
> //输出：排序好的非降序数组
> $for\ i \leftarrow 0\ ton-1\ do$
> $\quad forj \leftarrow 0\ ton-2-i\ do$
> $\quad\quad if\ A[j+1] < A[j]\ then\ swap\ A[j+1]\ and\ A[j]$

下面的动图展示了一个冒泡排序的实例（仅展示第一次遍历的比较过程，第一次遍历完成后，最大的元素移到了列表最后一个位置）：



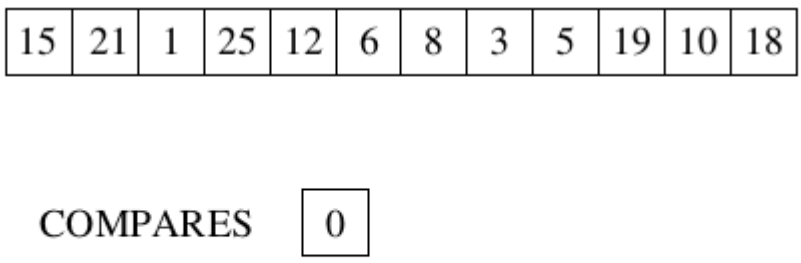
下面分析算法效率：

$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \frac{(n-1)n}{2} \in O(n^2)$

> - 最优效率：如果一开始，列表是排序好的，那么大约需要$\frac{n^2}{2}$次比较操作和0次交换操作，$O(n^2)$；
> - 最差效率：如果一开始，列表顺序是从大到小排序的，那么需要大约$\frac{n^2}{2}$次比较操作和$\frac{n^2}{2}$次交换操作，$O(n^2)$；
> - 平均效率：如果一开始，列表顺序是随机的，那么大概需要$\frac{n^2}{2}$次比较操作和少于$\frac{n^2}{2}$次交换操作，$O(n^2)$；

我们可以通过一些改进，可以提高冒泡排序在某些特定情况下算法效率，比如提前终止！提前终止策略就是，如果在某一个遍历列表时。没有交换操作发生，就说明列表已经完成排序，可以停止程序了！

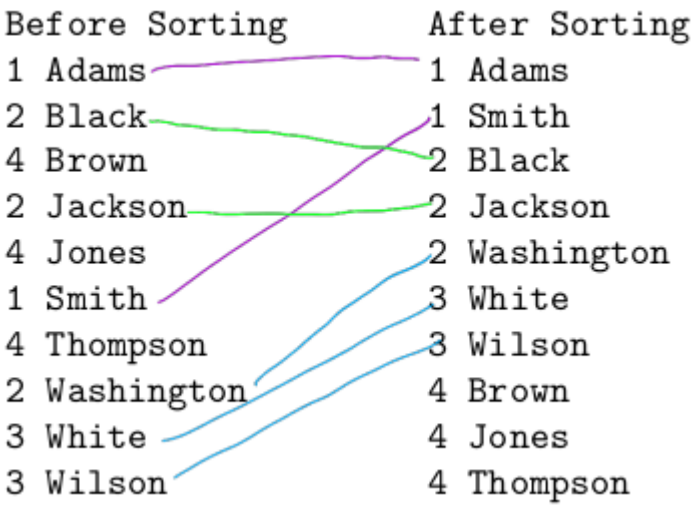
这样做以后，最优效率就变成了$O(n)$，因为只需要1次遍历，执行$n-1$次比较操作和0次交换操作。【注意：做茶效率和平均效率依旧是$O(n^2)$】。下面面的动图展示了一个，使用提前终止策略排序的例子：



2.4 什么是稳定排序？

如果排序前后，重复键值对应的元素的相对顺序不变，则称排序是稳定的。

比如，对单词首字母按照字典顺序排序，则下面的排序是稳定的(因为首字母相同的单词的相对顺序没有改变)：



我们给出的2个排序算法都是稳定的！但是稍微改一下，比如，对于选择排序而言，只要把$A[j] < A[min]$改为$A[j] \leq A[min]$，那么，选择排序算法就不稳定了！

想一想，对于上面描述的冒泡排序，应该怎么改，让排序不稳定呢？

三、实验步骤

代码已经为大家写好，在`./code`目录下。当然，如果希望自己写一下代码，也可以将提供的代码作为参考。

首先，创建一个排序算法的抽象类：

```
//SortAlgorithm.java
public abstract class SortAlgorithm
{
    public abstract void sort(int[] data);
    protected final void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

本实验使用SortDemo1作为运行时的入口，负责接受相关参数，选择合适的排序算法，对指定的数据排序。代码如下：

