《数据结构与算法》作业一

姓名：余弦 学号：2021112893 班级：2103401

一、数据结构与算法的关系

大部分算法的实现需要利用数据结构。比如深度优先搜索就经常要用到栈、宽度优先搜索就需要用到队列。而数据结构则是对数据的一种抽象，用于更高效率的解决问题。数据结构和算法一起构成了可以高效执行程序。

二、简单的排序算法

1.冒泡排序:

代码：

```
void bubble_sort(int num[N], int len){
    for(int i = 0 ; i < len - 1 ; i ++){
        for(int j = i + 1 ; j < len ; j++){
            if(num[i] < num[j]) swap(num[i], num[j]);
        }
    }
}
```

时间复杂度分析：双重循环，且每层循环次数都与n相关，故时间复杂度为$O(n^2)$

空间复杂度分析：在进行swap时需要一个tmp变量，所以其空间复杂度为$O(1)$

2. 选择排序

代码：

```
void select_sort(int num[N], int len){
    for(int i = 0 ; i < len - 1 ; i++){
        int min = i;
        for(int j = i + 1; j < len ; j++){
            if(num[j] < num[min]){
                min = j;
            }
        }
        swap(num[i], num[min]);
    }
}
```

时间复杂度分析：双重循环，且每层循环都与n相关，故时间复杂度为$O(n^2)$

空间复杂度分析：在进行swap时需要一个tmp变量，所以其空间复杂度为$O(1)$

3.插入排序

代码：

```
void insert_sort(int num[N], int len){
    for(int i = 1; i < len; i++){
        int val = num[i];
        int j;
        for(j = i - 1; j >= 0 && val < num[j]; j--){
            num[j + 1] = num[j];
        }
        num[j + 1] = val;
    }
}
```

时间复杂度分析：双重循环，且都与n相关，所以时间复杂度为$O(n)$

空间复杂度分析：只需一个val来临时储存，所以空间复杂度为$O(1)$

4.冒泡排序（递归）

代码：

```
void bubble_sort_recursion(int num[N], int len){
    if(len == 1) return;
    for(int i = 0 ; i < len - 1 ; i ++){
        if(num[i] > num[i + 1]) swap(num[i], num[i + 1]);
    }
    bubble_sort_recursion(num, len - 1);//相当于非递归实现的外层循环，只不过是从后往前进行循环
}
```

时间复杂度分析：本质上递归的实现只是取代了原冒泡排序中外层循环的作用，所以也可以当做双重循环，时间复杂度为$O(n^2)$

空间复杂度分析：只在swap使用额外的tmp变量，所以空间复杂度为$O(1)$

5.选择排序（递归）

代码：

```
void select_sort_recursion(int num[N], int len){
    if(len == 1) return;
    int max = 0;
    for(int i = 1 ; i < len ; i++){
        if(num[i] > num[max]) max = i;
    }
    swap(num[max], num[len - 1]);
    select_sort_recursion(num, len - 1);
}
```

时间和空间复杂度分析同非递归实现的选择排序

6.插入排序（递归）

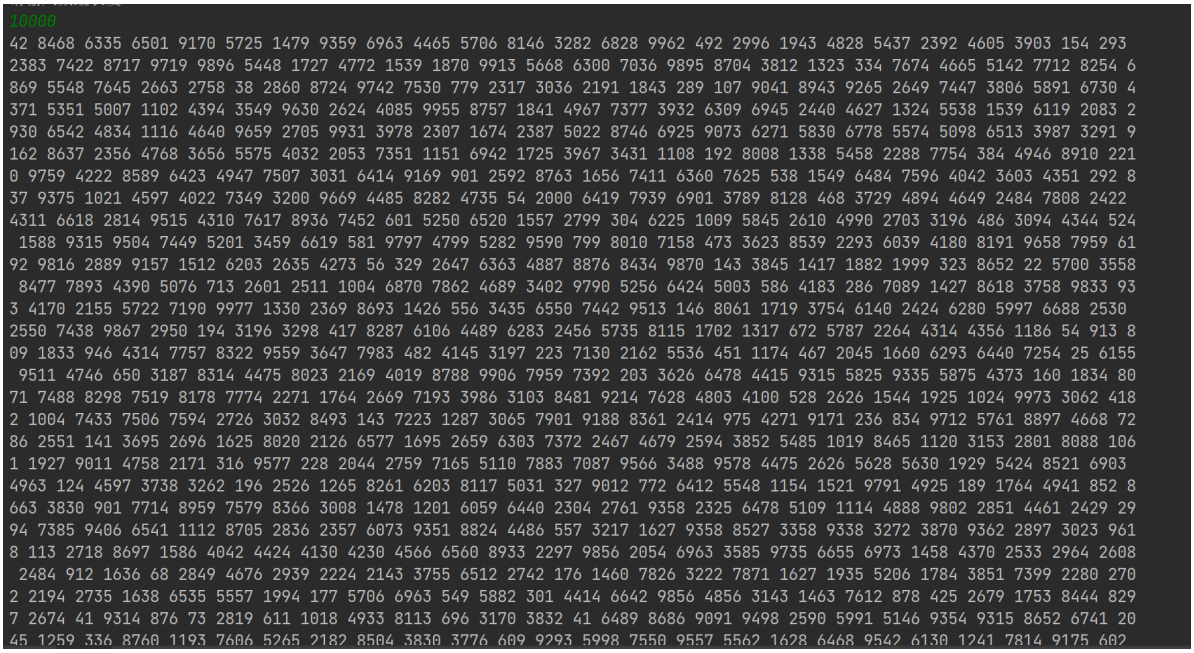
代码：

```
void insert_sort_recursion(int num[N], int len){
    if(len == 1) return;
    insert_sort_recursion(num, len - 1);//因为是从左到右进行插入，所以是先递归后执行操作
    int val = num[len - 1];
    int j;
    for(j = len - 2; j >= 0 && val < num[j]; j--){
        num[j + 1] = num[j];
    }
    num[j + 1] = val;
}
```

