深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验项目名称：排序算法性能分析

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：杜智华

报告人：欧阳宇杰 学号：2021150143 班级：计科 2 班

实验时间：2023 年 2 月 21 日~2023 年 3 月 7 日

实验报告提交时间：2023 年 3 月 7 日星期二

教务处制

一、 实验目的

1. 掌握选择排序、冒泡排序、合并排序、快速排序、插入排序算法原理
2. 掌握不同排序算法时间效率的经验分析方法，验证理论分析与经验分析的一致性。

二、 实验概述

排序问题要求我们按照升序排列给定列表中的数据项，目前为止，已有多种排序算法提出。本实验要求掌握选择排序、冒泡排序、合并排序、快速排序、插入排序算法原理，并进行代码实现。通过对大量样本的测试结果，统计不同排序算法的时间效率与输入规模的关系,通过经验分析方法,展示不同排序算法的时间复杂度，并与理论分析的基本运算次数做比较，验证理论分析结论的正确性。

三、 实验要求

1、实现选择排序、冒泡排序、合并排序、快速排序、插入排序算法；

2、以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计不同排序算法在 20 个样本上的**平均运行时间**；

3、分别以 n=10000，n=20000，n=30000，n=40000，n=50000 等等，重复 2 的实验，画出不同排序算法在 20 个随机样本的平均运行时间与输入规模 n 的关系，如下图 1 所示；

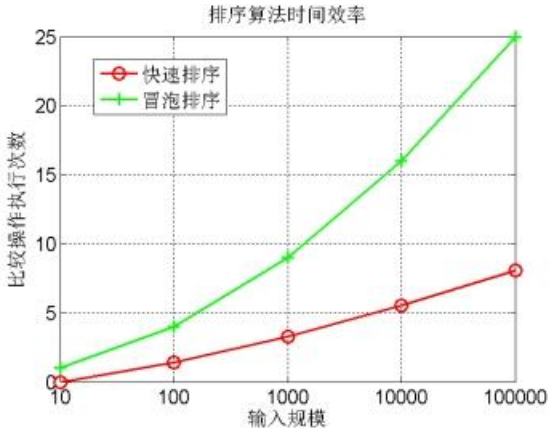


图 1. 时间效率与输入规模 n 的关系图

4、画出理论效率分析的曲线和实测的效率曲线，注意：由于实测效率是运行时间，而理论效率是基本操作的执行次数，两者需要进行对应关系调整。调整思路：以输入规模为 10000 的数据运行时间为基准点,计算输入规模为其他值的理论运行时间，画出不同规模数据的理论运行时间曲线，并与实测的效率曲线进行比较。经验分析与理论分析是否一致？如果不一致，请解释存在的原因。

5、思考题：现在有 1 亿的数据，请选择合适的排序算法与数据结构，在有限的时间内完成进行排序。

四、 实验内容步骤与结果

以下将按照选择排序、冒泡排序、插入排序、合并排序、快速排序的顺序，分别给出每一个排序算法的设计原理或思路和伪代码，然后完成老师在实验要求中要求要做的内容。最后探究实验要求中的思考题。

实现各个排序算法

1. 选择排序

1.1 设计原理或思路

首先在未排序的序列中找到最小的元素，将它和序列的第一个元素交换位置。然后，在剩下的序列中找到最小的元素，将它与序列的第二个元素交换位置。再在剩下的序列中找到最小的元素，将它与序列的第三个元素交换位置。以此类推往复，直到整个序列都有序。

```
选择排序伪代码:
SelectSort(array)
    for i=0 to array.length-2
        key=i
        for j=i+1 to array.length-1
            if array[j] < array[key]
                key=j;
        if key!=i
        swap(array[key],array[i])
```

如下图 1 所示，每次循环都会将剩余序列中的最小值移至已经有序序列的末尾，直至全部有序，排序完成。

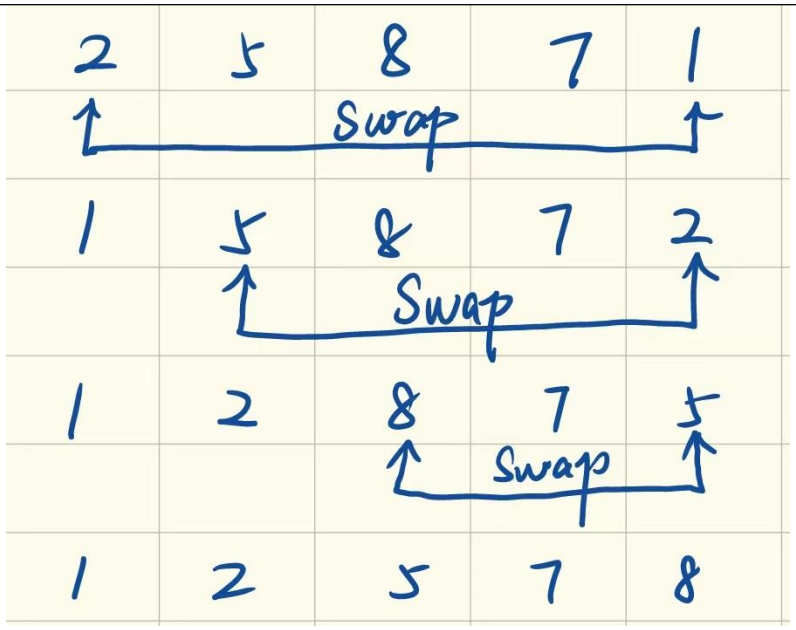


图 1　　选择排序示意图

1.2 算法实际执行时间

选取数据规模为：1、2、3、4、5、10、30、50，单位为万的数据规模进行排序算法运行时间测试。对于每一个数据规模，都随机产生 20 组测试样本来排序，取 20 次的运行时间平均值来作为每一个数据规模下,算法实际执行所需的时间值。以上过程分析在下文不再赘述。选择排序算法各数据规模下运行平均时间如下表所示，数据规模单位为个，时间单位为 ms。

