桂林电子科技大学2022-2023学年第2学期

算法设计与分析实验报告 课号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 用分治法实现元素的归并与检索 | | | | | | | |  | 辅导教师意见：  成绩 教师签名： |
| 院系 | 计算机与信息安全学院 | | | 专业 | | 计算机科学与技术 | | |
| 学号 |  | | | 姓名 | |  | | |
| 实验日期 | 2023 | 年 | 5 | | 月 | | 12 | 日 |
|  |  | | | | | | | |

实验一、用分治法实现元素的归并与检索

### 实验目的

（1）掌握分治策略求解问题的基本原理、基本设计流程；

（2）掌握分治算法的时间复杂度分析方法；

（3）掌握归并排序、二分检索算法。

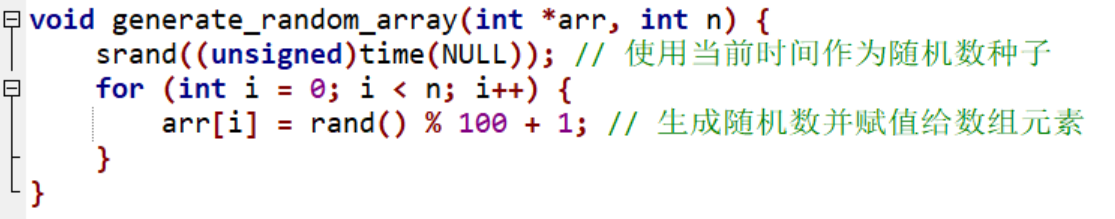
### 实验内容

随机生成20个从1－100之间的随机数，用归并排序和二分检索实现元素的查找算法。

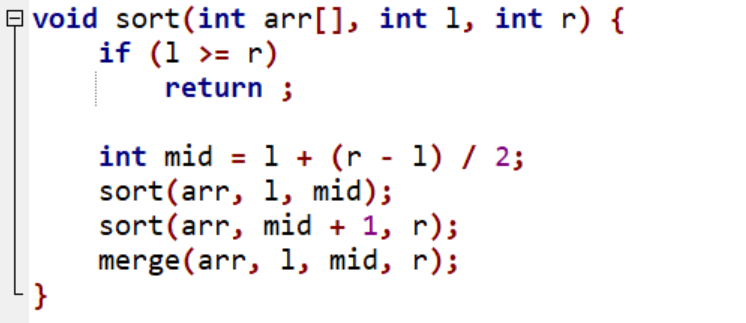
### 分析

思路：

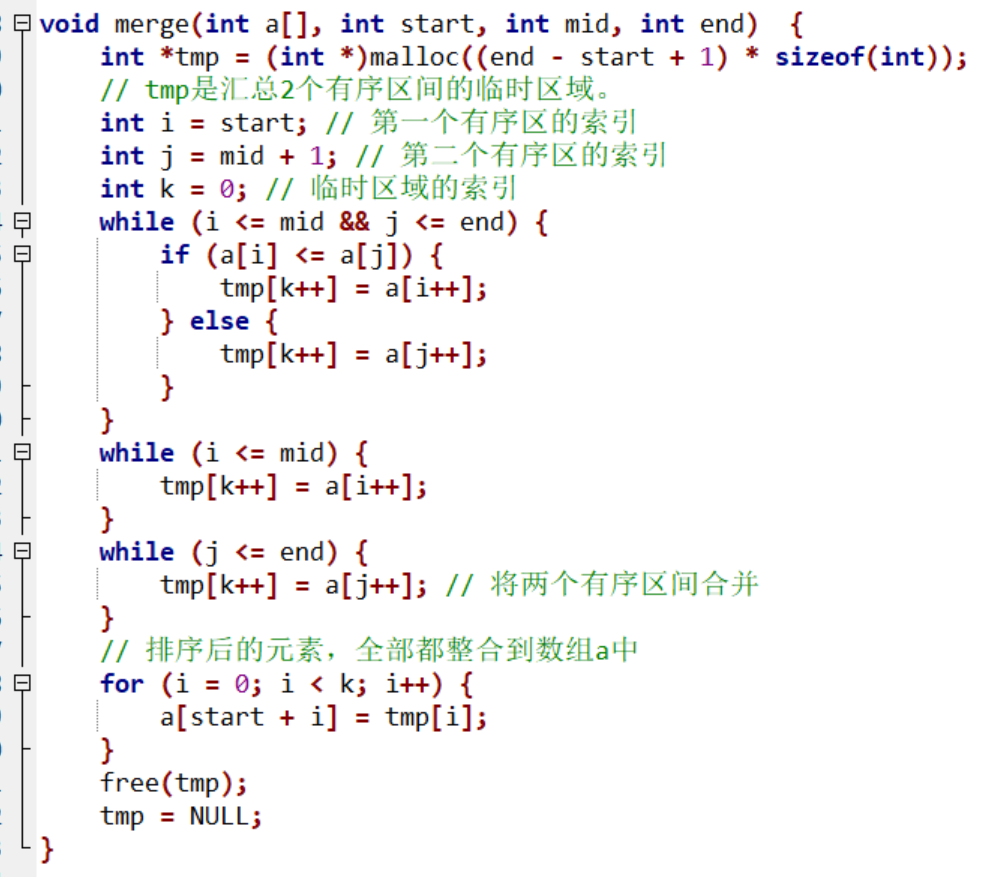
(1) 首先生成一个由20个取值范围1~100的随机数组成的数组arr，相应的函数签名：void generate\_random\_array(int a[], int n);



