作业 1 算法实现与复杂度

班级：22WL022　　姓名：杨明达　　学号：2022110829

一、 你能结合自己的编程实践，谈一下数据结构和算法之间的关系吗？

数据结构是组织和存储数据的方式，算法是解决问题的方法。数据结构是算法的静态本质，用编程语言来说就是语法。算法是数据结构的动态特征，用编程语言来表示就是语义。用编程语言来描述数据结构和算法，就是程序。数据结构和算法相辅相成，选择合适的数据结构可以简化算法的设计和实现，高效的算法能充分发挥数据结构的优势。

在编程中，以插入排序这个常见算法为例，插入排序，依赖于线性表结构，即从一个数据项出发可以取得它的前后数据项。从编程实践的角度，线性表可以实现为数组或链表，算法相对于数据结构有一定的独立性，算法可以对应不同的数据结构实现。

二、简单排序算法（具体代码请见 Simple_sorting_algorithm.c）

1、 编程实现（直接） 冒泡排序算法、（直接）选择排序算法和（直接）插入排序算法，分析其时间和空间复杂度。

（1）无递归的冒泡排序

```c
// 无递归的冒泡排序
void bubbleSort(int array[], int len) {
    int temp, i, j;
    for (i = 0; i < len - 1; i++) {
        for (j = 0; j < len - 1 - i; j++) {
            if (array[j] > array[j + 1]) {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

图 1 无递归的冒泡排序算法

时间复杂度：在算法中共有两层循环，第一层循环执行 $n-1$ 次，且在第一层循环执行一次时，第二层循环执行 $n-1-i$ 次，在内层循环中，至少执行一步，故算法的时间复杂度为：$T(n)=\sum_{i=0}^{n-2}\sum_{j=0}^{n-2-i}1\le O\left(\frac{n(n-1)}{2}\right)=O(n^2)$。

空间复杂度：由于冒泡排序需要用一个空间来进行 temp 数字交换，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

（2）无递归的选择排序

```
33  // 无递归的选择排序
34  void selectionSort(int array[], int len) {
35      int k, temp, i, j;
36      for (i = 0; i < len - 1; i++) {
37          k = i;
38          for (j = i + 1; j < len; j++) {
39              if (array[j] < array[k]) {
40                  k = j;
41              }
42          }
43          if (k != i) {
44              temp = array[k];
45              array[k] = array[i];
46              array[i] = temp;
47          }
48      }
49  }
```

图 2 无递归的选择排序算法

时间复杂度：在算法中共有两层循环，第一层循环执行 $n-1$ 次，且在第一层循环执行一次时，第二层循环执行 $n-1-i$ 次，在内层循环中，至少执行一步，故算法的时间复杂度为：$T(n)=\sum_{i=0}^{n-2}\sum_{j=0}^{n-2-i}1\le O\left(\frac{n(n-1)}{2}\right)=O(n^2)$。

空间复杂度：由于选择排序需要用一个空间来进行 temp 数字交换，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

（3）无递归的插入排序

```
70  // 无递归的插入排序
71  void insertSort(int array[], int len) {
72      int key, i, j;
73      for (i = 1; i < len; i++) {
74          key = array[i];
75          for (j = i - 1; j >= 0; j--) {
76              if (array[j] > key) {
77                  array[j + 1] = array[j];
78              } else {
79                  break;
80              }
81              array[j] = key;
82          }
83      }
84  }
```

图 3 无递归的插入排序算法

时间复杂度：在算法中共有两层循环，第一层循环执行 $n-1$ 次，且在第一层循环执行一次时，第二层循环执行 i 次，在内层循环中，至少执行一步，故算法的时间复杂度为：$T(n)=\sum_{i=0}^{n-2}\sum_{j=0}^{i}1=O\left(\frac{n(n-1)}{2}\right)=O(n^2)$。

空间复杂度：由于插入排序需要用一个空间来进行 $array[i]$ 的暂时存储，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

2、编程实现（直接）冒泡排序、（直接）选择排序和（直接）插入排序的递归算法，分析其时间和空间复杂度。

（1）有递归的冒泡排序

```
// 有递归的冒泡排序
void recursion_bubbleSort(int array[], int len) {
    int temp, i;
    for (i = 0; i < len - 1; i++) {
        if (array[i] > array[i + 1]) {
            temp = array[i];
            array[i] = array[i + 1];
            array[i + 1] = temp;
        }
    }
    len--;
    if (len != 0) {
        return recursion_bubbleSort(array, len);
    }
}
```

图 4 有递归的冒泡排序算法

时间复杂度：在算法中要递归 n 次，每次递归有 $n-1$ 次循环，该算法的函数用时

$$T(n)=\begin{cases} T(n-1)+(n-1) & n>1 \\ 0 & n=1 \end{cases},$$

$T(n)=\dfrac{n(n-1)}{2}$，$O(T(n))=O(n^2)$，故时间复杂度为 $O(n^2)$。

空间复杂度：由于有递归的冒泡排序需要用一个空间来进行 temp 数字交换，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

（2）有递归的选择排序

```
// 有递归的选择排序
void recursion_selectionSort(int array[], int len) {
    int k, i, j, temp;
    j = len - 1;
    k = len - 1;
    for (i = len - 2; i >= 0; i--) {
        if (array[i] > array[k]) {
            k = i;
        }
    }
    if (j != k) {
        temp = array[k];
        array[k] = array[j];
        array[j] = temp;
    }
    len--;
    if (len != 1) {
        return recursion_selectionSort(array, len);
    }
}
```

图 5 有递归的选择排序算法

时间复杂度：在算法中要递归 n 次，每次递归有 $n-1$ 次循环，该算法的函数用时

$$T(n)=\begin{cases}T(n-1)+(n-1) & n>1\\ 0 & n=1\end{cases},$$

$T(n)=\dfrac{n(n-1)}{2}$，$O(T(n))=O(n^2)$，故时间复杂度为 $O(n^2)$。

空间复杂度：由于有递归的选择排序需要用一个空间来进行 temp 数字交换，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

（3）有递归的插入排序

