

《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

一、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 $1 \sim 2n-1$ 范围的 n 个奇数。用查找 $1 \sim 2n-1$ 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

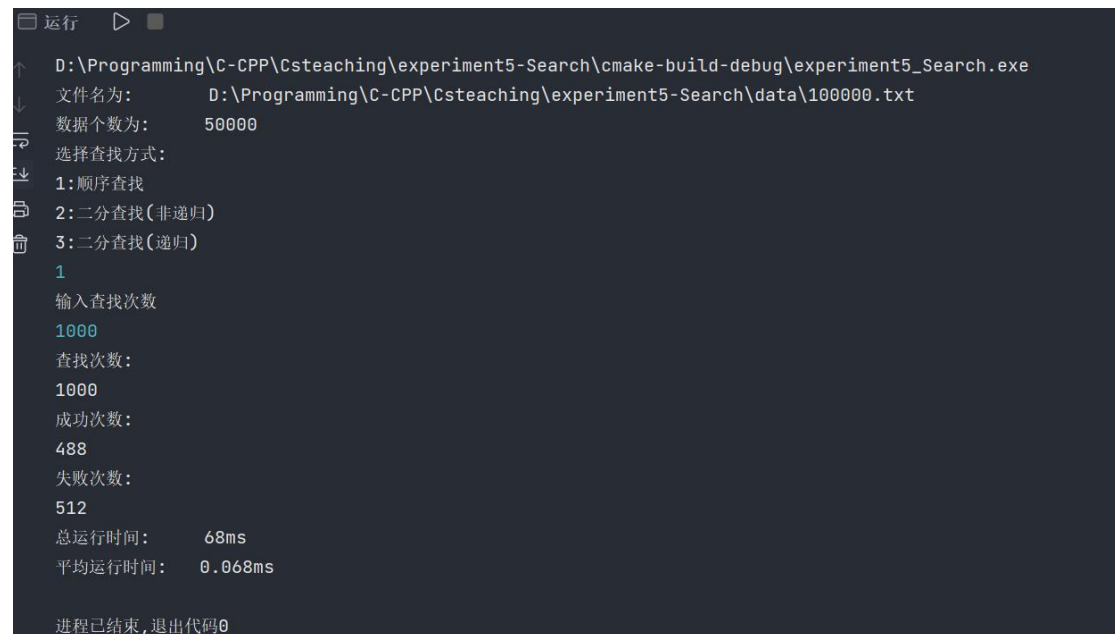
程序实现了以下的查找：

(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



```
运行
D:\Programming\C-CPP\Csteaching\experiment5-Search\cmake-build-debug\experiment5_Search.exe
文件名为: D:\Programming\C-CPP\Csteaching\experiment5-Search\data\100000.txt
数据个数为: 50000
选择查找方式:
1: 顺序查找
2: 二分查找(非递归)
3: 二分查找(递归)
1
输入查找次数
1000
查找次数:
1000
成功次数:
488
失败次数:
512
总运行时间: 68ms
平均运行时间: 0.068ms
进程已结束,退出代码0
```

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 $T(n) = O(n)$ ，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 $T(n) = O(\log n)$ ，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 $T(n) = O(\log n)$ ，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

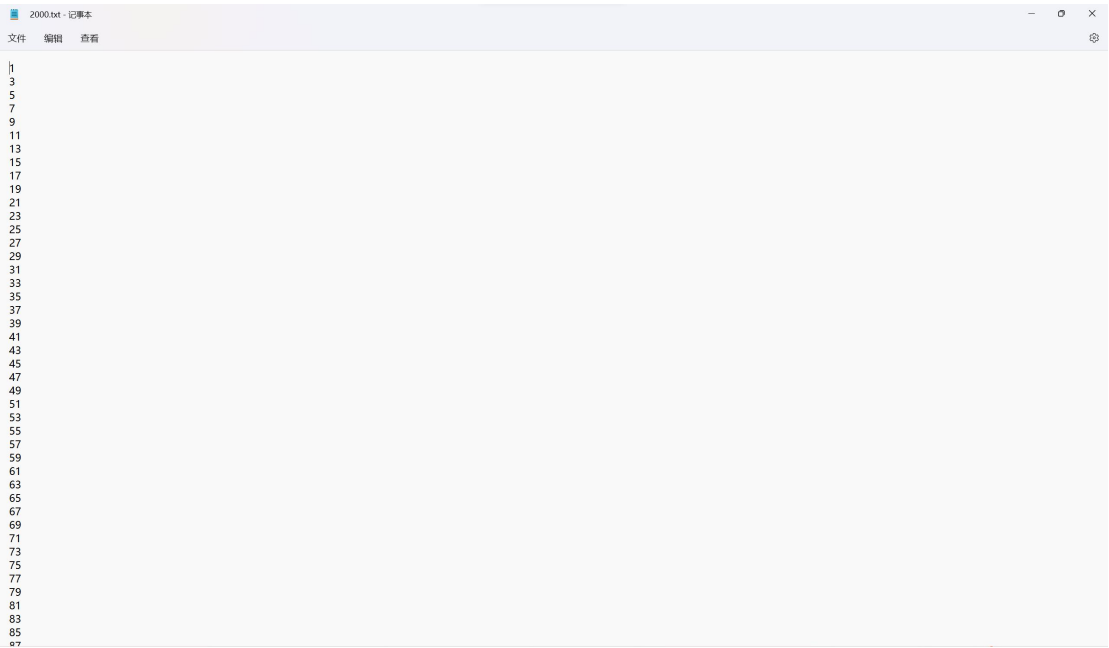
实验前测试:

