《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 21计科 | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 实验布置日期 | | 2022.10.26 | | 提交  日期 | 2022.11.16 | | 成绩 |  |

课程实践实验5：查找算法的实现及性能测试与比较

## 问题描述及要求

(1)通过实验比较非递归的顺序查找、二分查找1和二分查找2，分别给出在成功和失败查找情况下的绝对运行时间和平均查找长度；

(2)通过实验比较非递归二分查找1和递归二分查找1在成功查找和失败查找时的绝对运行时间。

方案推荐：

(1)为保证在相同的表下进行不同查找，查找表设定为有序表；

(2)为分别测试成功和失败查找情况下的性能，可在有序表中存放1~2n-1 范围的n个奇数。用查找1~2n-1范围中的奇数模拟成功查找，用查找该范围中的偶数来模拟失败查找；

(3)为了能更客观比较出各个查找算法执行的性能，需要对表中的数据进行多次查找，设为m次；为了做尽量真实的模拟，m次的成功查找或失败查找时查找不同关键字的概率要求相同。

## 二、概要设计

### 1.内容理解

实验要求程序来测试不同的查找算法的性能。

### 2.功能列表

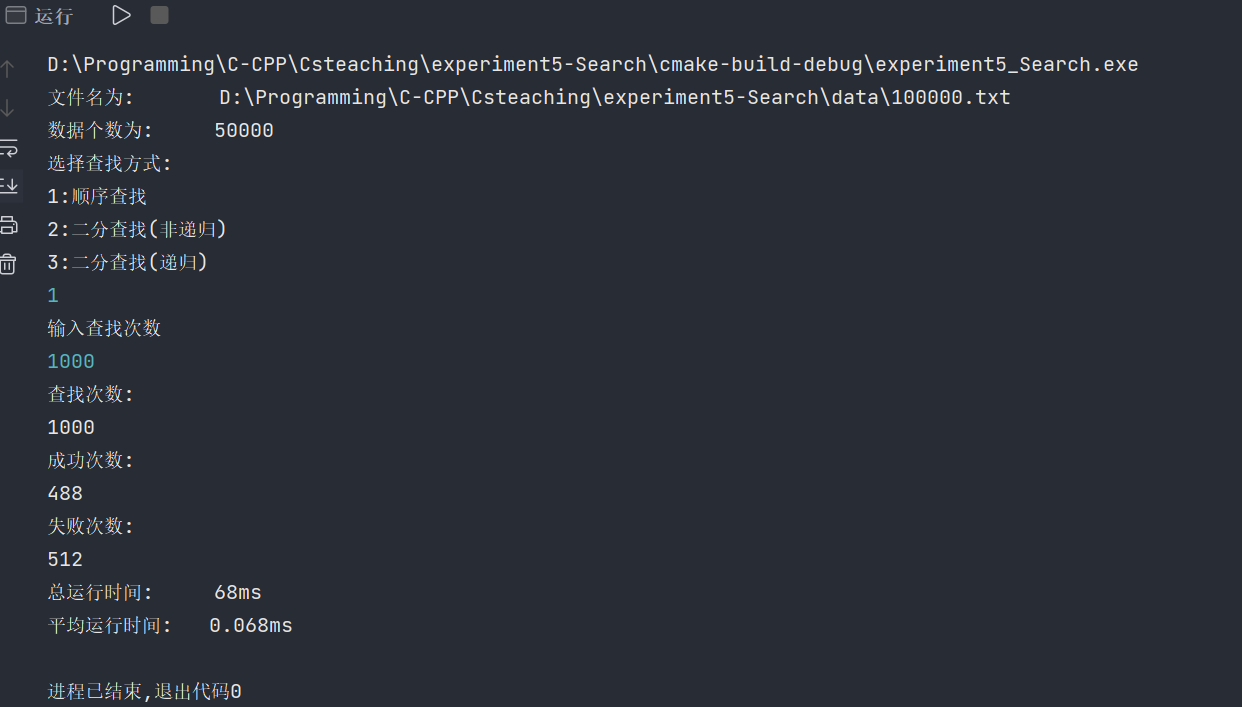
程序实现了以下的查找:

(1)顺序查找

(2)二分查找(非递归)

(3)二分查找(递归)

### 3.程序运行的界面设计



首先会显示出读取文件的文件名，其次会显示共读取的数据个数，然后屏幕上面才出现三个选项让用户选择查找的方式，选择完毕之后就会进行查找，并且给出查找成功和失败的次数，以及查找的时间.

