哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：排序算法及其应用

实验题目：内存排序算法

实验日期：2023年10月26日

班级：2237102

学号：2022111654

姓名：李宸

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 李秀坤 |

**一、实验目的**

本实验主要要求正确生成若干随机数并对其实现插入排序，快速排序，归并排序，最后绘制折线图以比较其时间性能。

1. **实验要求及实验环境**

实验要求：

**1. 从文本文件中将两行数据读入内存，其中第一行有一个整数n(n≤100000)， 表示待排序序列的长度，第二行有 n 个整数，用空格隔开，表示待排序序列。**

**2. 实现归并排序、快速排序算法，输出排序好的序列，并记录算法运行时间。**

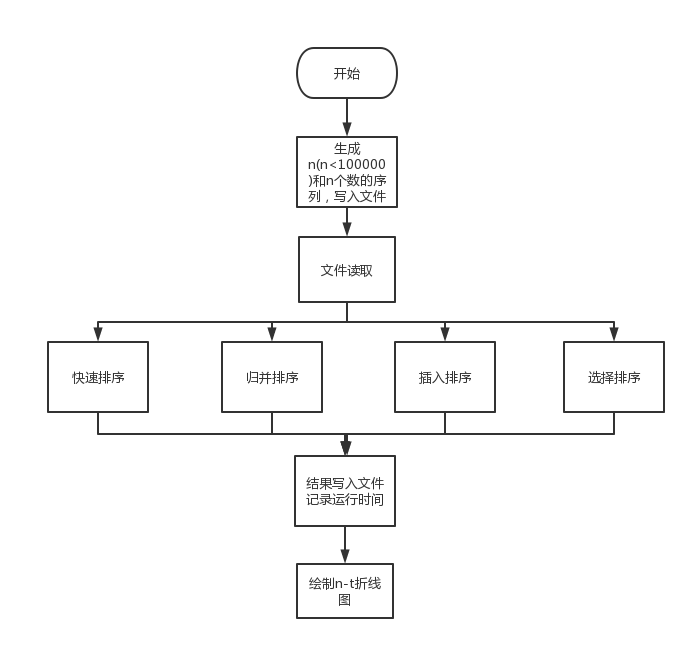
**3. 实现选择排序算法或插入排序算法，并将其运行时间与归并排序、快速排 序算法比较，随机生成多个适当规模的数据进行实验并绘制折线图，反映不 同算法运行时间随着输入规模的变化趋势，并与理论分析结果进行比较。**

实验环境：win11 vscode1.71.2version

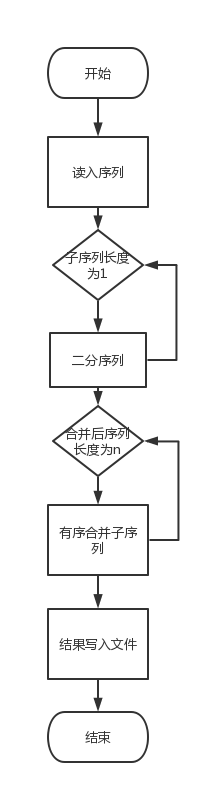
**三、设计思想**（本程序中的用到的所有数据类型的定义，各程序模块之间的调用关系、核心算法的流程图或主要步骤）

