哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法课程

类型：必修

实验项目： 排序方法的实验比较

实验题目： 排序方法的实现与实验比较

实验日期： 2018.12.27

班级：1703002

学号：1170500913

姓名：熊健羽

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

1. **实验目的**
2. **实验要求及实验环境**
   1. 实验要求

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。

* 1. 实验环境
     1. 硬件环境

CPU：Intel(R) Core(TM) i5-7200U @ 2.50GHz (64位)

GPU：Intel(R) HD Graphics 620

Nvidia GeForce 940MX

物理内存：8.00GB

磁盘：1TB HDD

128GB SSD

* + 1. 软件环境：

OS: Microsoft Windows 10 家庭中文版

编译环境：MinGW-W64

IDE：VS code、code::blocks

1. **设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）
   1. 逻辑设计
      1. 数据结构:

待排序的线性表：

struct records {

keytype key ;

fields other ;

}；

typedef records LIST[maxsize] ;

* + 1. 各个函数的功能及其调用关系：

void initRecords(LIST A, int num)；

功能：初始化线性表A，为每个元素生成随机数

调用本地函数：无

void bubbleSort(LIST A, int n)

功能：实现冒泡排序

调用本地函数：无

void printKey(LIST A, int n)

功能：依次打印线性表的关键字

调用本地函数：无

int findPivot(int i, int j, LIST A)

功能：快速排序中，找出A[i] – A[j]段的基准点。若A[i] – A[j]端均一样，无基准点，则返回-1

调用本地函数：无

void swap(int \*a, int \*b)

功能：交换两个地址中的值

调用本地函数：无

int partition(int i, int j, LIST A, int pivot)

功能：以pivot为基准点，划分A[i] – A[j]段，返回基准点的位置

调用本地函数：swap

void quickSort(int i, int j, LIST A)

功能：实现快速排序

调用本地函数：findPivot，partition

int main()

功能:主函数。生成随机数据，计算每次排序时间，并输出。

* 1. 物理设计

void initRecords(LIST A, int num)；

实现思路：利用rand()随机数种子，初始化线性表

void bubbleSort(LIST A, int n)

实现思路：

将待排序的记录看作是竖着排列的“气泡”，关键字较小的记录比较轻，从而要往上浮。对这个“气泡”序列进行n-1遍（趟）处理。所谓一遍（趟）处理，就是自底向上检查一遍这个序列，并注意两个相比较的关键字的顺序是否正确。如果发现顺序不对，即“轻”的记录在下面，就交换它们的位置。显然，处理1遍之后，“最轻”的记录就浮到了最高位置；处理2遍之后，“次轻”的记录就浮到了次高位置。在作第二遍处理时，由于最高位置上的记录已是“最轻”的，所以不必检查。一般地，第i遍处理时，不必检查第i高位置以上的记录的关键字，因为经过前面i-1遍的处理，它们已正确地排好序。

int findPivot(int i, int j, LIST A)

实现思路：

v =从A[i].key到A[j].key 最先找到的两个不同关键字中的最大者。（若A[i].key,…A[j].key之中至少有两个关键字不相同）

int partition(int i, int j, LIST A, int pivot)

实现思路：

(1)令游标r 从右(初始时r = j )向左扫描，越过关键字大于等于v 的所有记录，直到遇到A[r].key<v；A[l]=A[r]。令游标l 从左(初始时l = i )向右扫描，越过关键字小于等于v 的所有记录，直到遇到A[l].key≥v；A[r]=A[l]。

(2) 测试l 和r：若l < r，则转(1)；否则(l = r,)转(3)；

(3) 基准定位：A[l]=v(基准记录v定位于其排序后的最终位置).此时A[i…j]被划分成为独立的两部分：

A[i…l-1].key ≤ A[l].key=v ≤ A[l+1…j].key

void quickSort(int i, int j, LIST A)

实现思路：

任选一个记录的关键字作为基准元素(控制关键字、枢轴)

将待排序的记录划分成独立的两部分，其中一部分的所有记录关键字都比另外一部分的所有记录关键字都小（划分交换，一次划分排序、一遍快速排序）

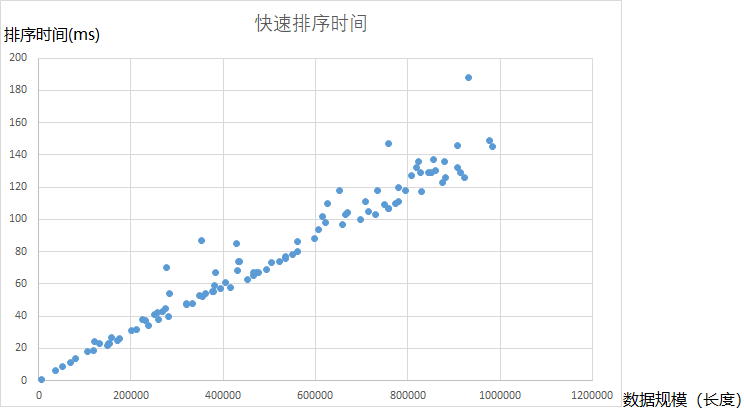
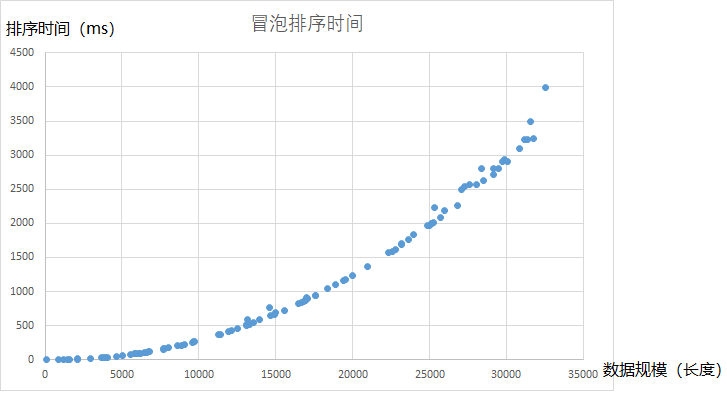
递归地对这两部分进行划分排序，以达到整个记录序列变成有序序列。