```
//SortDemo1.java
import java.io.*;
import java.util.*;

public class SortDemo1
{
    protected static final String progName = "SortDemo1";

```

```java
    /**可以调用的算法*/
    public enum Mode {
        BUB,
        SELECT
    }

    /**
     打印基本信息。
     */
    protected static void printUsage(String progName) {
        System.err.println("USAGE: " + progName + " [sort method] [input file]");
        System.err.println("  sort methods [bubble, quick, merge, cocktail]");
        System.err.println("EXAMPLE: " + progName + " quick random.txt");
    }
    /**
     * 主函数。接受命令行参数-包括算法名字和排序数据集
     */
    public static void main(String[] args) {
        try {
            if (args.length != 2) {
                printUsage(progName);
                System.exit(1);
            }
            // 获得排序算法名
            String algorithmUsed = args[0];
            // 获得需要排序的数据集
            String fileName = args[1];

            File inFile = new File(fileName);
            Scanner in = new Scanner(inFile);

            // 从文件读入数据
            ArrayList<Integer> buffer = new ArrayList<Integer>();
            while (in.hasNextInt()) {
                buffer.add(new Integer(in.nextInt()));
            }

            // 使用读入的数据创建列表
            int[] array = new int[buffer.size()];
            Iterator bit = buffer.iterator();
            int j = 0;
            while (bit.hasNext()) {
                array[j] = (Integer) bit.next();
                j++;
            }
            buffer = null;

            // 选择合适的算法
            SortAlgorithm sortAlgor = null;
            switch(algorithmUsed) {
                case "bubble":
                    sortAlgor = new BubbleSort();
                    break;
```

```
                case "select":
                    sortAlgor = new SelectionSort();
                    break;
                default:
                    System.err.println("Error: " + algorithmUsed + "is invalid.");
                    array = null;
                    printUsage(progName);
                    System.exit(1);
            }

            long startTime = System.nanoTime();
            // 排序
            sortAlgor.sort(array);
            long endTime = System.nanoTime();
            // 打印排序后的数组
            for (int i = 0; i < array.length; i++) {
                System.out.println(array[i]);
            }
            double timeElapsed = (endTime - startTime) / Math.pow(10, 9);
            System.out.println("Time elapsed (secs): " + timeElapsed);
        }
        catch (Exception e) {
            System.err.println(e.getMessage());
            printUsage(progName);
        }
    }
}
```

3.1 选择排序

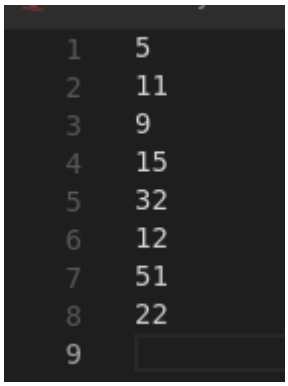
实现选择排序代码：

```
//SelectionSort.java
import java.util.ArrayList;
public class SelectionSort extends SortAlgorithm
{
    public void sort(int[] array) {
        for (int i = 0; i < array.length - 1; ++i)
        {
            int min = i;
            for (int j = i + 1; j < array.length; ++j )
            {
                if (array[j] < array[min])
                    min = j;
            }
            int temp = array[i];
            array[i] = array[min];
            array[min] = temp;
        }
    }
}
```

在源代码目录下编译java文件：

```
javac *.java
```

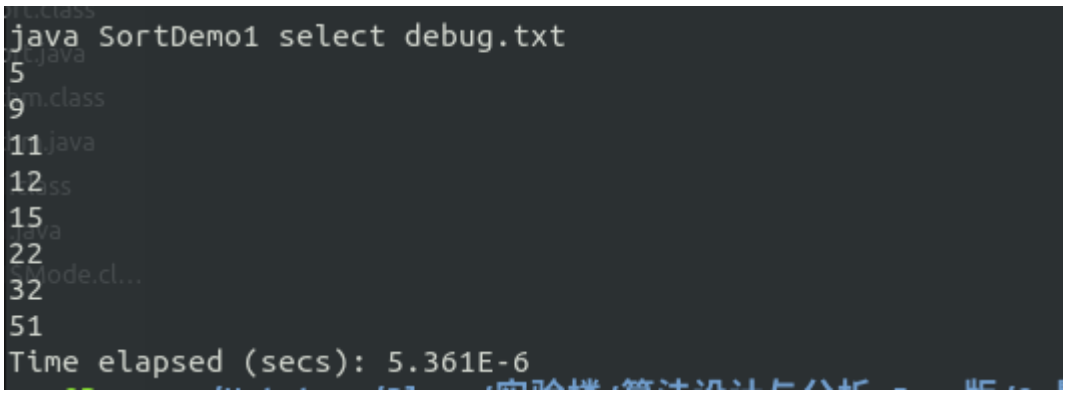
使用`debug.txt`文件验证算法正确性，此文件内容如下：



运行如下指令，检验代码正确性：