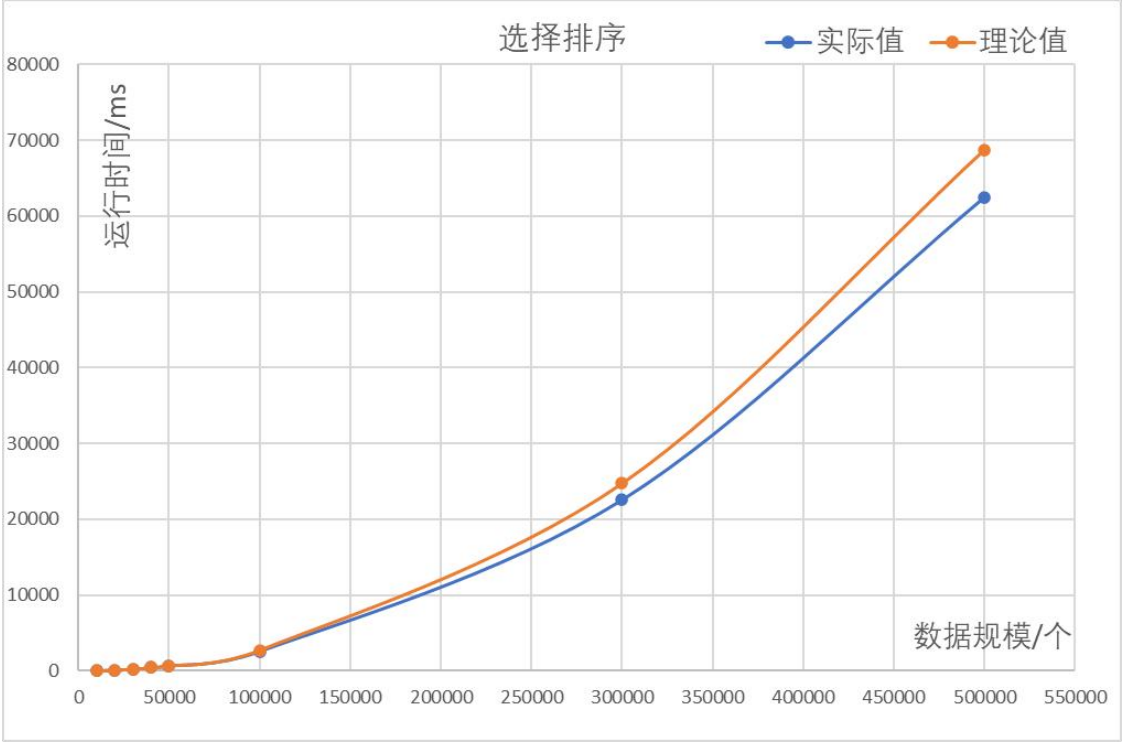
数据规模	10000	20000	30000	40000	50000	100000	300000	500000
运行时间	27.5	109.65	249.2	446.2	707.05	2600.2	22565.33	62458.5

1.3 对于算法效率的分析

已知选择排序算法平均时间复杂度为$O(n^2)$，经过分析和在实验要求里的提示，我以输入规模为 10000 的数据运行时间为基准点，通过 $\frac{n_1^2}{n_2^2} = \frac{time1}{time2}$ 的比例关系，计算得到各实验数据中算法执行时间的理论值。通过与实际值的对比得到如下表，其中误差为（实际值-理论值）/实际值。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
实际时间	27.5	109.65	249.2	446.2	707.05	2600.2	22565.33	62458.5
理论时间	27.5	110	247.5	440	687.5	2750	24750	68750
误差	0%	-0.32%	0.68%	1.39%	2.77%	-5.76%	-9.68%	-10.07%

下面为选择排序的理论效率曲线和实测效率曲线对比图：



1.4 对于上述对比结果的解释与分析

通过上面的曲线图可明显看出理论值和实验值在数据规模较小时差距很小，几乎相同，但是在数据规模增大时，实际值与理论值相对差距是不断增大的，且为实际值小于理论值。

实验数据和所绘制的图像显示理论值大多大于实验值，据我对实验数据的分析，我推测最小数据规模的运行时间偏大，而我们又将最小数据规模的运行时间作为基准点来进行理论值的计算，就会致理论值不准且偏大。这在数据规模大时更为明显，因为在数据规模较小时，计算误差时作为分母的实际运行时间较小，致使小的波动也可以造成大的误差比例影响，致使在数据规模较小时随机性较大。

2. 冒泡排序

2.1 设计原理或思路

比较相邻的元素。如果第一个元素比第二个元素大，则交换它们两个。接着比较第二个和第三个元素，如果第二个比第三个元素大，则交换它们两个。以此类推，对每一对相邻元素都做相同的动作，从开头的第一对到结尾的最后一对。完成一趟比较交换后，最后的元素就会是最大的元素。然后我们对前面剩余的元素序列做同样的工作，完成一趟后倒数第二个元素就会是次大的元素。如此重复下去，直到序列有序，排序完成。

```
1  冒泡排序伪代码:
2  BubbleSort(array)
3      for i=0 to array.length-2
4          for j=0 to array.length-2-i
5              if array[j] > array[j+1]
6                  swap(array[j],array[j+1])
```

