《数据结构与算法》实验报告

学生姓名		**院（系）**	计算机科学与技术
学　　号		**专　业**	
实验时间		**实验地点**	
实验项目	**实验 5：排序算法实验比较**		

实验目的：将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。

实验要求：灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。

实验内容：排序算法的实现与实验比较

实现一组经典的排序算法，通过实验数据的设计，考察不同规模和分布的数据对排序算法运行时间影响的规律，验证理论分析结果的正确性。

1．实现以下三组排序方法中的一组排序算法：

（1）冒泡排序和快速排序；

（2）插入排序和希尔排序；

（3）选择排序和堆排序。

2．产生不同规模和分布的数据，以“图或表”的方式给出输入规模和分布对排序方法运行时间变化趋势的影响（画出 T(n)的曲线）。并与理论分析结果比较。

3．将上述“图或表”采用图片等形式贴在实验报告中，与作适当分析或说明。

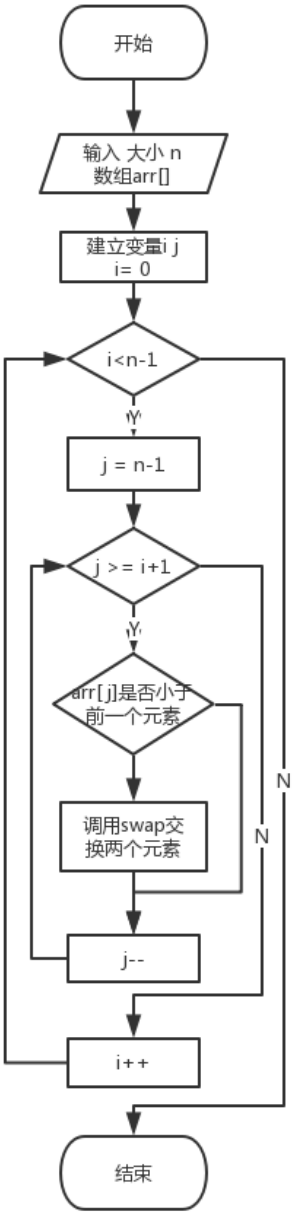
数据结构定义：

int arr[]　/* 用于排序的数组 */

算法设计与分析（要求画出核心内容的程序流程图）：

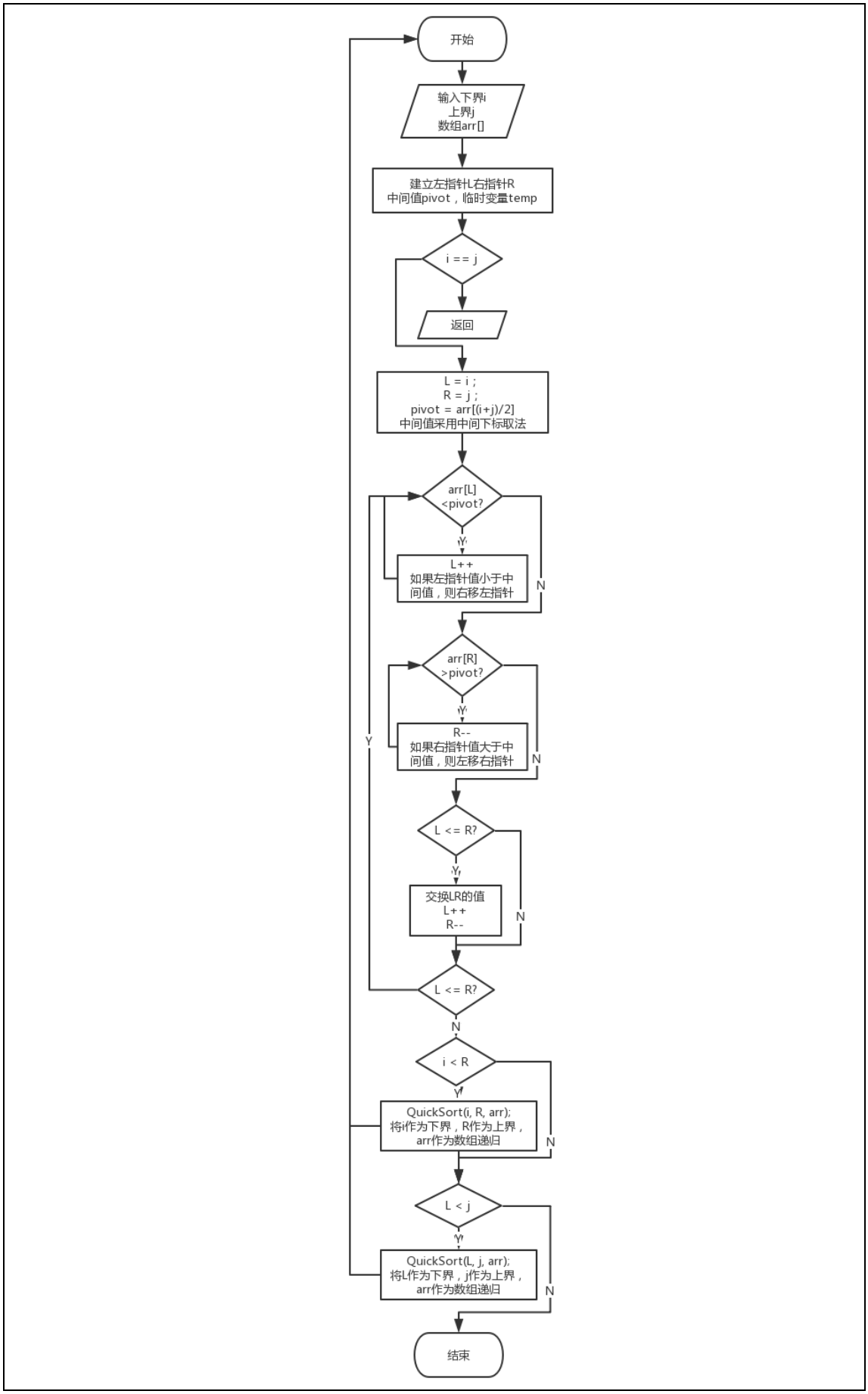
1．冒泡排序 Bubble Sort

冒泡排序基本思想是对所有的元素进行遍历，每相邻的两个元素比较大小，较小元素交换到前面，所有元素遍历完成后即排序完成。冒泡排序稳定但是时间消耗很大。时间复杂度为$O(n^2)$.



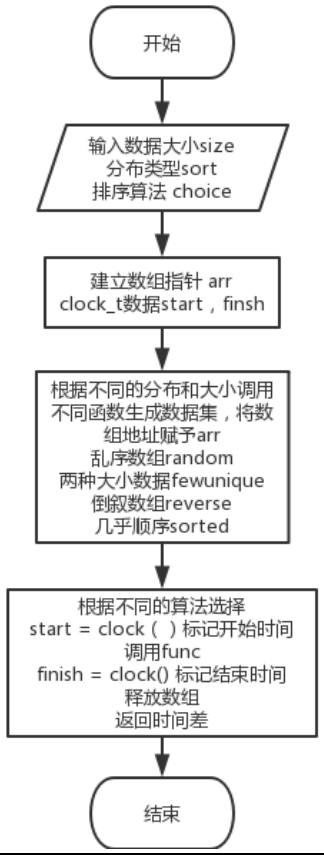
2. 快速排序 Quick Sort

快排的基本思想是分而治之，也是递归的思想。具体为取序列的中间值，以中间值为界限，从左右端向中间靠近，如果存在左端大于中间值，右端小于中间值，则交换左右值，使得中间值两边的值都小于或大于中间值，此时将中间值下标的两端作为新的序列的上界或下界进行递归，最后所有序列都排序完成。时间复杂度为O(nlogn).