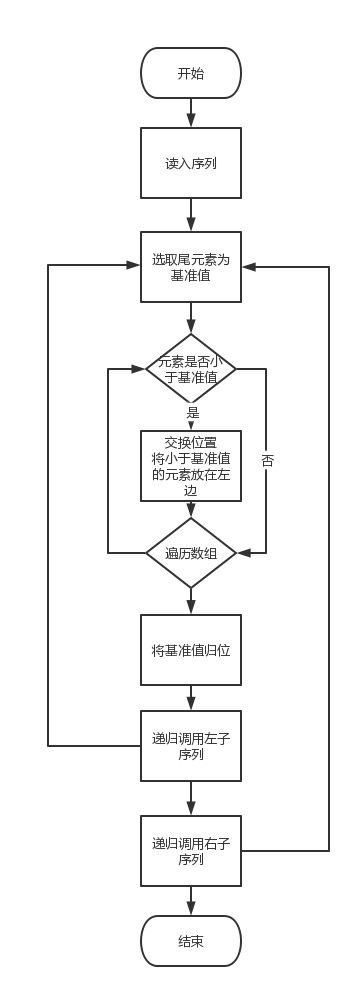
1．逻辑设计

整体流程图如下

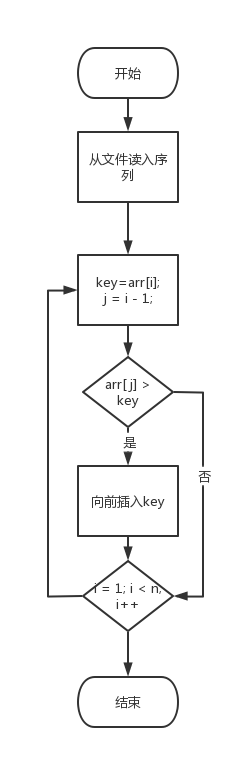


归并排序整体流程图如下：



快速排序整体流程图如下：

插入排序整体流程图如下：



生成随机数算法：

由srand(time(0))设置随机数种子。由于连续生成100组数，对时间精度要求较高，使用由srand(ts.tv\_nsec)使用纳秒部分作为种子值。

由于windows系统中rand()函数最多生成32767，故使用以下算法生成小于100000的数：

n1 = rand() % 1001;

n2 = rand() % 101;

n = n1 \* 100 + n2;

1. 物理设计

函数表如下：

|  |  |  |
| --- | --- | --- |
| 函数名 | 类型 | 功能 |
| Random | void | 生成随机序列并写入文件 |
| getCurrentTime | double | 获取当前时间 |
| writeToFile | void | 排序后序列写入文件 |
| partition | int | 根据基准值排序，返回基准值 |
| quickSort | void | 递归实现快速排序 |
| merge | void | 合并两个子序列 |
| mergeSort | void | 递归实现归并排序 |
| insertionSort | void | 实现插入排序 |
| main | int | 主函数，实现文件读取等 |

变量表如下：

|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 功能 |
| Size | int | 数据个数 |
| arr[size] | int | 存储序列的数组 |
| file | FILE \* | 进行文件操作，读入序列 |
| timefile | FILE \* | 进行文件操作，写入n-t |
| startTime | double | 起始时间 |
| endTime | double | 结束时间 |
| elapsedTime | double | 算法运行时间 |
| low | int | 快速排序头元素位置 |
| high | int | 快速排序尾元素位置 |
| pivot | int | 快速排序基准值 |
| L[n1] | int | 归并排序左子序列 |
| R[n2] | int | 归并排序右子序列 |
| n1 | int | 归并排序左子序列长度 |
| n2 | int | 归并排序右子序列长度 |
| key | int | 插入排序待插入元素 |

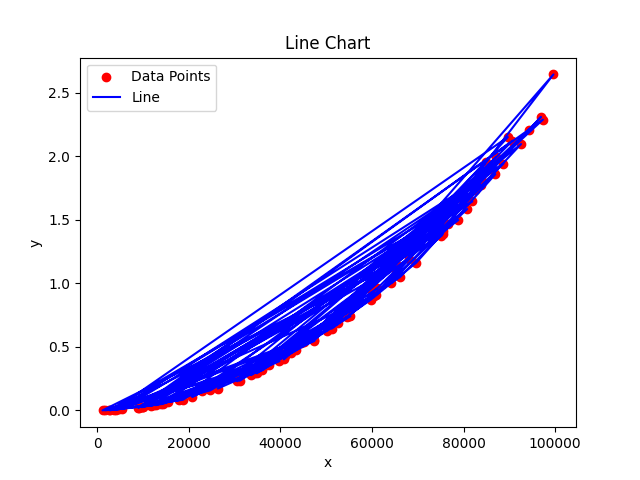
1. **测试结果**

测试数据由random函数生成，记录在data.txt

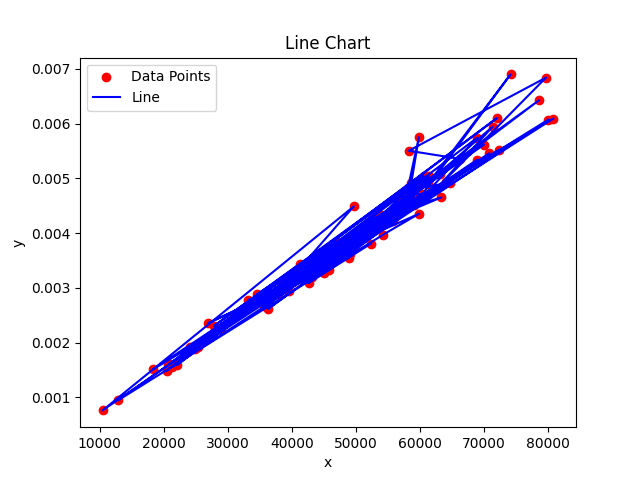
测试结果：排序后序列分别储存在对应排序sorted\_sequence.txt中