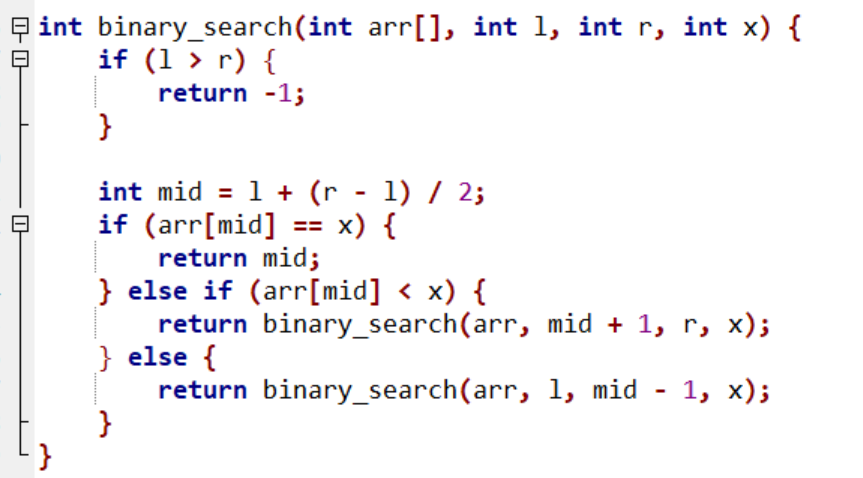
(2) 然后使用归并排序对arr数组进行排序，相应的函数签名：void sort(int arr[], int l, int r), 这里sort的时间复杂度是O(nlogn),



和void merge(int a[], int start, int mid, int end),merge的时间复杂度是O(n);



(3) 最后使用二分检索算法查询指定的元素x，这里的x使用随机数生成，范围是1-100；二分检索算法的函数签名是int binary\_search(int arr[], int l, int r, int x)，其中int mid = l+(r-l)/2 是为了避免整数达到亿这个级别时产生溢出的bug。作为最快的查找算法，它的时间复杂度是O(logn)。



### 程序

代码：

/\*

本关任务：随机生成20个从1－100之间的随机数，用递归与分治法编程实现元素的查找算法。

\*/

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

void generate\_random\_array(int a[], int n);

void sort(int arr[], int l, int r);

void merge(int a[], int start, int mid, int end) ;

int binary\_search(int arr[], int l, int r, int x);

int main( void ) {

int N = 20;

int arr[N];

generate\_random\_array(arr, N);

for (int i = 0; i < N; i++) {

printf("%d ", arr[i]);

}

printf("\n\n");

sort(arr, 0, N - 1);

for (int i = 0; i < N; i++) {

printf("%d ", arr[i]);