如下图 2 所示，每次循环都会将当前在排序的序列的最大值移至序列末尾，循环多次，直至排序完成

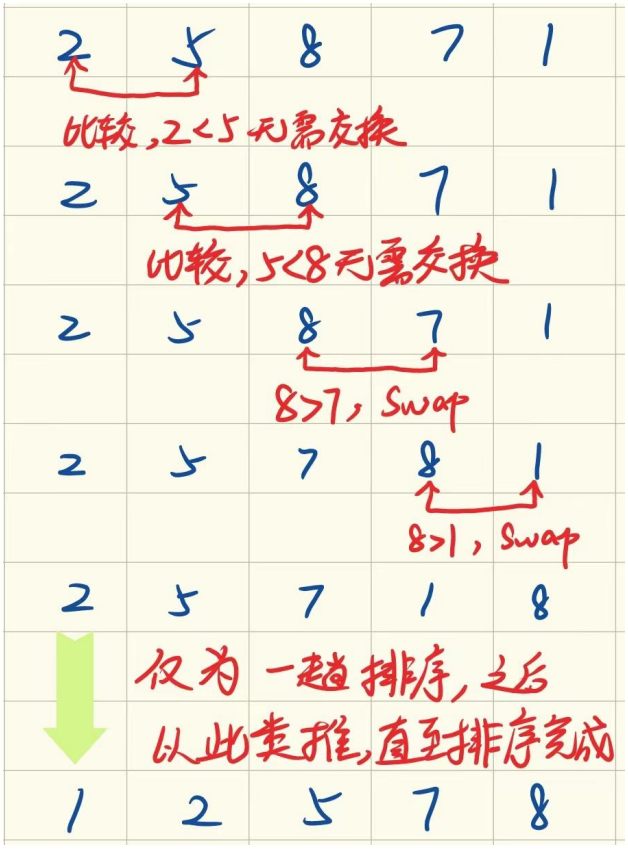


图 2　　冒泡排序示意图

2.2 算法实际执行时间

冒泡排序算法在各数据规模下平均运行时间如下表。

数据规模单位为个，平均运行时间单位为 ms。

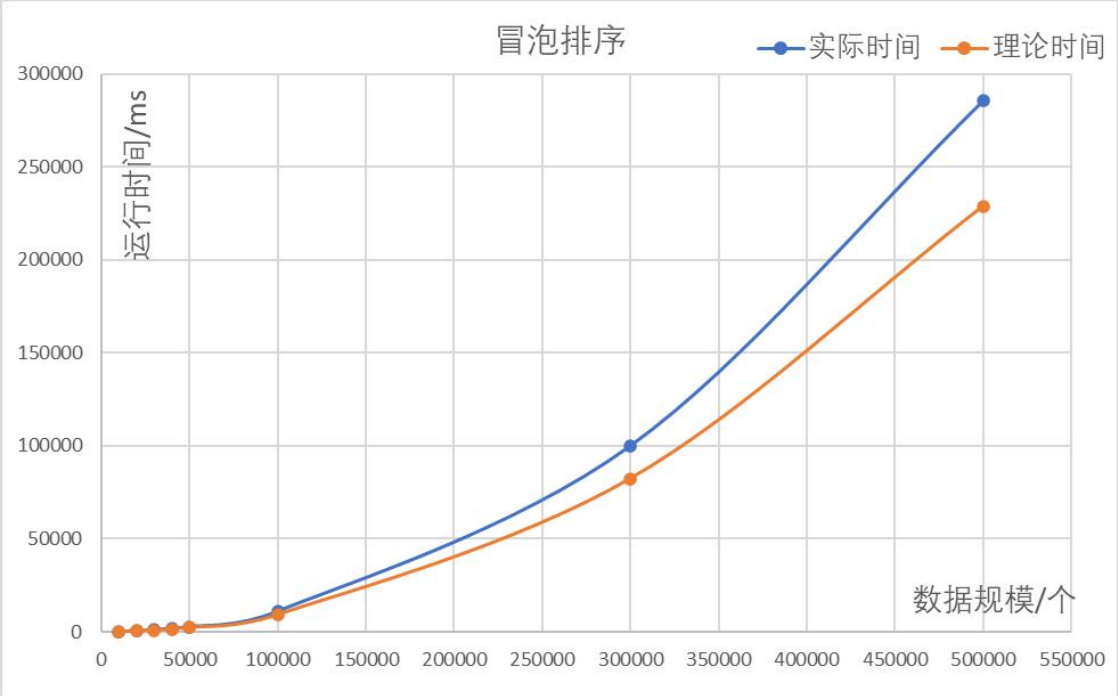
数据规模	10000	20000	30000	40000	50000	100000	300000	500000
运行时间	91.45	426.35	999	1780.5	2665.95	10927.6	99993.33	285581.5

2.3 对于算法效率的分析

已知冒泡排序算法平均时间复杂度为$O(n^2)$，经过分析和在实验要求里的提示，我以输入规模为 10000 的数据运行时间为基准点，通过 $\frac{n_1^2}{n_2^2} = \frac{time1}{time2}$ 的比例关系，计算得到各实验数据中算法执行时间的理论值。通过与实际值的对比得到如下表，其中误差为（实际值-理论值）/实际值。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
实际时间	91.45	426.35	999	1780.5	2665.95	10927.6	99993.33	285581.5
理论时间	91.45	365.8	823.05	1463.2	2286.25	9145	82305	228625
误差	0%	14.20%	17.60%	17.82%	14.24%	16.31%	17.69%	19.94%

下面为冒泡排序的理论效率曲线和实测效率曲线对比图：



2.4 对于上述对比结果的解释与分析

根据上面的曲线图可明显看出冒泡排序的实验值大于根据基准点算出来的理论值。我采用的是原始的冒泡排序算法，未经过优化，导致实验耗时很大。

实验数据和所绘制的图像显示理论值小于实验值，据我对实验数据的分析，我推测原因有：1. 实现的冒泡排序算法为未优化的算法，算法未优化的劣势在小数据规模时没怎么体现，但随着数据规模的增大，算法未优化的劣势所带来的时间差异会越来越明显，而我们又以最小数据规模的运行时间作为基准点计算理论值，这应该会导致理论值普遍偏小。2. 最小数据规模的运行时间偏小，而我们又将最小数据规模的运行时间作为基准点来进行理论值的计算，就会致理论值都不准且偏小。

3. 插入排序

3.1 设计原理或思路

将整个序列分为有序和无序的两个部分。前者在左边，后者在右边。开始有序的部分只有第一个元素 其余都属于无序的部分。每次取出无序部分的第一个（最左边）元素，把它加入到有序部分。假设合适的插入位置为 p，则原 p 位置及其后面的有序部分元素都向右移动一个位置，有序的部分即增加了一个元素。一直做下去，直到无序的部分没有元素，则序列整体有序，排序完成。

```
插入排序伪代码:
InserSort(A)
    for j=1 to A.length-1:
        key=A[j]
        //将A[j]插入已排序序列中
        i=j-1
        while i>=0 and A[i]>key
            A[i+1]= A[i]
            i=i-1
        A[i+1]=key
```