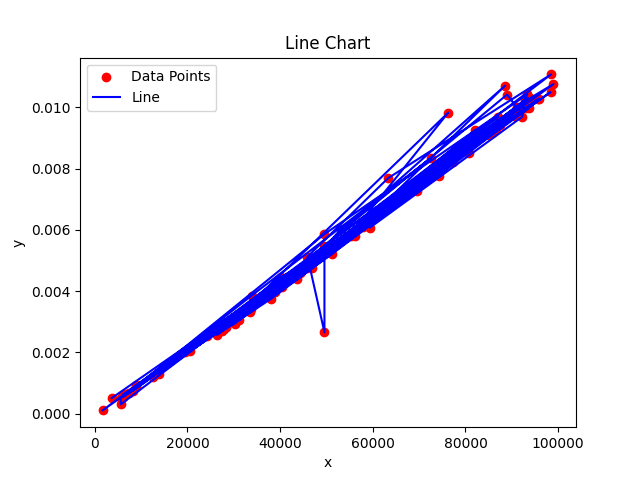
n-t数据存储在相应排序-time.txt中

由python绘制的折线图如下：

插入排序：

快速排序：

归并排序：



**五、经验体会与不足**

通过本次实验，我充分掌握了快速排序，归并排序，插入排序的原理，并通过绘制折线图分析其时间性能，在此基础上可以尝试实现堆排序算法（基于优先级队列）和基数排序算法，并将其运行时间与上述算法进行实验比较。

**六、附录：源代码（带注释）**

**快速排序：**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#define N 100

void random()

{

FILE \*file = fopen("data.txt", "w");

int n, n1, n2;

srand(time(0)); // 设置随机数种子

struct timespec ts;

clock\_gettime(CLOCK\_REALTIME, &ts); // 获取当前系统时间

srand(ts.tv\_nsec); // 使用纳秒部分作为种子值

// 生成随机数

n1 = rand() % 1001;

n2 = rand() % 101;

n = n1 \* 100 + n2;

fprintf(file, "%d\n", n);

// 生成随机序列（每个元素 <10000）

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

{

int num = rand() % 10000;

fprintf(file, "%d ", num);

}

printf("%d random number success\n", n);

fclose(file);

}

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high)

{

int pivot = arr[high]; //基准值

int i = low - 1; //小于基准值的元素的位置

for (int j = low; j <= high - 1; j++)//将小于基准值的元素放在左边

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);//将基准值归位

return i + 1;

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void writeToFile(int arr[], int size, const char \*filename)

{

FILE \*file = fopen("quick\_sorted\_sequence.txt", "w");

if (file == NULL)

{

printf("无法打开文件 %s\n", filename);

return;

}

for (int i = 0; i < size; i++)

{

fprintf(file, "%d ", arr[i]);

}

fclose(file);

}

double getCurrentTime()

{

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec + tv.tv\_usec / 1000000.0;

}

int main()

{

for (int i = 0; i < N; i++) // 循环生成100次

{

random();

int size;

FILE \*file = fopen("data.txt", "r");

fscanf(file, "%d", &size);

int arr[size];

for (int i = 0; i < size; i++)

{

fscanf(file, "%d", &arr[i]);

}

double startTime = getCurrentTime();

quickSort(arr, 0, size - 1);

double endTime = getCurrentTime();

double elapsedTime = endTime - startTime;

writeToFile(arr, size, "quick\_sorted\_sequence.txt");

printf("已将排序后的序列写入文件 quick\_sorted\_sequence.txt\n");

printf("快速排序运行时间：%.6f 秒\n", elapsedTime);

FILE \*timefile = fopen("quick-time.txt", "a");

fprintf(timefile, "%d %.6f\n", size, elapsedTime);

fclose(timefile);

}

return 0;

}

**归并排序：**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#define N 100

void random()