测试输入:

data/2000.txt

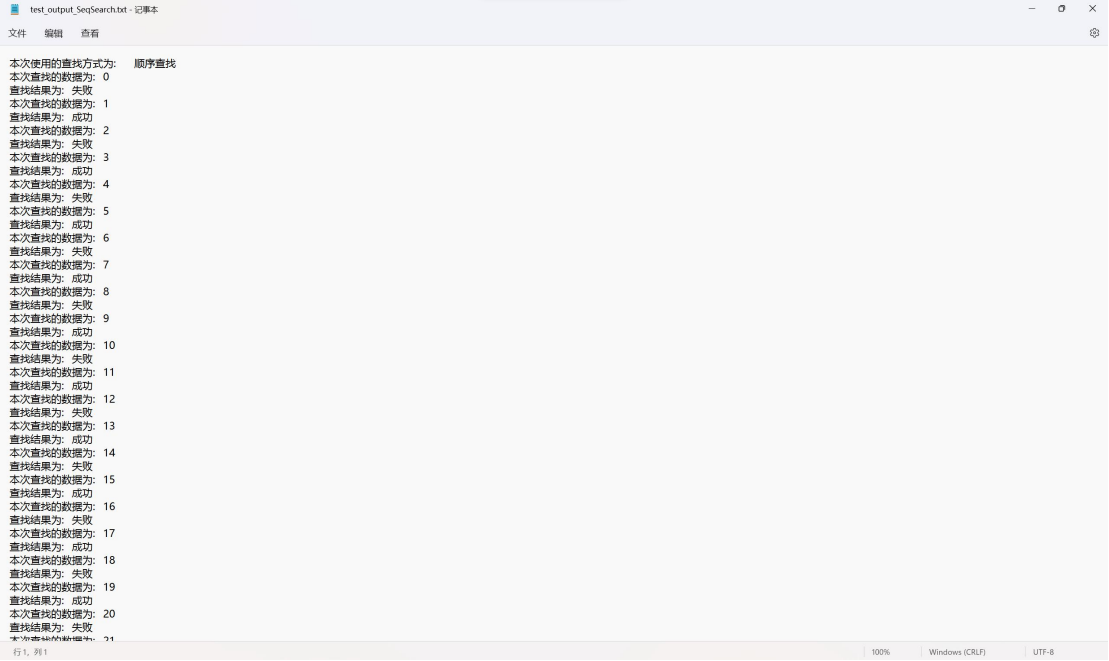
文件中为 1-2000 中的所有奇数

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:



实际输出:

顺序查找时: test_output_SeqSearch.txt

test_output_SeqSearch.txt - 记事本

文件 编辑 查看

本次使用的查找方式: 顺序查找
本次查找的数据为: 0
查找结果为: 失败
本次查找的数据为: 1
查找结果为: 成功
本次查找的数据为: 2
查找结果为: 失败
本次查找的数据为: 3
查找结果为: 成功
本次查找的数据为: 4
查找结果为: 失败
本次查找的数据为: 5
查找结果为: 成功
本次查找的数据为: 6
查找结果为: 失败
本次查找的数据为: 7
查找结果为: 成功
本次查找的数据为: 8
查找结果为: 失败
本次查找的数据为: 9
查找结果为: 成功
本次查找的数据为: 10
查找结果为: 失败
本次查找的数据为: 11
查找结果为: 成功
本次查找的数据为: 12
查找结果为: 失败
本次查找的数据为: 13
查找结果为: 成功
本次查找的数据为: 14
查找结果为: 失败
本次查找的数据为: 15
查找结果为: 成功
本次查找的数据为: 16
查找结果为: 失败
本次查找的数据为: 17
查找结果为: 成功
本次查找的数据为: 18
查找结果为: 失败
本次查找的数据为: 19
查找结果为: 成功
本次查找的数据为: 20
查找结果为: 失败