```
java SortDemo1 select debug.txt
```

得到如下结果：



3.2 冒泡排序

实现冒泡泡排序代码：

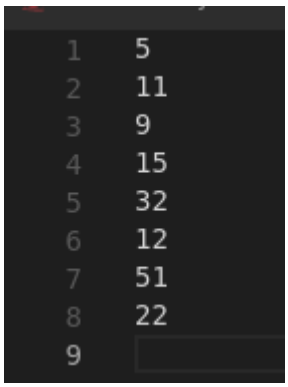
```
//BubbleSort.java
import java.util.ArrayList;

public class BubbleSort extends SortAlgorithm
{
    public void sort(int[] array) {

        for (int i = 0; i < array.length-1 ; i++) {
            for (int j = 0; j < array.length - 1 - i; j++) {
                // 交换
                if (array[j] > array[j+1]) {
                    Integer temp = array[j];
```

```
                    array[j] = array[j+1];
                    array[j+1] = temp;
                }
            }
        }
    }
}
```

在源代码目录下编译java文件：

```
javac *.java
```

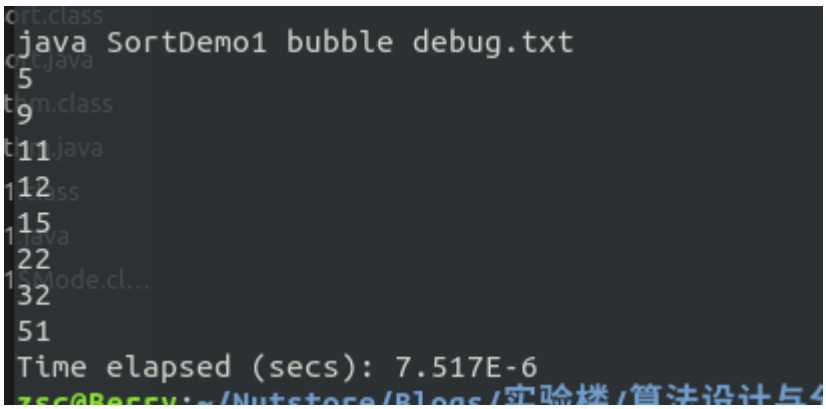
使用`debug.txt`文件验证算法正确性，此文件内容如下：



运行如下指令，检验代码正确性：