{

FILE \*file = fopen("data.txt", "w");

int n, n1, n2;

srand(time(0)); // 设置随机数种子

struct timespec ts;

clock\_gettime(CLOCK\_REALTIME, &ts); // 获取当前系统时间

srand(ts.tv\_nsec); // 使用纳秒部分作为种子值

// 生成随机数

n1 = rand() % 1001;

n2 = rand() % 101;

n = n1 \* 100 + n2;

fprintf(file, "%d\n", n);

// 生成随机序列（每个元素 <10000）

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

{

int num = rand() % 10001;

fprintf(file, "%d ", num);

}

printf("%d random number success\n", n);

fclose(file);

}

void merge(int arr[], int left, int mid, int right) {

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[left + i];

for (j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void writeToFile(int arr[], int size, const char \*filename)

{

FILE \*file = fopen(filename, "w");

if (file == NULL)

{

printf("无法打开文件 %s\n", filename);

return;

}

for (int i = 0; i < size; i++)

{

fprintf(file, "k%dk ", i);

fprintf(file, "%d ", arr[i]);

}

fclose(file);

}

double getCurrentTime()

{

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec + tv.tv\_usec / 1000000.0;

}

int main()

{

for (int i = 0; i < N; i++) // 循环生成100次

{

random();

int size;

FILE \*file = fopen("data.txt", "r");

fscanf(file, "%d", &size);

int arr[size];

for (int i = 0; i < size; i++)

{

fscanf(file, "%d", &arr[i]);

}

double startTime = getCurrentTime();

mergeSort(arr, 0, size-1);

double endTime = getCurrentTime();

double elapsedTime = endTime - startTime;

writeToFile(arr, size, "merge\_sorted\_sequence.txt");

printf("已将排序后的序列写入文件 merge\_sorted\_sequence.txt\n");

printf("归并排序运行时间：%.6f 秒\n", elapsedTime);

FILE \*timefile = fopen("merge-time.txt", "a");

fprintf(timefile, "%d %.6f\n", size, elapsedTime);

fclose(timefile);

}

return 0;

}

**插入排序：**

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#define N 100

void random()

{

FILE \*file = fopen("data.txt", "w");

int n, n1, n2;

srand(time(0)); // 设置随机数种子

struct timespec ts;

clock\_gettime(CLOCK\_REALTIME, &ts); // 获取当前系统时间

srand(ts.tv\_nsec); // 使用纳秒部分作为种子值

// 生成随机数

n1 = rand() % 1001;

n2 = rand() % 101;

n = n1 \* 100 + n2;

fprintf(file, "%d\n", n);

// 生成随机序列（每个元素 <10000）

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

{

int num = rand() % 10000;

fprintf(file, "%d ", num);

}

printf("%d random number success\n", n);

fclose(file);

}

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void writeToFile(int arr[], int size, const char \*filename)

{

FILE \*file = fopen(filename, "w");

if (file == NULL)

{

printf("无法打开文件 %s\n", filename);

return;

}

for (int i = 0; i < size; i++)

{

fprintf(file, "%d ", arr[i]);

}

fclose(file);

}

double getCurrentTime()

{

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec + tv.tv\_usec / 1000000.0;

}

int main()

{

for (int i = 0; i < N; i++) // 循环生成100次

{

random();

int size;

FILE \*file = fopen("data.txt", "r");

fscanf(file, "%lld", &size);

int arr[size];

for (int i = 0; i < size; i++)

{

fscanf(file, "%d", &arr[i]);

}

double startTime = getCurrentTime();

insertionSort(arr, size);

double endTime = getCurrentTime();

double elapsedTime = endTime - startTime;

writeToFile(arr, size, "insert\_sorted\_sequence.txt");

printf("已将排序后的序列写入文件 insert\_sorted\_sequence.txt\n");

printf("插入排序运行时间：%.6f 秒\n", elapsedTime);

FILE \*timefile = fopen("insert-time.txt", "a");

fprintf(timefile, "%d %.6f\n", size, elapsedTime);

fclose(timefile);

}

return 0;

}

**此外利用以下python代码绘制折线图：**

import matplotlib.pyplot as plt

# 读取文本文件中的数据

with open('insert-time.txt', 'r') as file:

lines = file.readlines()

x = []

y = []

# 解析每一行的数据

for line in lines:

data = line.strip().split(' ')

x.append(float(data[0]))

y.append(float(data[1]))

# 绘制散点图

plt.scatter(x, y, color='red', label='Data Points')

# 绘制折线图

plt.plot(x, y, color='blue', label='Line')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Line Chart')

plt.legend()

plt.show()