#如下图 5 所示，依次将右边的待排序序列中的首元素插入到左边的有序序列中，直至待排序序列为空，则排序完成。

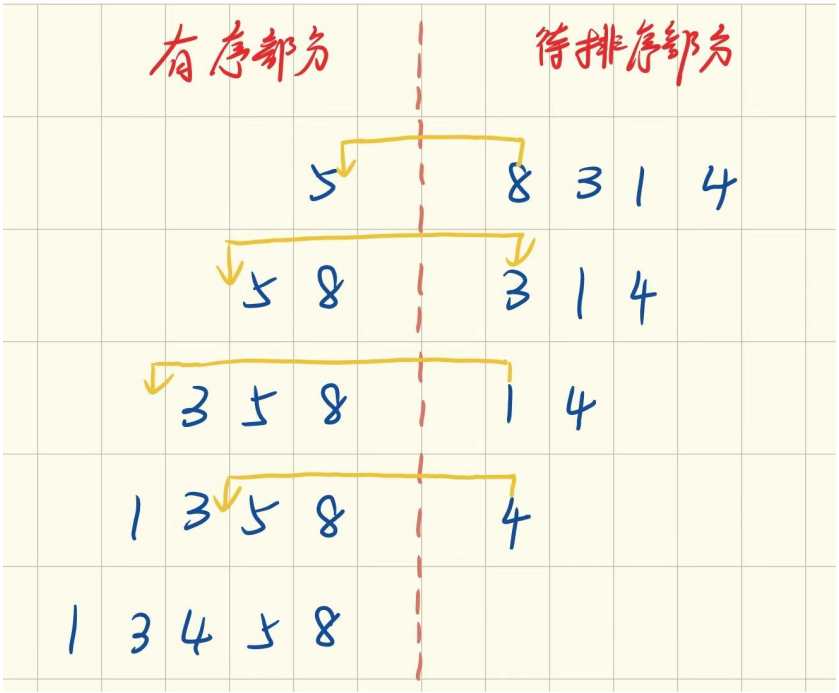


图 5　　插入排序示意图

3.2 算法实际执行时间

插入排序算法在各数据规模下平均运行时间如下表。

数据规模单位为个，平均运行时间单位为 ms。

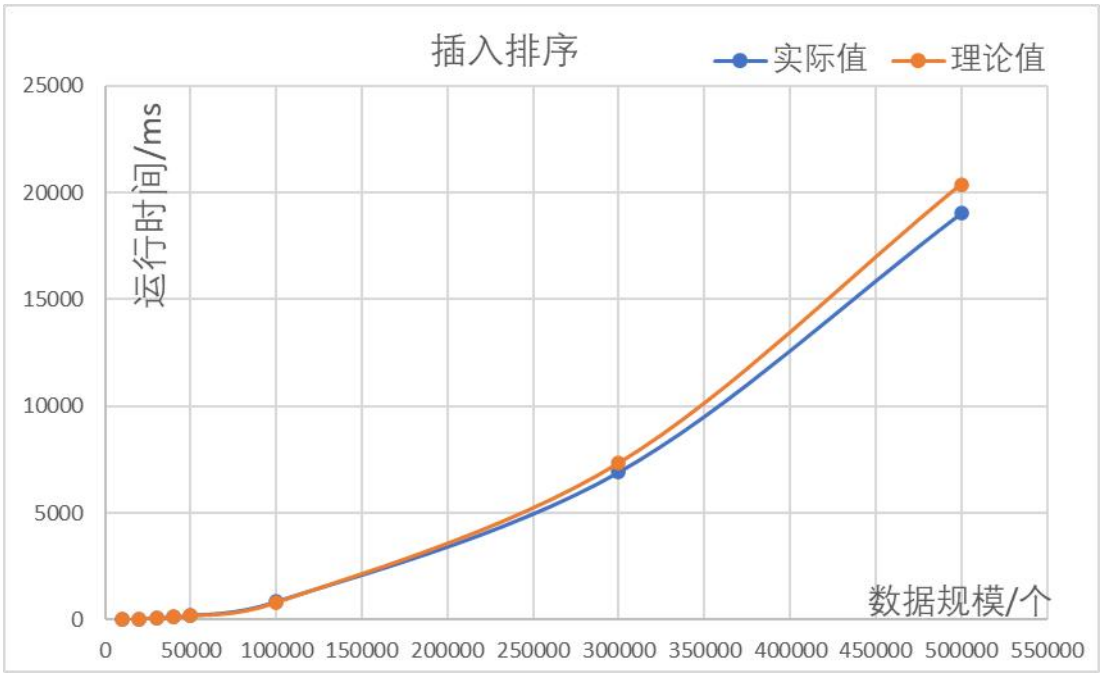
数据规模	10000	20000	30000	40000	50000	100000	300000	500000
运行时间	8.15	32.5	73.3	133.2	209.25	850.4	6910	19031.5

3.3 对于算法效率的分析

已知插入排序算法平均时间复杂度为$O(n^2)$，经过分析和在实验要求里的提示，我以输入规模为 10000 的数据运行时间为基准点，通过 $\frac{n_1^2}{n_2^2} = \frac{time1}{time2}$ 的比例关系，计算得到各实验数据中算法执行时间的理论值。通过与实际值的对比得到如下表，其中误差为（实际值-理论值）/实际值。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
实际时间	8.15	32.5	73.3	133.2	209.25	850.4	6910	19031.5
理论时间	8.15	32.6	73.35	130.4	203.75	815	7335	20375
误差	0%	-0.30%	-0.07%	2.10%	2.63%	4.16%	-6.15%	-7.00%

下面为插入排序的理论效率曲线和实测效率曲线对比图：



3.4 对于上述对比结果的解释与分析

通过上面的曲线图可明显看出理论值和实验值在数据规模较小时差距很小，几乎相同，但是在数据规模增大时，实际值与理论值相对差距是不断增大的，且为实际值小于理论值。

探讨算法本质我们可以知道，插入排序时间复杂度的最优情况是$O(n)$，而平均和最坏情况下才是$O(n^2)$。所以当选取平均时间复杂度为$O(n^2)$时，我们通过 $\frac{n_1^2}{n_2^2} = \frac{time1}{time2}$ 的比例关系来求运行时间理论值会出现数据规模越大，实验值与理论值相差越大的情况，并且应当是理论值大于实际值。