时间和空间复杂度分析同非递归实现的选择排序

7.验证

排序效果：排序前如下



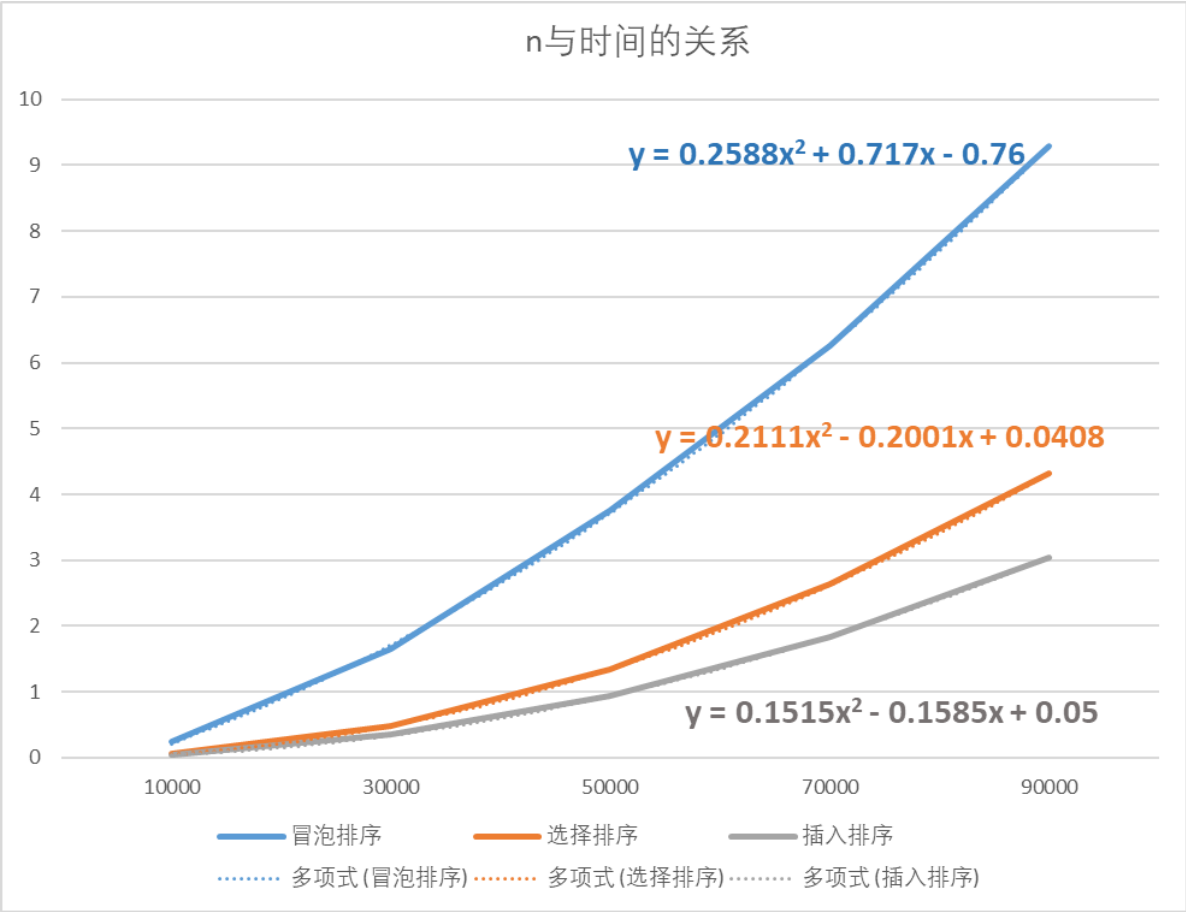
排序后如下：

```
1  1  3  4  4  4  5  5  7  9  9  10  13  13  13  13  15  17  17  18  21  21  22  22  23  25  25  27  28  29  29  29  31
 32  32  32  33  33  34  36  36  37  37  38  38  38  39  40  40  40  41  41  41  42  42  42  42  43  43  44  45  46  48
 48  49  49  49  50  51  51  54  54  54  54  54  56  58  58  58  58  58  59  59  60  62  63  66  66  66  67  67  68  68
 68  71  71  72  73  73  73  74  74  75  76  76  76  77  78  79  80  81  81  81  81  84  84  84  84  85  85  86  89  89
 90  90  91  91  91  94  94  96  97  99  100  102  102  102  102  103  103  103  104  104  104  105  107  107  107  107
 107  110  112  113  114  114  115  117  118  119  119  120  120  121  122  122  123  124  124  125  128  128  129  130
 130  131  131  133  133  133  134  135  135  135  136  136  137  138  138  138  139  140  140  140  141  142  143  143
 143  143  143  143  144  145  145  145  146  147  147  147  149  149  149  149  150  150  151  154  154  155  156  159
 160  160  160  161  163  164  166  166  168  169  169  169  169  170  170  170  171  171  171  172  174  175  176  176
 176  177  179  179  181  181  181  183  184  184  184  184  184  185  185  186  188  189  190  190  190  190  190  190
 191  192  192  194  194  194  195  196  196  196  199  200  200  202  203  203  203  204  205  205  206  206  209  210
 212  213  213  214  216  217  217  217  218  220  222  223  224  225  225  226  227  228  228  228  229  230  230  232
 232  235  236  237  237  238  238  239  239  239  242  242  243  246  246  249  251  253  254  254  254  259  259  260
 260  260  260  260  261  261  264  265  266  268  268  269  269  269  269  270  271  271  271  271  272  273  274  275
 277  279  279  280  281  283  283  284  285  285  286  286  286  288  288  289  289  290  290  291  292  293  294  294