### 4.总体设计思路

(1) class LineSearch

LineSearch类为线性表类，类内函数如下:

class LineSearch  
{  
private:  
 int data[**MaxSize**]{};  
 int length;  
public:  
 LineSearch(const int a[], int n); *//构造函数* ~LineSearch() = default; *//析构函数* int SeqSearch(int data\_to\_be\_searched); *//线性查找* int BinarySearch(int data\_to\_be\_searched); *//二分查找* int RecursiveBinarySearch(int low, int high, int data\_to\_be\_searched); *//二分查找* void Print(); *//输出数组*};

(2) int Search(LineSearch L, int data\_to\_be\_searched, int selected\_search\_method) *//单次查找，查找的数为 data\_to\_be\_searched*

单次查找，传入的形参分别是LineSearch类，需要查找的数和选择的查找方式

(3) void MultipleSearch(LineSearch L, int select\_search\_method, int search\_times) *//多次查找，查找次数为search\_times*

多次查找，查找方式为select\_search\_method，查找的次数为search\_times，其中，每次查找查询的数为随机数

## 三、详细设计

### 1**.获得程序运行时间**

需要#include<ctime>，使用ctime中的clock()函数，来给begin和end两个clock\_t类型的变量赋值,查找开始之前用begin来记录时间，查找结束后用end来记录时间，然后让end-begin来获取查找的函数的运行时间

### 2.int SeqSearch(int data\_to\_be\_searched)

线性查找，函数的形参为本次线性查找需要查询的数，时间复杂度，为线性时间

### 3. int BinarySearch(int data\_to\_be\_searched)

二分查找，形参为需要查找的数，时间复杂度，为指数时间

### 4. int RecursiveBinarySearch(int low, int high, int data\_to\_be\_searched)

递归二分查找，时间复杂度，由于使用了递归，因此运行时间会比非递归的略长

### 5. int RandomSearchNumber()

使用了C++11中的std::mt19937来生成随机数，同时使用了std::random\_device来生成随机数种子，使用std::uniform\_int\_distribution来分布随机数

## 四、实验结果

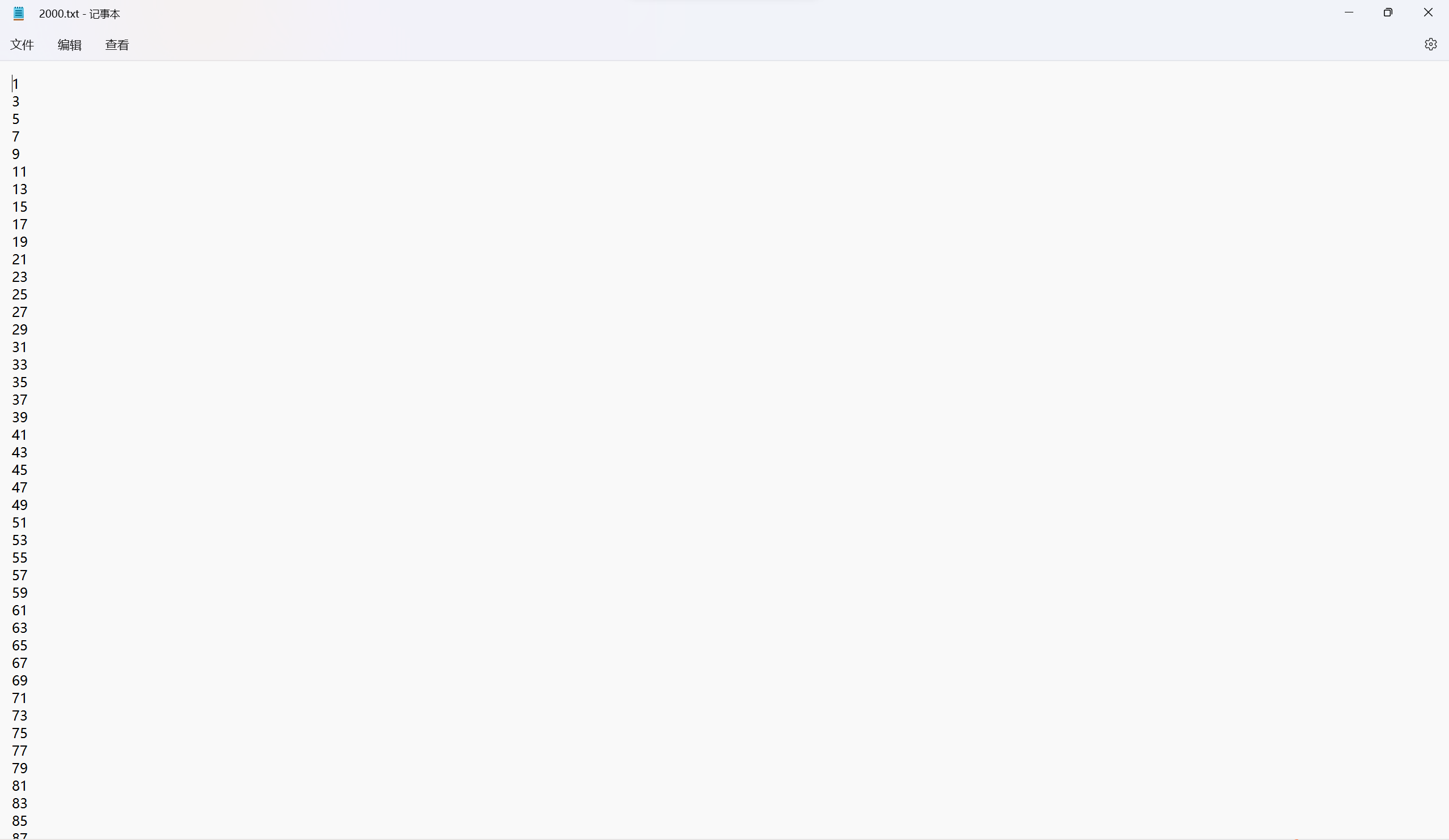
### 实验前测试:

