实验一实验报告

2112060-孙蕗

1. **题目**

对数列X={12，61，5，30，12，4，12，60，61，30，12，5，61}，使用快速排序分别对奇数和偶数进行非降序排序，在不使用新数组的前提下，奇数全部置于数组头部，偶数全部置于数组尾部。

例如数组X的排序后结果为{5，5，61，61，61，4，12，12，12，12，30，30，60}

1. **代码及简单的算法注释**

#include <iostream>

using namespace std;

void Quick\_Sort(int\* arr, int begin, int end) {

if (begin > end)

return;

int tmp = arr[begin];

int i = begin;

int j = end;

while (i != j) {

while (arr[j] >= tmp && j > i)

j--;

while (arr[i] <= tmp && j > i)

i++;

if (j > i) {

int t = arr[i];

arr[i] = arr[j];

arr[j] = t;

}

}

arr[begin] = arr[i];

arr[i] = tmp;

Quick\_Sort(arr, begin, i - 1);

Quick\_Sort(arr, i + 1, end);

}

int main()

{

int array[13] = { 12,61,5,30,12,4,12,60,61,30,12,5,61 };

int odd = 0;

int tmp1, tmp2, tmp;

for (int i = 0; i <= 12; i++)

{

cout << array[i] << " ";

}

cout << endl;

//确定奇数个数

for (int i = 0; i < 13; i++)

{

if (array[i] % 2 != 0)

{

odd++;

}

}

//奇偶分开

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

{

tmp1 = i;

if (array[tmp1] % 2 == 0)

{

for (int j = 12; j >= odd - 1; j--)

{

tmp2 = j;

if (array[tmp2] % 2 == 1)

{

tmp = array[tmp1];

array[tmp1] = array[tmp2];

array[tmp2] = tmp;

break;

}

}

}

}

//快速排序

Quick\_Sort(array, 0, odd-1);

Quick\_Sort(array, odd, 12);

for (int i = 0; i <= 12; i++)

{

cout << array[i] << " ";

}

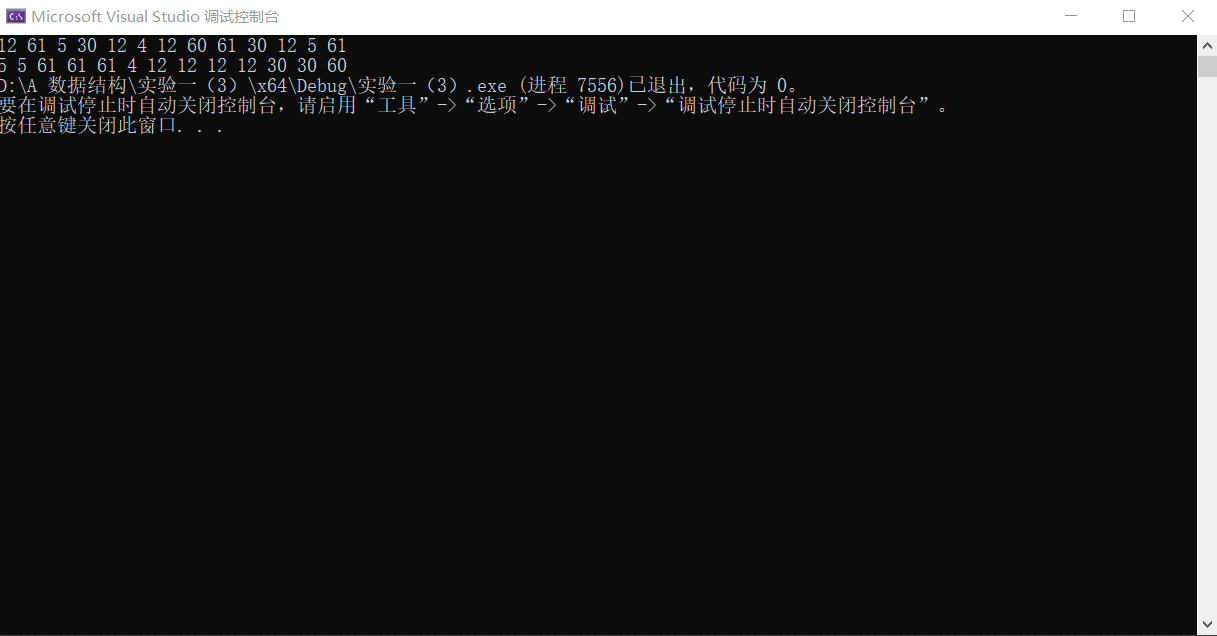
return 0;

}

1. **复杂度分析**
2. 声明一个数组，时间复杂度O（1）,空间复杂度为O(n)
3. 确定数组中奇数个数，for循环遍历，时间复杂度O（n），空间复杂度O（1）
4. 奇数偶数分开，for循环嵌套，时间复杂度O（n2）,空间复杂度O（1）
5. 奇数偶数分别快速排序，时间复杂度0（nlog n）,空间复杂度O（log n）
6. 两次for循环输出数组，时间复杂度O（n），空间复杂度O（1）

所以，时间复杂度为O(n2),空间复杂度为O（n）;

1. **运行截图**



1. **心得总结**

编写代码的时候，使用合适的[数据结构](https://so.csdn.net/so/search?q=%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/Charlesssyang/article/details/_blank)和算法，在处理体量非常庞大的数据的时候，可以极大地提高计算效率，引入时间复杂度和空间复杂度作为衡量标准用来衡量代码的效率。时间复杂度主要衡量一个算法的运行快慢，空间复杂度主要衡量一个算法运行所需要的额外空间。分析复杂度，在特定的场景下选用合适的正确的算法。

通过第一次实验初步学习到空间复杂度时间复杂度的计算方法，认识到程序代码不仅只是要追求解决问题，程序也有尽可能降低空间复杂度和时间复杂度的需求，初步体会到空间复杂度与时间复杂度如何平衡。