4. 合并排序

4.1 设计原理或思路

合并排序通过递归的方式将大的序列不断分割，直至分至序列的大小为 1，因为此时有且仅有一个元素，自然该序列有序，然后将两两相邻的大小为 1 的序列合并成大小为 2 的有序序列，再将两两相邻的大小为 2 的有序序列合并成大小为 4 的有序序列，以此类推，直至整个序列有序，排序完成。

```
合并排序伪代码
//Merge函数用于合并两个有序序列
Merge(sourceArray,tempArray,start,mid,end)
    i = start  j = mid+1  k = start
    //两两比较将较小的并入新的有序数列中
    while i <= mid and j <= end
        if sourceArray[i] < sourceArray[j]
            tempArray[k] = sourceArray[i++]
        else
            tempArray[k] = sourceArray[j++]
        k++
//将mid前剩余的并入
    while i <= mid
        tempArray[k++] = sourceArray[i++]
//将mid后剩余的并入
    while j <= end
        tempArray[k++] = sourceArray[j++]
//将已经有序的序列拷贝给原序列
    for x = start to end
        sourceArray[x] = tempArray[x]

MergeSort(sourceArray,tempArray,start,end)
    if start < end
       mid = start + (end - start)/2  //将原序列平分
       MergeSort(sourceArray,tempArray,start,mid) //使序列前半有序
       MergeSort(sourceArray,tempArray,mid+1,end) //使序列后半有序
       Merge(sourceArray,tempArray,start,mid,end) //使序列整体有序
```

#如下图 3 所示，不断细分直至序列长度为 1，然后再逐步合并同时排序，直至合并后的序列长度与原序列长度一样，此时排序也完成了。

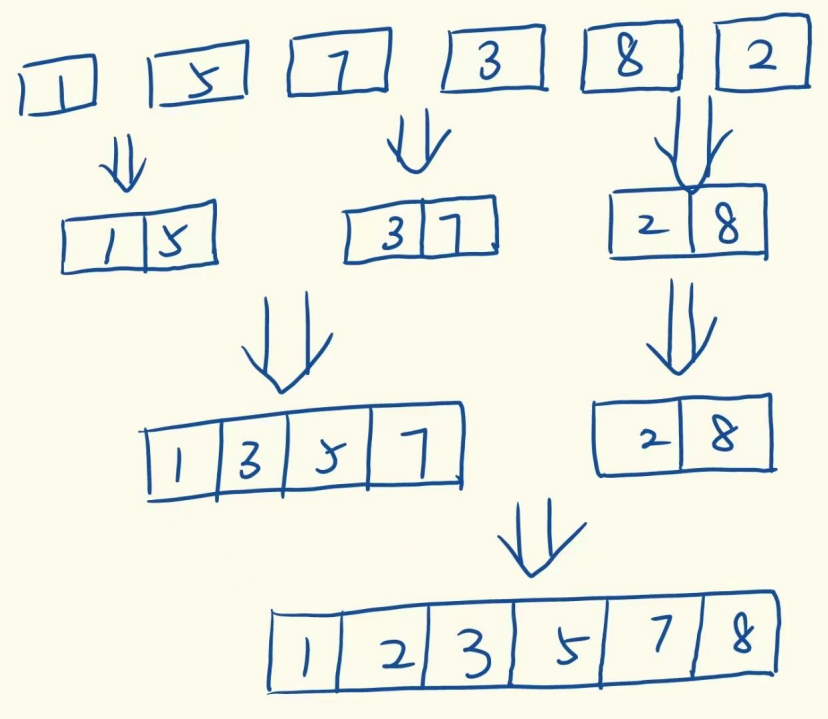


图 3　　合并排序示意图

4.2 算法实际执行时间

合并排序算法在各数据规模下平均运行时间如下表。

数据规模单位为个，平均运行时间单位为 ms。

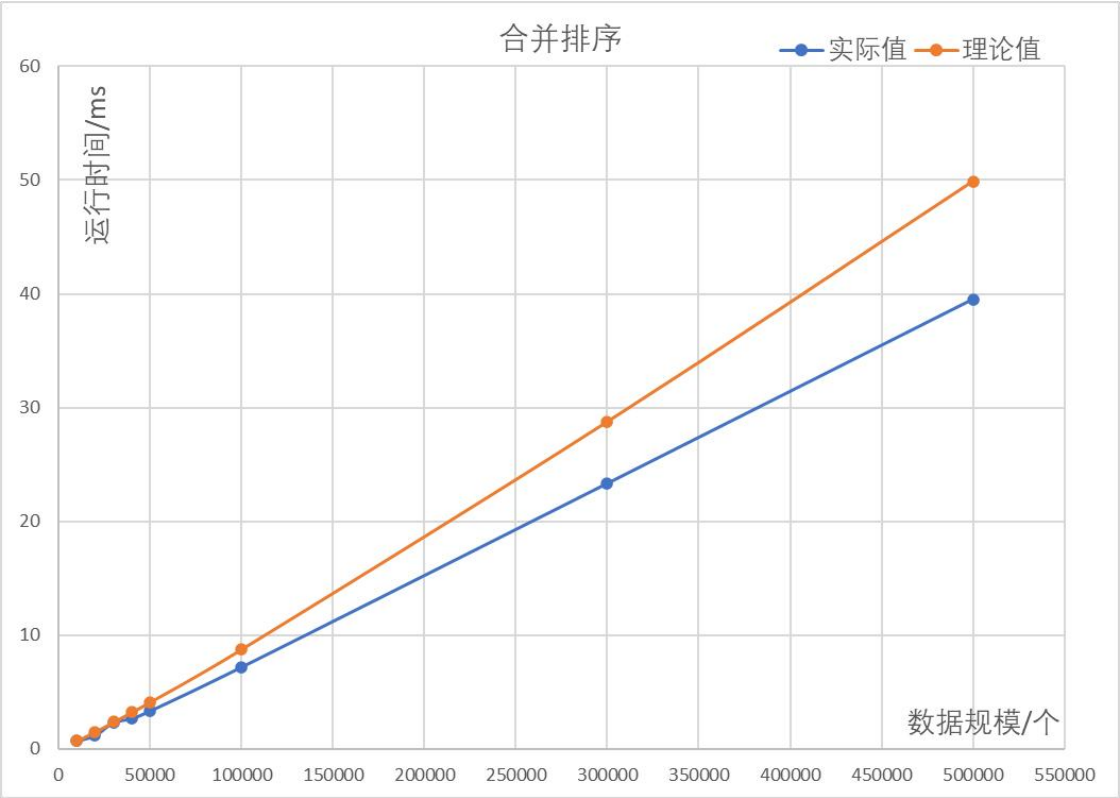
数据规模	10000	20000	30000	40000	50000	100000	300000	500000
运行时间	0.7	1.2	2.33	2.67	3.35	7.2	23.33	39.5