测试输入:

data/2000.txt

文件中为1-2000中的所有奇数

程序中以此查找1-2000中的所有数字



测试目的:测试各个查找算法是否正确

正确输出:

图形用户界面

中度可信度描述已自动生成

实际输出:

顺序查找时: test\_output\_SeqSearch.txt

形状

中度可信度描述已自动生成

二分查找(非递归): test\_output\_BinarySearch.txt

图片包含 形状

描述已自动生成

二分查找(递归): test\_output\_RecursiveBinarySearch.txt

图形用户界面, 应用程序

描述已自动生成测试结论:正确  
实验时:

测试输出:

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

测试结果分析:

输入20000组数据，其中10000组成功的数据，10000组失败的数据

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 时间(ms) | | | | |  |
| 测试1 | 测试2 | 测试3 | 测试4 | 测试5 | 平均比较时间 |
| 线性查找 | 成功 | 438 | 444 | 480 | 443 | 493 | 459.6 |
| 失败 | 524 | 563 | 565 | 559 | 580 | 558.2 |
| 二分查找 | 成功 | 197 | 187 | 188 | 206 | 190 | 193.6 |
| 失败 | 242 | 188 | 287 | 224 | 208 | 229.8 |
| 递归二分查找 | 成功 | 233 | 278 | 220 | 229 | 189 | 229.8 |
| 失败 | 245 | 239 | 226 | 239 | 234 | 236.6 |

## 五、实验分析与探讨

线性查找在平均情况以及最坏情况下时间复杂度均为,即使对二分查找进行优化，也只能改变时间复杂度中的常量阶时间，很难产出数量级的变化。而产生此现象的原因是因为线性查找必须遍历整个列表，因此时间复杂度一定为。优点是线性查找对被查找的序列无要求，并且对线性表和链表都有着很好的兼容性

而二分查找每次都是与待查找数组中间的数相比较，每次都能使被查找的序列的长度减半，因此时间复杂度为，是一种较快的查找。缺点是二分查找对待查找的序列的要求较高。一旦序列没排序，二分查找就无法工作。一旦序列无法通过下标，比如链表，来快速获取数据时，二分查找的效率将变差.

递归二分查找的代码可读性更高，但是由于递归时需要调用堆栈，因此空间消耗会比非递归的大，同时递归的次数受到了堆栈大小的限制，在递归次数过多后可能会出现stackoverflow.