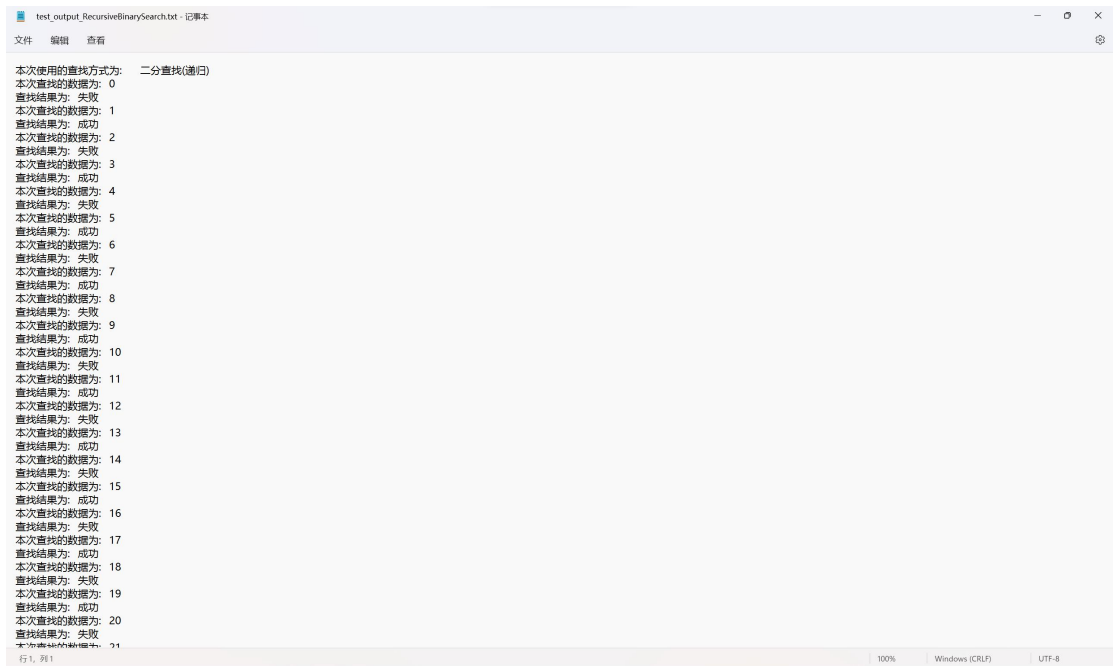
二分查找(非递归): test_output_BinarySearch.txt

test_output_BinarySearch.txt - 记事本

文件 编辑 查看

本次使用的查找方式: 二分查找(非递归)
本次查找的数据为: 0
查找结果为: 失败
本次查找的数据为: 1
查找结果为: 成功
本次查找的数据为: 2
查找结果为: 失败
本次查找的数据为: 3
查找结果为: 成功
本次查找的数据为: 4
查找结果为: 失败
本次查找的数据为: 5
查找结果为: 成功
本次查找的数据为: 6
查找结果为: 失败
本次查找的数据为: 7
查找结果为: 成功
本次查找的数据为: 8
查找结果为: 失败
本次查找的数据为: 9
查找结果为: 成功
本次查找的数据为: 10
查找结果为: 失败
本次查找的数据为: 11
查找结果为: 成功
本次查找的数据为: 12
查找结果为: 失败
本次查找的数据为: 13
查找结果为: 成功
本次查找的数据为: 14
查找结果为: 失败
本次查找的数据为: 15
查找结果为: 成功
本次查找的数据为: 16
查找结果为: 失败
本次查找的数据为: 17
查找结果为: 成功
本次查找的数据为: 18
查找结果为: 失败
本次查找的数据为: 19
查找结果为: 成功
本次查找的数据为: 20
查找结果为: 失败