4.3 对于算法效率的分析

已知合并排序算法平均时间复杂度为$O(n\log n)$，经过分析和在实验要求里的提示，我以输入规模为 10000 的数据运行时间为基准点，通过 $\frac{n_1\log n_1}{n_2\log n_2}=\frac{time1}{time2}$ 的比例关系，计算得到各实验数据中算法执行时间的理论值。通过与实际值的对比得到如下表，其中误差为（实际值-理论值）/实际值。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
实际时间	0.7	1.2	2.33	2.67	3.35	7.2	23.33	39.5
理论时间	0.7	1.5	2.35	3.22	4.11	8.75	28.75	49.86
误差	0%	-25%	-0.86%	-20.60%	-22.69%	-21.53%	-23.23%	-26.23%

下面为合并排序的理论效率曲线和实测效率曲线对比图：



4.4 对于上述对比结果的解释与分析

根据上面的曲线图可看出对比其他 3 个平均时间复杂度为$O(n^2)$的算法，合并排序已经基本呈现出线性递增情形，那么我们显然可以知道合并排序的效率要远远优于其他 3 个平均时间复杂度为$O(n^2)$的算法。

实验数据和所绘制的图像显示理论值大于实验值，据我对实验数据的分析，我推测是最小数据规模的运行时间偏大，而我们又将最小数据规模的运行时间作为基准点来进行理论值的计算，就会致理论值都不准且偏大。但在数据规模较小时，由于运行时长很小，致分母很小，致使微小的波动也会造成较大的误差比例，误差的随机性大，这属正常情况。

5. 快速排序

5.1 设计原理或思路

我们将待排序序列中第一个元素设置为枢轴元素，然后将序列中所有小于枢轴元素的元素放在它左边，所有大于等于枢轴元素的元素放在它右边。那么我们容易得到此时枢轴元素所在的位置就是正确的位置。然后我们对枢轴元素左边的子序列和枢轴元素右边的子序列都进行相同的操作。依此类推，利用递归的思想，我们可以让所有元素到它们的正确位置上，实现序列有序。

```
快速排序伪代码:
//对arr[low...high]进行快速排序
QuickSort( arr, low, high)
    if low < high // 长度大于1
        pivotloc = Partition(arr, low, high); // 划分子表操作，定位枢轴记录
        QuickSort( arr, low, pivotloc-1); // 对低子表递归排序
        QuickSort( arr, pivotloc+1, high); // 对高子表递归排序

Partition(arr, low, high)
    pivotkey = arr[low]; // 用序列的第一个元素作枢轴元素
    while low < high // 从表的两端交替地向中间扫描
    // 将比枢轴元素小的元素移到低端
        while low<high and arr[high]>=pivotkey --high;
        arr[low] = arr[high];
    // 将比枢轴元素大的元素移到高端
        while low<high and arr[low]<=pivotkey ++low;
        arr[high] = arr[low];
    arr[low] = pivotkey; // 枢轴元素到位
    return low; // 返回枢轴位置
```

#下图 4 展示了快速排序的一趟排序过程，每经过一趟排序过程，就有一个枢轴元素到达正确的位置，要想完成整个排序过程，需要不断对左序列和右序列都进行递归调用，不断地进行下图 4 中所展示的操作。

图 4　　快速排序示意图

5.2 算法实际执行时间

快速排序算法在各数据规模下平均运行时间如下表。

数据规模单位为个，平均运行时间单位为 ms。

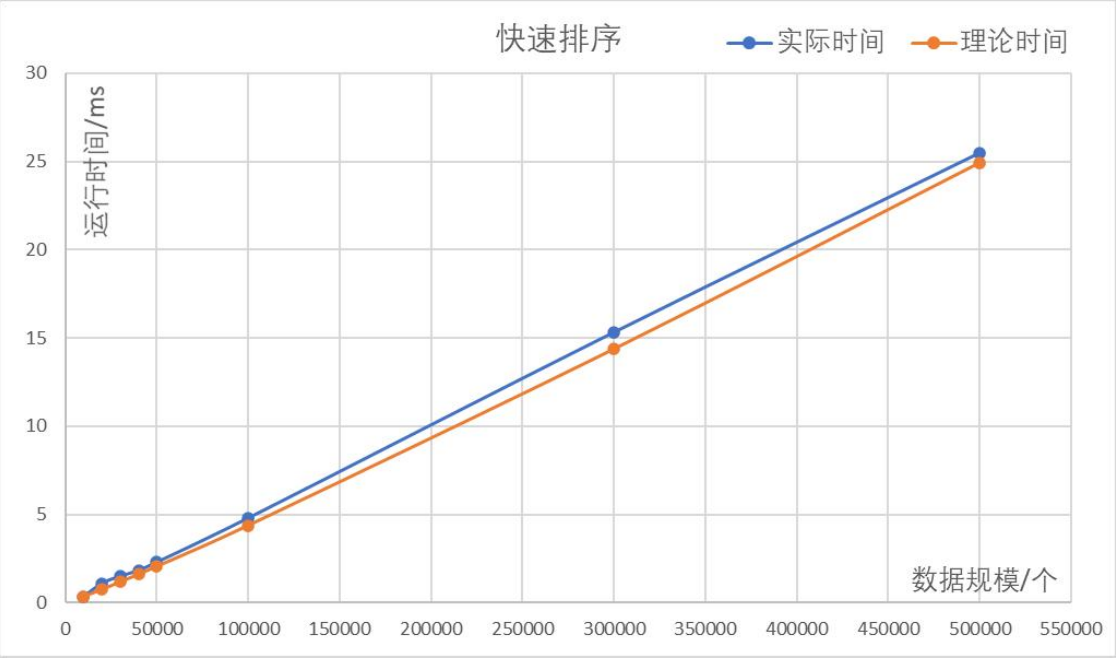
数据规模	10000	20000	30000	40000	50000	100000	300000	500000
运行时间	0.35	1.1	1.5	1.83	2.3	4.8	15.33	25.5