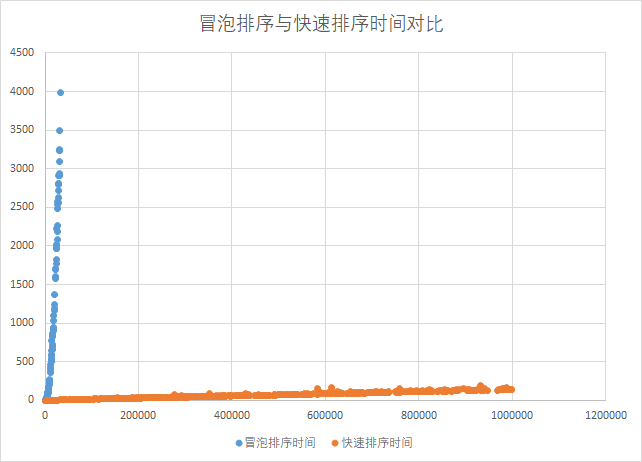
1. **测试结果**

生成的data\_bubble.txt和data\_quick.txt文件，为测试结果数据



两组数据，分别用Excel画图:





结论：

理论上：冒泡排序的时间复杂度为O(n^2),快速排序的时间复杂度为O(n\*log2n)

图中可看出，随着数据规模的增大，冒泡排序的耗时远远大于快速排序的耗时。

冒泡排序的曲线大致符合n^2的增长趋势，而快速排序的耗时也大致符合n\*log2n。

1. **经验体会与不足**

经验体会：

排序算法的选择很关键，特别是当数据很大时，O(n^2) 与O(n\*log2n)的复杂度有质的区别。

不足：

由于能力、时间所限，只生成了均匀分布的数据进行测试，未生成正态分布的数据，测试不够完备。

由于时间所限，计算机性能有限，未生成更大规模的数据，分析存在一定的误差。

1. **附录：源代码（带注释）**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <stdbool.h>

#include <time.h>

#define MAX 1000000

**long** exch\_bubble = 0;

**long** exch\_quick = 0;

**typedef** **struct**

{

**int** key;

**int** other;

} records;

**typedef** records LIST[MAX];

LIST B, Q;

**void** initRecords(LIST A, **int** num)

{

srand(time(NULL));

**for** (**int** i = 0; i < num; i++)

A[i].key = rand();

}

**void** bubbleSort(LIST A, **int** n)

{

bool isExchange;

**for** (**int** i = 0; i < n - 1; i++)

{

isExchange = false;

**for** (**int** j = n - 1; j > i; j--)

{

**if** (A[j].key < A[j - 1].key)

{

isExchange = true; *//该轮无序*

records temp = A[j];

A[j] = A[j - 1];

A[j - 1] = temp;

}

}

**if** (!isExchange) *//该轮有序，不用继续排序了*

**return**;

}

}

**void** printKey(LIST A, **int** n)

{

**for** (**int** i = 0; i < n; i++)

printf("%d ", A[i].key);

printf("**\n**");

}

*/\*find the pivot node's value\*/*

**int** findPivot(**int** i, **int** j, LIST A)

{

**int** fisrtKey = A[i].key;

**for** (**int** k = i + 1; k <= j; k++)

{

**if** (A[k].key > fisrtKey)

**return** A[k].key;

**if** (A[k].key < fisrtKey)

**return** fisrtKey;

}

**return** -1; *//i ~ j 已经有序*

}

**void** swap(**int** \*a, **int** \*b)

{

**int** temp = \*a;

\*a = \*b;

\*b = temp;

}

**int** partition(**int** i, **int** j, LIST A, **int** pivot)

{

**int** r = i;

**int** l = j;

**while** (r < l)

{

swap(&(A[r].key), &(A[l].key)); *//交换r l的key*

**while** (A[r].key < pivot)

r++;

**while** (A[l].key >= pivot)

l--;

}

**return** l; *//返回基准点的下标*

}

**void** quickSort(**int** i, **int** j, LIST A)

{

**int** pivot;

**if** ((pivot = findPivot(i, j, A)) != -1)

{

**int** k = partition(i, j, A, pivot);

quickSort(i, k, A);

quickSort(k + 1, j, A); *//分别对两边进行递归的排序*

}

}

**int** main()

{

**int** num; *//数据组数*

srand(time(NULL));

*/\*快速排序\*/*

freopen(".**\\**data\_quick.txt", "w", stdout);

**for** (num = 0; num < 500; num++)

{

**int** len = rand() \* 1000 % MAX; *// 数据长度随机*

initRecords(Q, len); *// 随机生成一组排序数据*

**int** start = clock();

quickSort(0, len - 1, Q);

**int** quick\_time = (clock() - start);

printf("%d**\t**%d**\n**", len, quick\_time); *//生成时间数据*

}

*/\*冒泡排序\*/*

freopen(".**\\**data\_bubble.txt", "w", stdout);

**for** (num = 0; num < 100; num++)

{

**int** len = rand() % MAX; *// 数据长度随机*

initRecords(B, len); *// 随机生成一组排序数据*

**int** start = clock();

bubbleSort(B, len);

**int** bubble\_time = (clock() - start);

printf("%d**\t**%d**\n**", len, bubble\_time); *//生成数据*

}

**return** 0;

}