```
java SortDemo1 bubble debug.txt
```

得到如下结果：



3.3 比较算法效率

本实验提供以下几个数据文件（每个文件都包含数万个数字）：

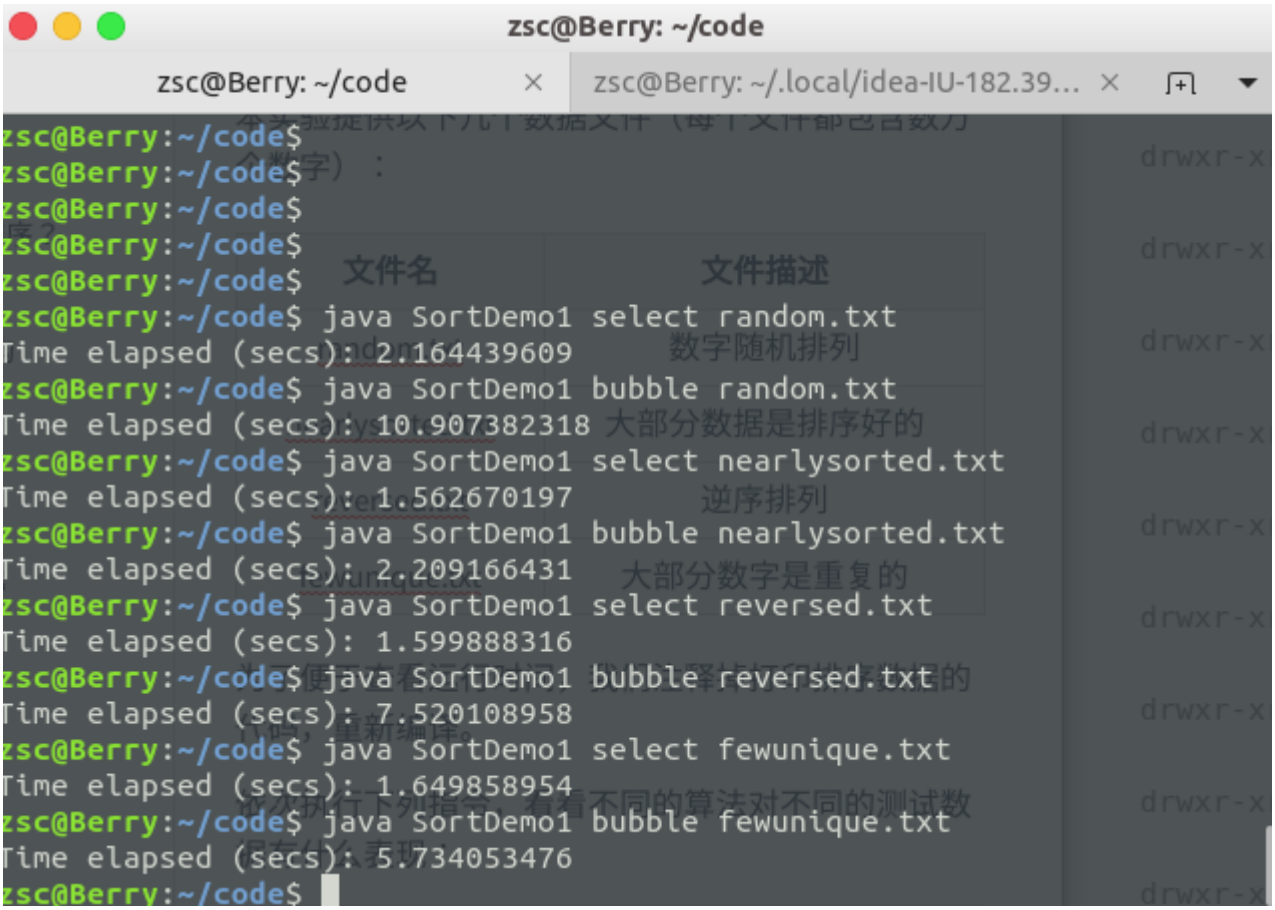
文件名	文件描述
random.txt	数字随机排列
nearlysorted.txt	大部分数据是排序好的
reversed.txt	逆序排列
fewunique.txt	大部分数字是重复的

为了便于查看运行时间，我们注释掉打印排序数据的代码，重新编译。

依次执行下列指令，看看不同的算法对不同的测试数据有什么表现：

```
jacac *.java
java SortDemo1 select random.txt
java SortDemo1 bubble random.
java SortDemo1 select nearlysorted.txt
java SortDemo1 bubble nearlysorted.txt
java SortDemo1 select reversed.txt
java SortDemo1 bubble reversed.txt
java SortDemo1 select fewunique.txt
java SortDemo1 bubble fewunique.txt
```

得到如下结果：



对每个数据文件，比较两个算法的运行时间，你发现了什么呢？是不是发现很符合我们之前分析得到的规律呢？

四、总结

本实验介绍蛮力算法（Brute Fore）在排序中的应用。蛮力算法是一种是哪个简单直接地解决问题的方法，常常直接基于问题的描述来设计算法流程。在排序算法中，典型的蛮力算法有选择排序和冒泡排序，完成本实验后，学生应该掌握了如下内容：

- 介绍选择排序和冒泡排序原理；
- 实现选择排序和冒泡排序程序；
- 评估算法效率；

希望读者在实验过程中有自己的思考，学会根据实际情况选择合适的排序算法。

五、课后习题

使用实验中提供给你的数据文件（或者自己生成的模拟测试数据），完成以下任务：

1. 模拟写操作很费时的情况，看看冒泡排序和比较排序的效率差多少？现在你理解为什么说“选择排序在写操作耗时的情况下可以近似看做线性复杂度”这句话了吗？【提示：可以考虑在每个交换操作后面添加一个0.001秒的延时模拟“写操作耗时”的情况】
2. 对同样一组数据，比较冒泡排序对乱序输入和排好序的输入的效率的异同？【要求使用提前终止策略】