开始
输入下界i
上界j
数组arr[]
建立左指针L右指针R
中间值pivot，临时变量temp
i == j
返回
L = i ;
R = j ;
pivot = arr[(i+j)/2]
中间值采用中间下标取法
arr[L]
<pivot?
Y
L++
如果左指针值小于中
间值，则右移左指针
N
arr[R]
>pivot?
Y
R--
如果右指针值大于中
间值，则左移右指针
N
Y
L <= R?
Y
交换LR的值
L++
R--
N
L <= R?
N
i < R
Y
QuickSort(i, R, arr);
将i作为下界，R作为上界，
arr作为数组递归
N
L < j
Y
QuickSort(L, j, arr);
将L作为下界，j作为上界，
arr作为数组递归
N
结束

3. 测试函数 Test

测试函数输入参数为数据大小，分布和测试用的排序算法，然后生成不同规模测试数据对算法测试，并计算排序所用时间



实验测试结果及结果分析：

本次测试数据大小从 1000 到 300000，数量级为 1e3 和 1e4.其中 1e4 数量级的数据从 10000 测到 300000，步长为 10000.

数据分布共有 4 中，RANDOM，FEWUNIQUE，REVERSE，SORTED。

RANDOM：完全随机数据序列，数据大小从 0 到 size-1，乱序

FEWUNIQUE：只有两种大小的数据 5 和 10，数据是乱序放置

REVERSE：大小从 0 到 size-1，顺序为倒序

SORTED：几乎已经排好序的数据，只有少部分打乱。

测试结果表格形式给出，对于不同大小的数据，以四种分布对两种算法进行时间测试，最后结果制成折线图（分布默认乱序）。

表格：

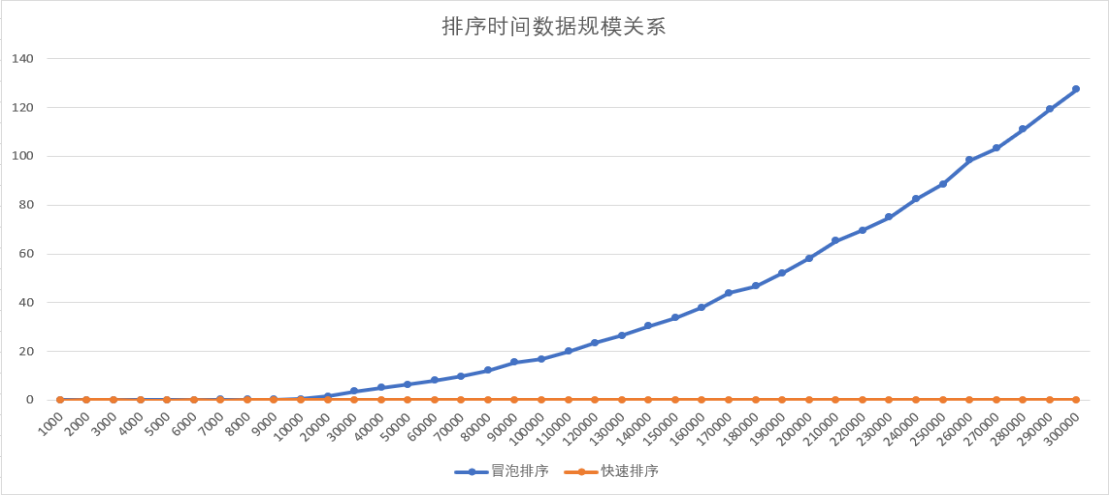
SIZE	TYPE	RANDOM	FEWUNIQUE	REVERSE	SORTED
1000	Bubble Sort	0.01200s	0.01400s	0.01000s	0.00700s
	Quick Sort	0.00100s	0.00000s	0.00000s	0.00000s
2000	Bubble Sort	0.04600s	0.03600s	0.04600s	0.03000s
	Quick Sort	0.00100s	0.00000s	0.00100s	0.00000s
3000	Bubble Sort	0.04600s	0.08400s	0.10300s	0.07100s
	Quick Sort	0.00200s	0.00000s	0.00000s	0.00100s
4000	Bubble Sort	0.05200s	0.05200s	0.05300s	0.03600s
	Quick Sort	0.00000s	0.00000s	0.00000s	0.00000s
5000	Bubble Sort	0.09000s	0.06600s	0.08500s	0.05100s
	Quick Sort	0.00100s	0.00000s	0.00100s	0.00000s
6000	Bubble Sort	0.12500s	0.10700s	0.11900s	0.07500s
	Quick Sort	0.00100s	0.00100s	0.00000s	0.00000s
7000	Bubble Sort	0.18400s	0.14600s	0.16500s	0.10900s
	Quick Sort	0.00100s	0.00000s	0.00000s	0.00000s
8000	Bubble Sort	0.23600s	0.20300s	0.21700s	0.13100s
	Quick Sort	0.00100s	0.00100s	0.00000s	0.00100s
9000	Bubble Sort	0.32400s	0.25000s	0.27400s	0.17100s
	Quick Sort	0.00100s	0.00100s	0.00100s	0.00100s
SIZE	TYPE	RANDOM	FEWUNIQUE	REVERSE	SORTED
10000	Bubble Sort	0.38800s	0.31800s	0.33600s	0.21600s
	Quick Sort	0.00100s	0.00000s	0.00000s	0.00100s
20000	Bubble Sort	1.56600s	1.27500s	1.33800s	0.88100s
	Quick Sort	0.00300s	0.00100s	0.00100s	0.00100s
30000	Bubble Sort	3.71100s	2.99000s	3.06500s	2.05900s
	Quick Sort	0.00400s	0.00200s	0.00100s	0.00200s
40000	Bubble Sort	5.70900s	5.60100s	5.77500s	3.64500s
	Quick Sort	0.00500s	0.00200s	0.00200s	0.00300s
50000	Bubble Sort	6.88700s	5.42900s	9.11200s	5.32700s
	Quick Sort	0.00500s	0.00300s	0.00200s	0.00300s
60000	Bubble Sort	8.16100s	5.87100s	12.05800s	6.61800s
	Quick Sort	0.00600s	0.00300s	0.00200s	0.00400s