同时，由于查找成功之后就停止查找，而查找失败需要一直查找直到遇到终止条件，因此在所有查找方式中，查找成功的时间均小与查找失败的时间。

## 六、小结

查找的实验到此结束，在实验中我仍然有些不足之处，比如测试的数据不足，希望在以后能改进

补充:

若程序运行失败，需要在main.cpp的main函数中修改string filename值，将filename的值改为存储数据的地址.

## 附录：源代码

1. 实验环境  
   

编译器:mingw

gcc version 8.1.0 (x86\_64-posix-seh-rev0，Built by NinGW-w64 project)

2、源代码:

(1)main.cpp

#include <iostream>  
#include <ctime>  
#include <fstream>  
#include <random>  
  
#define MaxSize 100000  
  
class LineSearch  
{  
private:  
 int data[MaxSize]{};  
 int length;  
public:  
 LineSearch(const int a[], int n);  
  
 ~LineSearch() = default;  
  
 int SeqSearch(int data\_to\_be\_searched);  
  
 int BinarySearch(int data\_to\_be\_searched);  
  
 int RecursiveBinarySearch(int low, int high, int data\_to\_be\_searched);  
  
 void Print();  
};  
  
LineSearch::LineSearch(const int a[], int n)  
{  
 *//data = new int[n];* for (int i = 0; i < n; i++)  
 {  
 this->data[i + 1] = a[i]; *//data[0]设置为哨兵* }  
 this->length = n;  
}  
  
int LineSearch::SeqSearch(int data\_to\_be\_searched)  
{  
 int i = this->length;  
 this->data[0] = data\_to\_be\_searched;  
 while (data[i] != data\_to\_be\_searched)  
 {  
 i--;  
 }  
 return i;  
}  
  
int LineSearch::BinarySearch(int data\_to\_be\_searched)  
{  
 int mid;  
 int low = 1;  
 int high = this->length;  
 while (low <= high)  
 {  
 mid = (low + high) / 2;  
 if (data\_to\_be\_searched < data[mid])  
 {  
 high = mid - 1;  
 }  
 else if (data\_to\_be\_searched > data[mid])  
 {  
 low = mid + 1;  
 }  
 else  
 {  
 return mid;  
 }  
 }  
 return 0;  
}  
  
int LineSearch::RecursiveBinarySearch(int low, int high, int data\_to\_be\_searched)  
{  
 if (high < low)  
 {  
 return 0;  
 }  
 else  
 {  
 int mid = (low + high) / 2;  
 int i = data[mid];  
 if (data\_to\_be\_searched < data[mid])  
 { return RecursiveBinarySearch(low, mid - 1, data\_to\_be\_searched); }  
 else if (data\_to\_be\_searched > data[mid])  
 { return RecursiveBinarySearch(mid + 1, high, data\_to\_be\_searched); }  
 else  
 { return mid; }  
 }  
}  
  
void LineSearch::Print()  
{  
 for (int i = 0; i <= length; i++)  
 {  
 std::cout << data[i] << " ";  
 }  
 std::cout << std::endl;  
}  
  
LineSearch ReadFile(const std::string &filename) *//读取文件并且返回一个LineSearch*{  
 std::cout << "文件名为:\t" << filename << std::endl;  
 std::cout << "数据个数为:\t50000" << std::endl;  
 int length = 50000;  
 int count = 0;  
 int a[length];  
 std::ifstream fin(filename);  
 if (!fin.is\_open())  
 {  
 std::cout << "Open file wrong" << std::endl;  
 exit(0);  
 }  
 while (!fin.eof())  
 {  
 fin >> a[count];  
 *//std::cout<<a[count]<<" ";* count++;  
 }  
 LineSearch(L){a, length};  
 *//L.Print();* return L;  
}  
  
int RandomSearchNumber() *//随机生成数字*{  
 std::random\_device rd;*//用于生成随机数种子* std::mt19937 r\_eng(rd());*//随机数生成器* std::uniform\_int\_distribution<int> dis(1, 50000);*//随机数分布器 闭区间  
 //int i = dis(r\_eng)\*2+1;  
 //return i;* return dis(r\_eng);  
}  
  