}

srand((unsigned)time(NULL));

int x = rand() % 100 + 1;

printf("\n\n要查找的值%d:\n", x);

printf("\n结果：%d", binary\_search(arr, 0, N - 1, x));

return 0;

}

void generate\_random\_array(int \*arr, int n) {

srand((unsigned)time(NULL)); // 使用当前时间作为随机数种子

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

arr[i] = rand() % 100 + 1; // 生成随机数并赋值给数组元素

}

}

void sort(int arr[], int l, int r) {

if (l >= r)

return ;

int mid = l + (r - l) / 2;

sort(arr, l, mid);

sort(arr, mid + 1, r);

merge(arr, l, mid, r);

}

void merge(int a[], int start, int mid, int end) {

int \*tmp = (int \*)malloc((end - start + 1) \* sizeof(int));

// tmp是汇总2个有序区间的临时区域。

int i = start; // 第一个有序区的索引

int j = mid + 1; // 第二个有序区的索引

int k = 0; // 临时区域的索引

while (i <= mid && j <= end) {

if (a[i] <= a[j]) {

tmp[k++] = a[i++];

} else {

tmp[k++] = a[j++];

}

}

while (i <= mid) {

tmp[k++] = a[i++];

}

while (j <= end) {

tmp[k++] = a[j++]; // 将两个有序区间合并

}

// 排序后的元素，全部都整合到数组a中

for (i = 0; i < k; i++) {

a[start + i] = tmp[i];

}

free(tmp);

tmp = NULL;

}

int binary\_search(int arr[], int l, int r, int x) {

if (l > r) {

return -1;

}

int mid = l + (r - l) / 2;

if (arr[mid] == x) {

return mid;

} else if (arr[mid] < x) {

return binary\_search(arr, mid + 1, r, x);

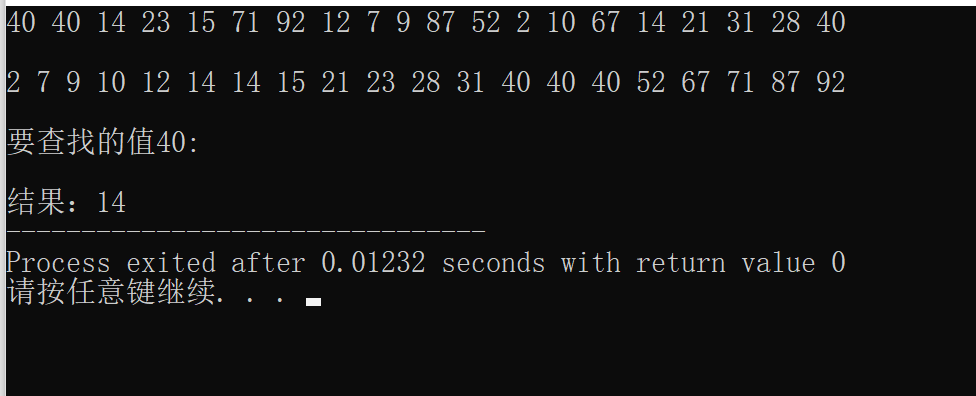
} else {

return binary\_search(arr, l, mid - 1, x);

}

}

运行截图:



### 实验体会

分治算法的基本思想是：

当我们求解某些问题时，由于这些问题要处理的数据相当多，或求解过程相当复杂，使得直接求解法在时间上相当长，或者根本无法直接求出。对于这类问题，我们往往先把它分解成几个子问题，找到求出这几个子问题的解法后，再找到合适的方法，把它们组合成求整个问题的解法。如果这些子问题还较大，难以解决，可以再把它们分成几个更小的子问题，以此类推，直至可以直接求出解为止。

在这个过程中合并是关键，而能否成功合并又和怎么分是息息相关的。分得好才能合，分的时候要以能否合并为主要参考条件。另外，分出来的子问题一般都是与原问题性质相同的小规模问题，这就使得递归算法派上了用场。递归和分治的联系很紧密，在程序设计过程中它们经常一起出现。