70000	Bubble Sort	10.03400s	7.13300s	17.21000s	8.55800s
	Quick Sort	0.00600s	0.00400s	0.00300s	0.00600s
80000	Bubble Sort	11.78100s	8.82000s	21.61800s	11.00200s
	Quick Sort	0.00700s	0.00400s	0.00300s	0.00600s
90000	Bubble Sort	14.41100s	11.11500s	26.63200s	13.07600s
	Quick Sort	0.00700s	0.00500s	0.00400s	0.00600s

SIZE	TYPE	RANDOM	FEWUNIQUE	REVERSE	SORTED
100000	Bubble Sort	16.69900s	13.60100s	32.70700s	15.75600s
	Quick Sort	0.00900s	0.00600s	0.00500s	0.00600s
110000	Bubble Sort	19.97700s	16.44200s	41.52700s	19.37300s
	Quick Sort	0.00800s	0.00600s	0.00500s	0.00800s
120000	Bubble Sort	23.44500s	20.22700s	48.54500s	22.43600s
	Quick Sort	0.00900s	0.00600s	0.00500s	0.00700s
130000	Bubble Sort	26.50300s	23.21800s	56.19900s	25.61600s
	Quick Sort	0.01000s	0.00700s	0.00700s	0.00900s
140000	Bubble Sort	30.31600s	27.46900s	66.12600s	30.05900s
	Quick Sort	0.01000s	0.01000s	0.00700s	0.00900s
150000	Bubble Sort	33.68200s	30.56700s	73.80400s	33.36700s
	Quick Sort	0.01000s	0.00800s	0.00600s	0.01000s
160000	Bubble Sort	38.05400s	34.83500s	83.97000s	37.41300s
	Quick Sort	0.01100s	0.00900s	0.00600s	0.01200s
170000	Bubble Sort	43.98400s	40.56500s	96.84700s	42.60100s
	Quick Sort	0.01100s	0.01600s	0.00900s	0.01200s
180000	Bubble Sort	46.74300s	43.70600s	105.30100s	47.25300s
	Quick Sort	0.01200s	0.01000s	0.00900s	0.01100s
190000	Bubble Sort	52.09700s	48.93700s	118.10100s	51.50100s
	Quick Sort	0.01100s	0.01000s	0.00800s	0.01300s
200000	Bubble Sort	58.05200s	54.24500s	131.45300s	57.08700s
	Quick Sort	0.01300s	0.01100s	0.01100s	0.01500s
210000	Bubble Sort	65.29900s	60.57600s	144.97500s	62.73300s
	Quick Sort	0.01300s	0.01100s	0.00900s	0.01300s