int Search(LineSearch L, int data\_to\_be\_searched, int selected\_search\_method) *//单词查找，查找的数为 data\_to\_be\_searched*{  
 int location\_of\_the\_data\_to\_be\_searched = -1;  
 if (selected\_search\_method == 1)  
 {  
 location\_of\_the\_data\_to\_be\_searched = L.SeqSearch(data\_to\_be\_searched);  
 }  
 else if (selected\_search\_method == 2)  
 {  
 location\_of\_the\_data\_to\_be\_searched = L.BinarySearch(data\_to\_be\_searched);  
 }  
 else if (selected\_search\_method == 3)  
 {  
 location\_of\_the\_data\_to\_be\_searched = L.RecursiveBinarySearch(1, 50000, data\_to\_be\_searched);  
 }  
 else  
 {  
 location\_of\_the\_data\_to\_be\_searched = 0;  
 }  
 return location\_of\_the\_data\_to\_be\_searched;  
}  
  
void WhetherToBeFound(int location\_of\_the\_data\_to\_be\_searched, int data\_to\_be\_searched)  
{  
 if (location\_of\_the\_data\_to\_be\_searched == 0)  
 {  
 std::cout << "未找到" << data\_to\_be\_searched << std::endl;  
 }  
 else  
 {  
 std::cout << "在第" << location\_of\_the\_data\_to\_be\_searched << "号位置查找到了" << data\_to\_be\_searched  
 << std::endl;  
 }  
}  
  
void TestMultipleSearch(LineSearch L, int select\_search\_method, int search\_times)  
{  
 std::string output\_filename = R"(D:\Programming\C-CPP\Csteaching\experiment5-Search\output\test\_output\_RecursiveBinarySearch.txt)";  
 int success\_times = 0;  
 int fail\_times = 0;  
 std::fstream file\_out(output\_filename);  
 file\_out << "本次使用的查找方式为:\t二分查找(递归)\n";  
 for (int i = 0; i < search\_times; i++)  
 {  
 *//int data\_to\_be\_searched = RandomSearchNumber();* int data\_to\_be\_searched = i;  
 int address = Search(L, data\_to\_be\_searched, select\_search\_method);  
 if (address == 0)  
 {  
 fail\_times++;  
 file\_out << "本次查找的数据为:\t" << data\_to\_be\_searched << std::endl;  
 file\_out << "查找结果为:\t失败" << std::endl;  
 }  
 else  
 {  
 success\_times++;  
 file\_out << "本次查找的数据为:\t" << data\_to\_be\_searched << std::endl;  
 file\_out << "查找结果为:\t成功" << std::endl;  
 }  
 }  
 file\_out << "查找次数:\n" << search\_times << std::endl;  
 file\_out << "成功次数:\n" << success\_times << std::endl;  
 file\_out << "失败次数:\n" << fail\_times << std::endl;  
 file\_out.close();  
 std::cout << "查找次数:\n" << search\_times << std::endl;  
 std::cout << "成功次数:\n" << success\_times << std::endl;  
 std::cout << "失败次数:\n" << fail\_times << std::endl;  
}  
  
void MultipleSearch(LineSearch L, int select\_search\_method, int search\_times) *//多次查找，查找次数为search\_times*{  
 int success\_times = 0;  
 int fail\_times = 0;  
 for (int i = 0; i < search\_times; i++)  
 {  
 int data\_to\_be\_searched = RandomSearchNumber();  
 *//int data\_to\_be\_searched = i;* int address = Search(L, data\_to\_be\_searched, select\_search\_method);  
 if (address == 0)  
 {  
 fail\_times++;  
 }  
 else  
 {  
 success\_times++;  
 }  
 }  
 std::cout << "查找次数:\n" << search\_times << std::endl;  
 std::cout << "成功次数:\n" << success\_times << std::endl;  
 std::cout << "失败次数:\n" << fail\_times << std::endl;  
}  
  
int main()  
{  
 int select\_search\_method;  
 int search\_times;  
 std::string filename = R"(D:\Programming\C-CPP\Csteaching\experiment5-Search\data\100000.txt)";  
 auto L = ReadFile(filename);  
 std::cout << "选择查找方式:\n1:顺序查找\n2:二分查找(非递归)\n3:二分查找(递归)" << std::endl;  
 std::cin >> select\_search\_method;  
 std::cout << "输入查找次数" << std::endl;  
 std::cin >> search\_times;  
 auto begin = clock();  
 MultipleSearch(L, select\_search\_method, search\_times);  
 auto end = clock();  
 std::cout << "总运行时间:\t" << (double) (end - begin) << "ms" << std::endl;  
 std::cout << "平均运行时间:\t" << (double) (end - begin) / search\_times << "ms" << std::endl;  
 return 0;  
}