5.3 对于算法效率的分析

已知快速排序算法平均时间复杂度为$O(n\log n)$，经过分析和在实验要求里的提示，我以输入规模为 10000 的数据运行时间为基准点，通过 $\frac{n_1\log n_1}{n_2\log n_2}=\frac{time1}{time2}$ 的比例关系，计算得到各实验数据中算法执行时间的理论值。通过与实际值的对比得到如下表，其中误差为（实际值-理论值）/实际值。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
实际时间	0.35	1.1	1.5	1.83	2.3	4.8	15.33	25.5
理论时间	0.35	0.75	1.18	1.61	2.06	4.38	14.38	24.93
误差	0%	31.82%	21.33%	12.02%	10.40%	8.75%	6.20%	2.24%

下面为快速排序的理论效率曲线和实测效率曲线对比图：



5.4 对于上述对比结果的解释与分析

根据上面的曲线图可看出与其他 3 个平均时间复杂度为$O(n^2)$的算法相比，合并排序基本呈现出线性递增情形，则显然我们可以知道快速排序的效率要远远优于其他 3 个平均时间复杂度为$O(n^2)$的算法，且比复杂度相同的合并排序也要稍快。

实验数据显示理论值小于实验值，原因我推测有：1. 快速排序的最坏时间复杂度为$O(n^2)$，最好和平均时间复杂度才是$O(n\log n)$，故易知选取平均时间复杂度

$O(n\log n)$的情况下，我们通过 $\frac{n_1\log n_1}{n_2\log n_2} = \frac{time1}{time2}$ 的比例关系来计算理论值，势必将导致计算得到的理论值偏小。2. 快排实现用的是递归写法，递归造成额外的空间和时间开销。

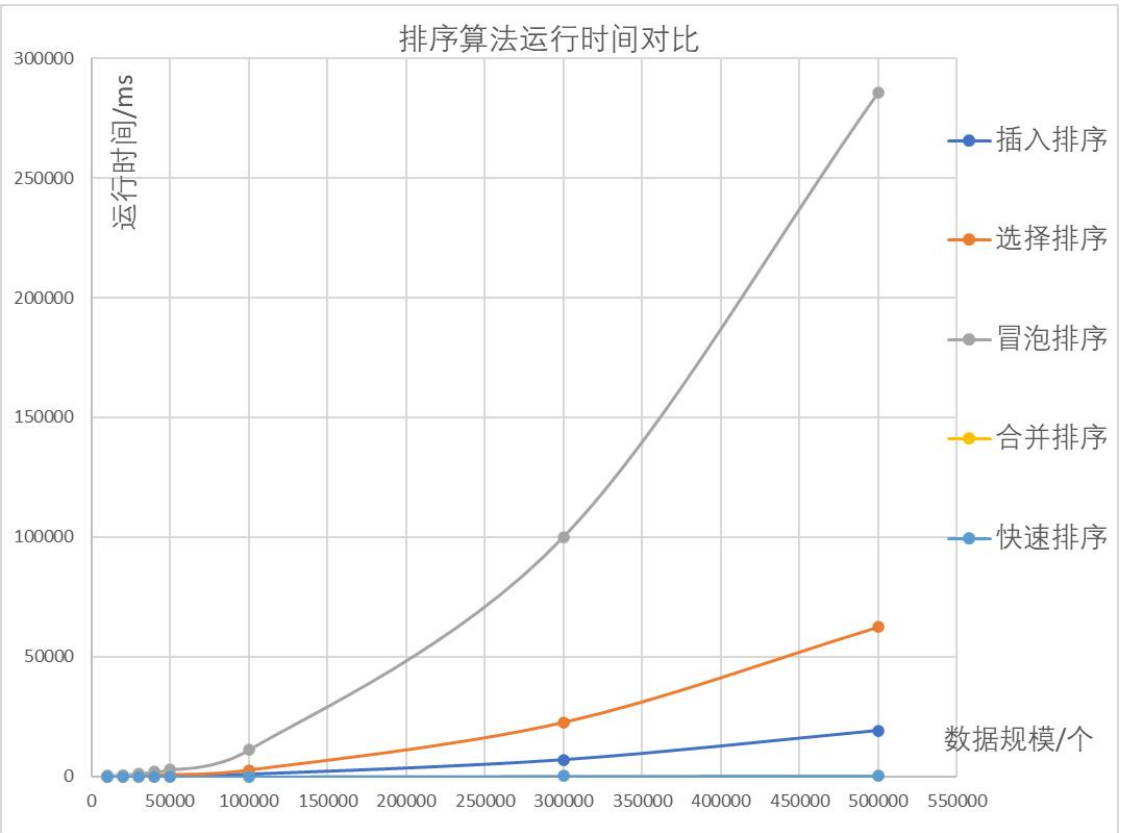
排序算法运行时间对比及分析

1. 排序算法运行时间对比

五个排序算法在不同数据规模下的实际运行时间如下，绘得图像如下所示，以下时间均为毫秒(ms)，并且均为在对应数据规模下在 20 个随机样本的平均运行时间。

数据规模	10000	20000	30000	40000	50000	100000	300000	500000
InsertSort	8.15	32.5	73.3	133.2	209.25	850.4	6910	19031.5
SelectSort	27.5	109.65	249.2	446.2	707.05	2600.2	22565.33	62458.5
BubbleSort	91.45	426.35	999	1780.5	2665.95	10927.6	99993.33	285581.5
MergeSort	0.7	1.2	2.33	2.67	3.35	7.2	23.33	39.5
QuickSort	0.35	1.1	1.5	1.83	2.3	4.8	15.33	25.5

画出不同排序算法在 20 个随机样本的平均运行时间与输入规模 n 的关系，如下图所示：



2. 分析

观察上图可知，虽然插入、冒泡、选择排序算法的平均时间复杂度一致，但冒泡排序的运行时间实际值大于其他两个，究其原因应该是：使用了未优化的冒泡排序算法，且选择排序一般情况下（序列有序性不高）要快于冒泡排序，因为其交换次数少。而插入排序优于选择排序是因为插入排序算法的时间复杂度最优情况为$O(n)$，最坏情况才是$O(n^2)$，而选择排序的时间复杂度最优和最坏情况都是$O(n^2)$。