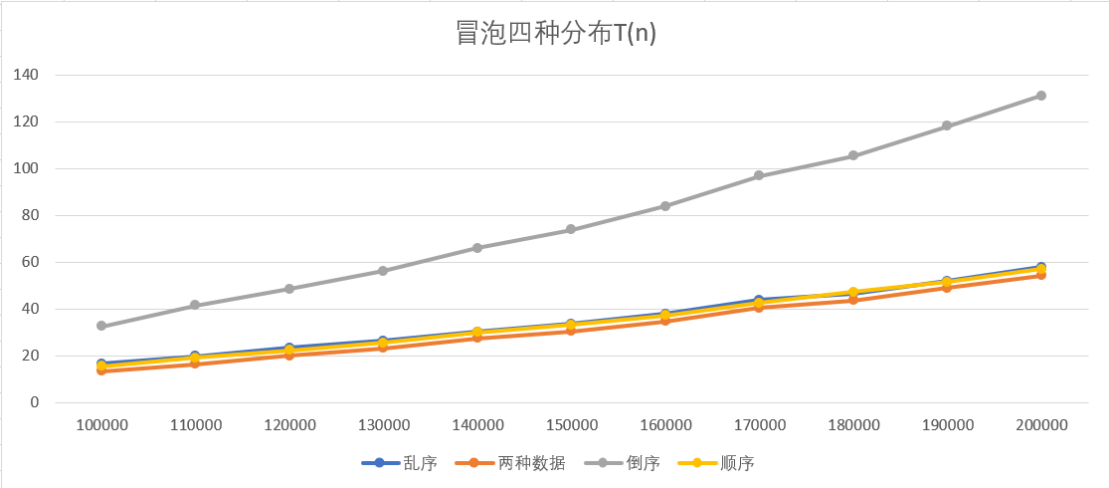
220000	Bubble Sort	69.62200s	65.91300s	160.71900s	70.52300s
	Quick Sort	0.01400s	0.01300s	0.01100s	0.01400s
230000	Bubble Sort	74.94500s	71.41100s	172.16800s	74.88600s
	Quick Sort	0.01500s	0.01400s	0.01100s	0.01300s
240000	Bubble Sort	82.50300s	79.61400s	188.13100s	80.71200s
	Quick Sort	0.01500s	0.01300s	0.01100s	0.01400s
250000	Bubble Sort	88.60200s	85.03300s	207.89300s	88.33000s
	Quick Sort	0.01700s	0.01700s	0.01100s	0.01500s
260000	Bubble Sort	98.35000s	95.29500s	229.35700s	95.97600s
	Quick Sort	0.01600s	0.01400s	0.01200s	0.01500s
270000	Bubble Sort	103.18000s	99.55900s	239.96600s	103.93400s
	Quick Sort	0.01600s	0.01600s	0.01200s	0.01700s
280000	Bubble Sort	110.98700s	109.58900s	264.05900s	111.28900s
	Quick Sort	0.01700s	0.01600s	0.01400s	0.01600s

290000	Bubble Sort	119.15100s	120.45200s	283.84300s	119.11100s
	Quick Sort	0.01700s	0.01700s	0.01400s	0.01600s
300000	Bubble Sort	127.41200s	124.47700s	301.54000s	127.74400s
	Quick Sort	0.02800s	0.02300s	0.01500s	0.01700s