```
// 有递归的插入排序
void recursion_insertSort(int array[], int len, int num) {
    int key;
    key = array[num];
    for (int i = num - 1; i >= 0; i--) {
        if (array[i] > key) {
            array[i + 1] = array[i];
        } else {
            break;
        }
        array[i] = key;
    }
    num++;
    if (num != len) {
        return recursion_insertSort(array, len, num);
    }
}
```

图 6 有递归的插入排序算法

时间复杂度：在算法中要递归 n 次，每次递归有 $n-1$ 次循环，该算法的函数用时

$$T(n)=\begin{cases}T(n-1)+(n-1) & n>1\\ 0 & n=1\end{cases},$$

$T(n)=\dfrac{n(n-1)}{2}$，$O(T(n))=O(n^2)$，故时间复杂度为 $O(n^2)$。

空间复杂度：由于有递归的插入排序需要用一个空间来进行 $array[num]$ 的暂时存储，并且该变量在循环外声明，与 n 无关，故空间复杂度为：$O(1)$。

3、你能用实验的方法验证上述算法时间复杂度的分析结论吗？

利用 clock_t 函数，可以测出上述 6 种排序算法在 $n=30000$ 时所用的时间，输出结果如下：

```
The time of bubbleSort is 1.966000s
The time of selectionSort is 0.964000s
The time of insertSort is 0.524000s
The time of recursion_bubbleSort is 1.914000s
The time of recursion selectionSort is 0.847000s
The time of recursion_insertSort is 0.549000s
```

图 7 六种排序算法的运行时间

三、 汉诺塔问题（具体代码请见 Hanoi_tower_problem.c）

1、 编程实现汉诺塔问题的递归算法，并分析的时间和空间复杂度。

```
6   // 汉诺塔递归
7   void hannuota(int num, char begin, char middle, char end) {
8       if (num == 1) {
9           printf("%c-->%c\n", begin, end);
10          key++;
11      } else {
12          hannuota(num - 1, begin, end, middle);
13          hannuota(1, begin, middle, end);
14          hannuota(num - 1, middle, begin, end);
15      }
16  }
```

图 8 汉诺塔问题的递归算法

时间复杂度：在算法中，在 $num=n$ 时，一次递归要进行下一级的三个递归，其两个递归为 $num=n-1$，一个递归为 $num=1$，该算法的函数用时

$$T(n)=\begin{cases}2*T(n-1)+1 & n>1\\ 1 & n=1\end{cases}$$

故$T(n)=2^n-1$，$O(T(n))=O(n^2)$，故时间复杂度为$O(n^2)$。

空间复杂度：递归调用会在内存中创建函数调用的堆栈，每一层递归都需要一定的空间，总共开辟 n 块空间，空间复杂度为$O(n)$。

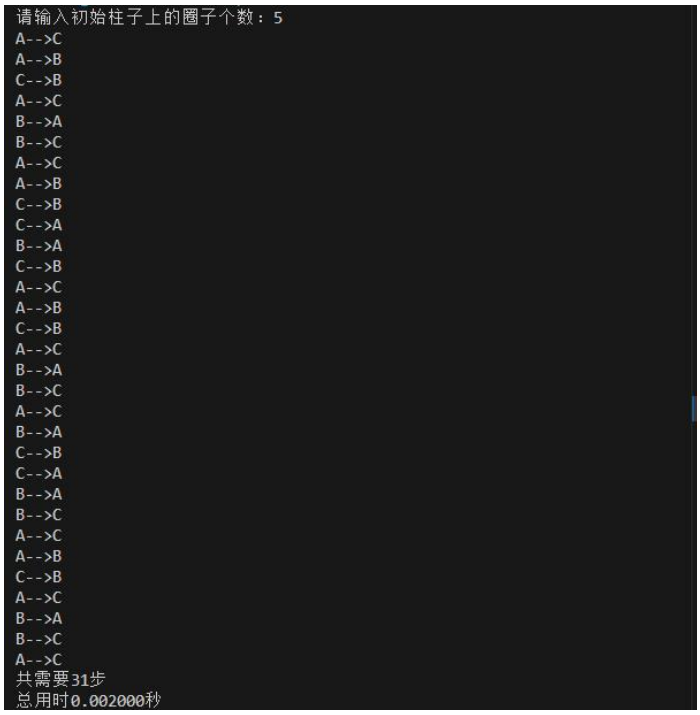

图 9 汉诺塔问题的递归算法的测试结果

四、角谷猜想(冰雹猜想)(具体代码见 Kakutani_conjecture.c)

对于任何正整数 n，若是偶数，则除以 2；若是奇数，则乘以 3 加上 1。这样就得到一个新的整数，若继续进行上述处理，则最后得到的数一定是 1 吗？

图 10 验证角谷猜想的正确性

经过编写程序，最后得到的数一定是 1。

1、编写程序对 100 以内的整数进行测试验证，并找出其中最长的序列。

图 11 查找 100 以内最长序列

通过编写程序发现对 100 以内的整数测试，初始数为 97 的序列最长，共进行 118 次操作。