## 《算法分析理论及应用》课程实验报告

**班级：软工182 姓名： 邓棋 学号：2018081062**

**一、实验题目**

1. 分析代码1给出的顺序查找算法的最好、最坏和平均复杂度

2. 根据算法代码2，分析其空间复杂度，其中临时空间为变量i、maxi占用的空间。

3. 分析代码3的递归算法，分析调用maxelem(a，0，n-1)的空间复杂度。

4. 阅读代码4，填写缺少的语句, 分析冒泡排序的时间复杂度,并给出运行结果。

**二、实验内容**

**1、顺序查找时间复杂度**

**最好时间复杂度：**数组的第一个数a[0]就是要查找的k值，此时的时间复杂度为O(1)

**最坏时间复杂度：**数组的最后一个数a[n-1]为要查找查找的k值，此时的时间复杂度为O(n)。

**平均时间复杂度：**其平均数为（n+1）/ 2,即为O(n)。

**2、计算空间复杂度**

只有i和maxi临时占用了存储空间，空间复杂度为O(1)。

**3、递归函数计算空间复杂度**

当n为1时，程序只运行一次，没有递归，此时的空间复杂度为O(1)；当n>1时，进行了递归操作，所占用的空间应该表示为f(n)=2\*f(n/2)+O(1)，此时的空间复杂度为O(n)。

**4、冒泡排序**

**①.** **填写缺少的语句**

（1）while (exchange != 0)

（2）int temp = r[j];

r[j] = r[j + 1];

r[j + 1] = temp;

（3）exchange = j + 1;

**②．时间复杂度**

BubbleSort()函数实现的冒泡排序的时间复杂度主要依赖于内外循环的时间开销,最优的情况就是开始已经排好序了，后续循环不需要进行交换，则时间开销为：[n(n-1)]/2，最优情况的时间复杂度为O(n^2);最差的情况就是开始时元素全是逆序的，每一次排序都要交换两个元素，时间花销为：[3n(n-1)]/2，最差情况的时间复杂度为O(n^2)。

综上，冒泡排序的平均时间复杂度为O(n^2)。

**③．运行结果**

排序前:4 3 9 2 6 1

排序后:1 2 3 4 6 9

**三、实验目的**

1. 培养学生分析算法复杂度的能力并且能够描述算法复杂性的一些基本理论。

2. 通过教学案例培养学生的逻辑思维能力，能够分析算法复杂性，并且能够计算算法的时空复杂性。

**四、实验代码**

**1、顺序查找**

int seqSearch(int a[ ], int n, int k)

{ int i=0;

while (i<n && a[i]!=k)

{

i++;

}

if (i<n) return i;

else return -1;

}

**2、计算空间复杂度**

int max(int a[]，int n)

{

int i，maxi=0;

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

if (a[i]>a[maxi])

maxi=i;

return a[maxi];

}

**3、递归函数**

int maxelem(int a[],int i,int j)

{

int mid=(i+j)/2,max1,max2;

if (i<j)

{

max1=maxelem(a,i,mid);

max2=maxelem(a,mid+1,j);

return (max1>max2)?max1:max2;

}

else return a[i];

}

**4、冒泡排序**

#include <iostream.h>

void BubbleSort(int r[ ], int n);

int main()

{

int i, r[]={4,3,9,2,6,1};

cout<<"排序前：";

for(i=0;i<6;i++)

cout<<r[i]<<" ";

cout<<endl;

BubbleSort(r,6);

cout<<"排序后：";

for(i=0;i<6;i++)

cout<<r[i]<<" ";

cout<<endl;

return 0;

}

void BubbleSort(int r[ ], int n)

{

int bound, exchange = n - 1; //第一趟起泡排序的区间是[0,-1]

（1） //当上一趟排序有记录交换时

{

bound = exchange; exchange = 0;

for (int j = 0; j < bound; j++) //一趟起泡排序区间是[0, bound]

if (r[j] > r[j + 1]) {

（2） ;

//交换记录

（3） //记载每一次记录交换的位置

}

}

}

**五、实验总结**

书写以下实验总结：1、算法伪代码编写；2、算法设计策略描述；3、算法时空复杂度分析。4、遇到的问题及解决方法。

1. **冒泡排序：(假设排序成顺序排列)**

for (int i = 0; i < n – 1; i++)

{

for (int j = 0; j < n – i – 1; j ++)

{

if (a[j] > a[j+1])

{

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

1. **冒泡排序（顺序排列）的策略描述：**

假设有n个不同大小的数随机排列，只需要n-1次排列，就能将所有的数都换个位置，这也是冒泡排序外层循环所用的思想，内层循环则是更注重于细节，实际地比较两数的大小，将每一次能遍历到的最大的数挪到波及范围的最后。

具体地讲，外层的第一次循环，内层循环使n个数据都两两比较了一次，并将此数列中的数交换位置，使得最大的数不断后移，使它变成最后一个数；外层的第二次循环，内层循环就只需要考虑n-1个数了，因为此时数列最末尾的数已经是最大的数了，不需要再纳入考虑范围，再将第二大的数挪到倒数第二个位置……外层循环每循环完一次就会浮出一个最大数，依次往后排，同样的，小数也不断浮到数列前面，循环完毕后，数列也就有了顺序。

1. **冒泡排序的时空复杂度分析：**

**时间复杂度：**冒泡排序的时间复杂度主要依赖于内外循环的时间开销,最优的情况就是开始已经排好序了，后续循环不需要进行交换，只有外层循环纳入时间开销，则时间开销为：[n(n-1)]/2，最优情况的时间复杂度为O(n^2);最差的情况就是开始时元素全是逆序的，每一次排序都要交换两个元素，时间花销为：[3n(n-1)]/2，最差情况的时间复杂度为O(n^2)。综上，冒泡排序的平均时间复杂度为O(n^2)。

**空间复杂度：**冒泡排序的空间复杂度全在于交换元素时临时变量（temp）作占的内存空间，同样的最优情况仍是不需要交换位置，临时变量不会分配内存，最优情况空间复杂度为0；最差的情况仍是元素的初态为逆序的，每次内层循环都会调用if语句块，一共有[n(n-1)]/2个临时变量生成，但是临时变量的生命周期很短，只在if语句块内，语句完毕后给它分配的内存会被释放，这样来看，n个临时变量都是反复地占用了同一块内存，此时的空间复杂度为O(1)。综上，冒泡排序的平均空间复杂度为O(1)。

**六、算法策略的英文描述（字数>200）**

Inner loop and outer loop are the main components of bubble sort. Suppose there are n random numbers of different sizes, you can change all the numbers' position in only n-1 times swaps. This is the idea of ​​outer loop of bubble sort. Inner loop pays more attention to detail, comparing the size of the two numbers actually, and moves the maximum number to the end of range where can be involved each time.

Specifically, the inner loop loops the n data one by one and swaps the numbers in the sequence to make the largest number move backwardly until making it become the last one when the first loop of the outer loop. Then, when the second loop of the outer loop, the inner loop only needs to consider n-1 numbers. Because the largest number of the sequence already in the end, no need to be considered again. And then the second largest number will be moved to the penultimate position......

The outer loop will float a maximum number after each loop, and then put them in the back row. Similarly, the small number will be floated to the front of this sequence. After the loop is completed, the data in the sequence has been sorted .