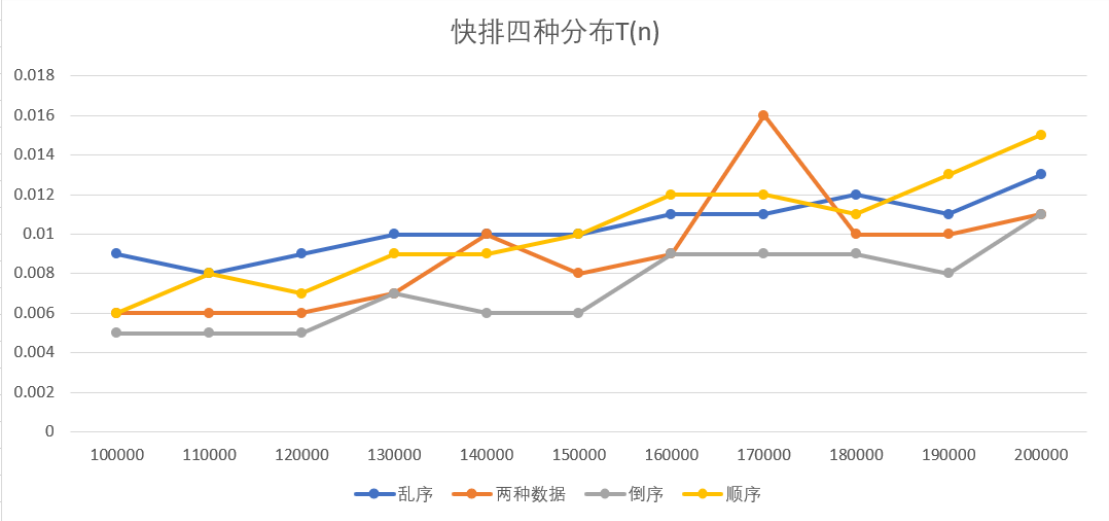


从图表中可以看出，随着数据量的增大，冒泡排序的时间曲线呈现抛物线形状，由于快排的时间消耗基本在零点几秒左右，所以快排基本呈现水平直线。

在数据大小为 10000 以内是，快排和冒泡的时间消耗基本上相同，而数据增大时，差距逐渐拉开，在数据大小到 300000 时，冒泡排序需要 127 秒，而此时快排仅需要 0.028 秒，消耗为 4500 倍。这与冒泡的时间复杂度$O(n^2)$基本符合，也与快排的$O(nlogn)$也基本符合。



由图可见，在四种分布的数据排序中，倒序对冒泡排序的时间消耗最大，而其余三种的时间消耗基本相同，这是因为冒泡排序要对每个相邻的元素进行交换，而倒序的数据每个相邻的元素大小都是颠倒的，所以每次比较都会调用 swap 函数，这样会大大增加的时间消耗。即便如此，最少时间消耗基本上还是在 30 到 40 秒以上。



由图可见，四种数据分布对快排的时间影响几乎相同，其中在数据量在 100000 到 150000 时，乱序的时间消耗比较高，而在数据量超过 150000 时，顺序消耗的时间最多，这与理论分析相符合。在完全顺序且 pivot 选在两端的最坏情况下，快排的时间复杂度回退化到 O(n^2)。但是本程序的 pivot 选在了序列的中间位置，减少了这种情况的发生。

倒序的时间消耗最小，这是因为倒序中取中值恰好为倒序的中间值，每一次 L 和 R 都会交换，而交换后的 L 和 R 值就为顺序拜访，所以只要遍历一遍，当 LR 相邻时，数据即排序完成。这样时间消耗是最少的。注意到在数据量为 11 万到 20 万时，快排的时间消耗基本在 0.02 秒以内，远比冒泡快的多。

问题及解决方法：

1. 如何产生不同大小的数组？

   使用 malloc 函数，动态申请不同大小的数组。

2. 如何计算排序所需要的时间？

   调用 c 自带的 clock()函数，该函数可以返回运行程序所需要的处理器时间周期，除以 CLOCKS_PER_SEC 常量可计算出对应的秒数。

源程序名称：1163450201.c

注意：正文文字为宋体小 4 号，图中文字为宋体 5 号。行距为多倍行距 1.25。
源程序与此报告打包提交，压缩包采用学号命名。