两个平均时间复杂度为$O(n\log n)$的排序算法远远优于其他三个为$O(n^2)$的算法，所需时间较其他三个甚至可忽略不计。

思考题

现在有 1 亿的数据，请选择合适的排序算法与数据结构，在有限的时间内完成进行排序。

- ✧ 因为 1 亿的数据经计算需要内存约为 95.37MB，故 1 亿的数据可以存放在内存中。
- ✧ 我将 1 亿个随机数据存放在数组中，用比较排序中的最快的快速排序对 1 亿数据进行 5 次排序，统计得到排序平均运行时间为 39882.8ms。
- ✧ 我仍觉得过慢，我了解得到可以用一种更快的排序算法进行排序——计数排序。这种排序算法是一个非基于比较的排序算法，它的优势在于对一定范围内的整数排序时复杂度为 O（ n + k ）（其中 k 是整数的范围），快于任何比较排序算法。

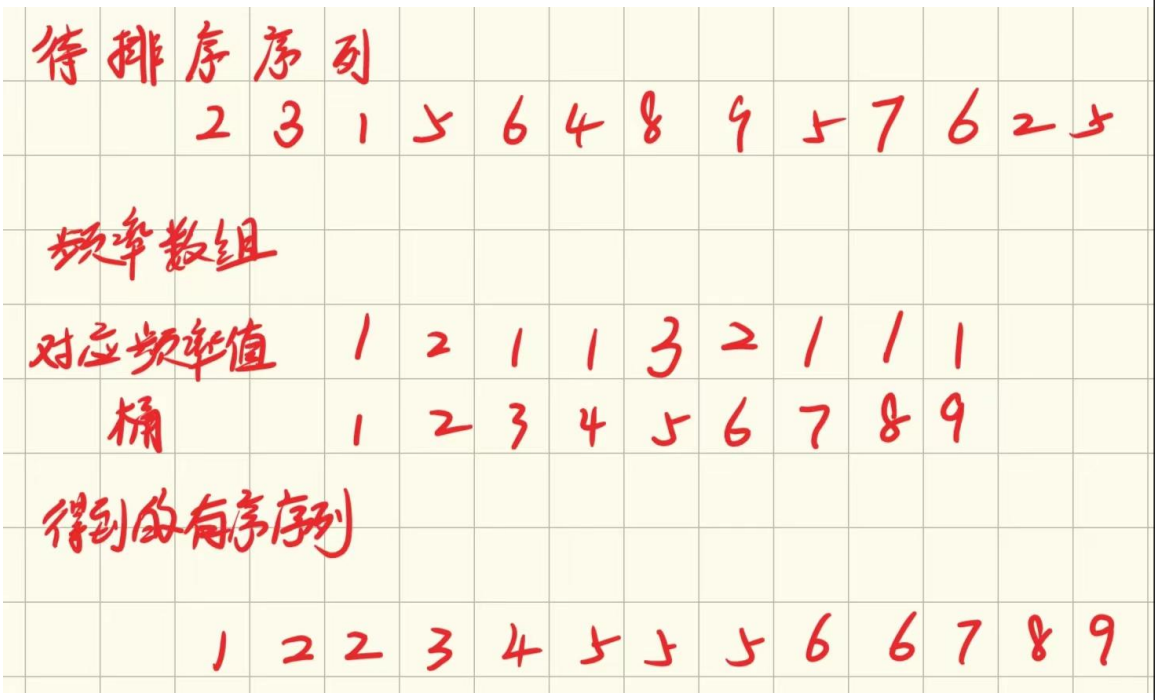
➢ 计数排序

设计原理或思路

开辟一个新数组作为频率数组，数组中的每个元素用来作为桶存待排序序列中每个元素出现的频率。然后遍历待排序序列，将每个元素放入对应的桶中（频率值加 1），最后遍历频率数组，并将每个元素存到另一个新序列中，得到排好序的序列。

```
计数排序伪代码：
CountingSort(A)
let C[0..max] be a new array
index=0
for i = 0 to max-1
    C[i]=0
for j=0 to A.length-1
    C[A[j]]=C[A[j]]+1;
//C[i] now contains the number of elements equal to i.
for i =0 to max-1
    while C[i]>0
    B[index]=i
    index=index+1
    C[i]=C[i]-1
```

#如下图所示，我以 2, 3, 1, 5, 6, 4, 8, 9, 5, 7, 6, 2, 5 这个序列为例，展示下计数排序的过程。



✧ 我将 1 亿个随机数据存放在数组中，用上面介绍的计数排序对 1 亿数据进行 10 次排序，统计得到排序平均运行时间仅为 91.1ms。

✧ 可以看出，计数排序的时间消耗比比较排序中最快的算法——快速排序仍要快出非常多。

五、 实验经验总结或体会

1） 具体问题需要具体分析

- 对于数量级较小的序列，可以选择时间复杂度为$O(n^2)$的算法，可以很快成功排序。若数据量＞10^4，则应该使用时间复杂度为$O(nlogn)$的算法，例如快排和归并排序。对于数据量特别巨大的情况，例如＞10^7的情况，则可以考虑使用计数排序。

2） 可以考虑对算法的不足进行优化

- 如冒泡排序，在算法操作时存在可优化的空间，冒泡排序算法需全部循环运行结束才结束，因此可能存在在循环结束前，序列已经有序，但仍需要进行循环，这是无意义的，将造成时间浪费。可以通过定义标志变量判断冒泡的一趟是否进行交换，如果未进行交换则序列已经有序，break 出循环减少无效循环，从而优化算法。
- 如快速排序，当数据大小大致成二分分布时，有很好的性能，但当数据恰好倒序或恰好正序时，则性能很差，时间复杂度为$O(n^2)$。我们可以使用三数取中选取基准。它的思想是：选取数组开头，中间和结尾的元素，通过比较，选择中间的值作为快排的基准。这可以优化快速排序算法。

深圳大学学生实验报告用纸

指导教师批阅意见：   成绩评定：   指导教师签字： 2023 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。