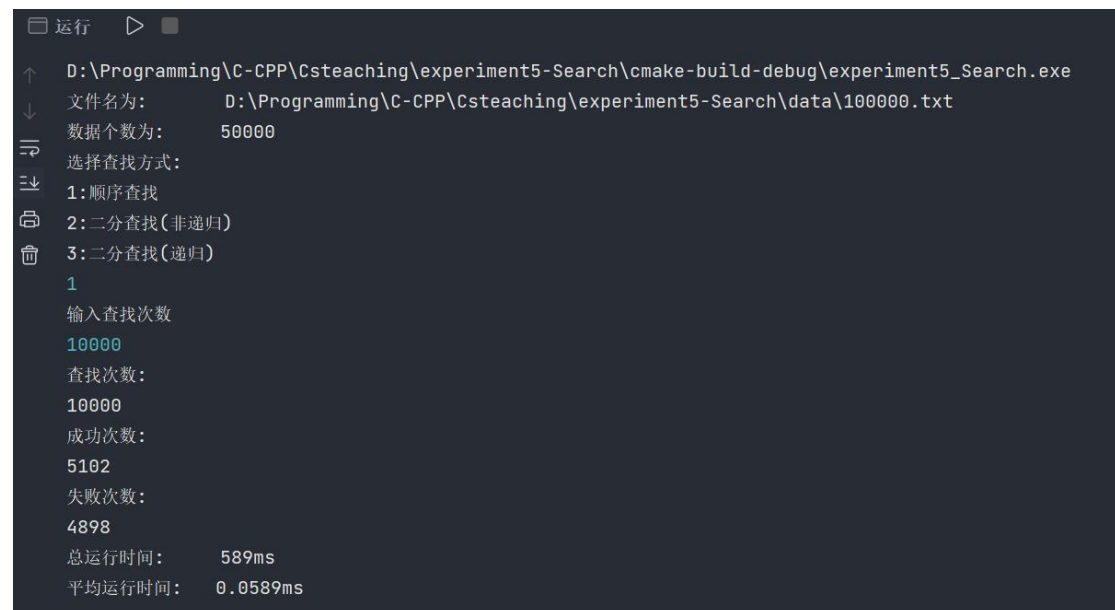
二分查找(递归): test_output_RecursiveBinarySearch.txt



测试结论:正确

实验时:

测试输出:



```
运行
D:\Programming\C-CPP\Csteaching\experiment5-Search\cmake-build-debug\experiment5_Search.exe
文件名为: D:\Programming\C-CPP\Csteaching\experiment5-Search\data\100000.txt
数据个数为: 50000
选择查找方式:
1: 顺序查找
2: 二分查找(非递归)
3: 二分查找(递归)
2
输入查找次数
10000
查找次数:
10000
成功次数:
4922
失败次数:
5078
总运行时间: 248ms
平均运行时间: 0.0248ms
```

```
运行
D:\Programming\C-CPP\Csteaching\experiment5-Search\cmake-build-debug\experiment5_Search.exe
文件名为: D:\Programming\C-CPP\Csteaching\experiment5-Search\data\100000.txt
数据个数为: 50000
选择查找方式:
1: 顺序查找
2: 二分查找(非递归)
3: 二分查找(递归)
3
输入查找次数
10000
查找次数:
10000
成功次数:
5056
失败次数:
4944
总运行时间: 218ms
平均运行时间: 0.0218ms
```

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)					平均比较时间
		测试 1	测试 2	测试 3	测试 4	测试 5	
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 $T(n) = O(n)$,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 $T(n) = O(n)$ 。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 $T(n) = O(\log n)$,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

BeHappy
Legion R9000K2021H

重命名这台电脑

① 设备规格

复制

^

设备名称	BeHappy	
处理器	AMD Ryzen 7 5800H with Radeon Graphics	3.20 GHz
机带 RAM	16.0 GB (15.9 GB 可用)	
设备 ID	F535E77C-5B08-4427-BCA4-7175E81D4149	
产品 ID	00342-36298-13256-AAOEM	
系统类型	64 位操作系统, 基于 x64 的处理器	
笔和触控	没有可用于此显示器的笔或触控输入	

相关链接 域或工作组 系统保护 高级系统设置

Windows 规格

复制

^

版本	Windows 11 家庭中文版
版本	22H2
安装日期	2022/5/13
操作系统版本	22623.891
序列号	PF35G0RP
体验	Windows Feature Experience Pack 1000.22637.1000.0
Microsoft 服务协议	
Microsoft 软件许可条款	

② 支持

复制

^

制造商	Lenovo
网站	联机支持

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_out
put_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;
}

```