 294  298  298  299  300  301  301  301  302  302  303  304  304  305  305  306  308  309  309  310  310  310  312  312
 313  314  314  316  317  319  319  320  321  321  323  324  324  324  325  327  327  329  329  331  334  334  334  334
 334  336  336  336  337  338  338  338  339  340  341  341  342  343  343  345  346  349  351  351  351  352  354  354
 354  355  356  357  358  360  362  363  364  366  367  367  370  370  371  372  372  373  374  374  378  378  379  379
 381  381  382  382  383  383  383  384  384  384  384  387  387  388  389  389  390  390  391  392  392  392  393  394
 394  394  395  396  397  397  397  398  398  399  401  402  403  404  404  404  406  407  407  407  409  412  413  416
 416  416  416  417  417  417  419  419  420  421  421  422  422  423  423  424  425  425  425  426  427  427  429  430
 431  433  433  433  433  434  436  436  438  438  440  440  441  441  442  444  445  445  445  445  448  449  449  449
 450  450  451  453  455  457  457  460  461  462  463  464  464  466  467  467  467  468  468  469  469  469  470  471
 471  471  472  472  473  473  475  475  475  475  476  476  477  477  477  479  480  480  480  482  482  482  483  483
 485  486  486  486  486  486  488  489  489  490  490  490  491  492  494  496  498  498  498  499  499  499  499  500
 503  503  503  504  504  504  505  507  507  507  509  510  510  511  512  515  519  519  520  522  523  523  524  524
 524  525  525  526  527  528  528  529  529  529  529  531  531  531  532  533  533  538  538  539  539  539  539  540
```

通过使用时间函数来对算法的时间复杂度进行验证，通过使用随机数种子来使得每次数组的数据都是相同的。可以绘制出得出如下的结果。由于递归次数过多会导致栈溢出，所以50000以后的数据为空。

次数	10000	30000	50000	70000	90000
冒泡排序	0.236	1.657	3.755	6.254	9.286
选择排序	0.054	0.483	1.335	2.629	4.315
插入排序	0.041	0.344	0.935	1.839	3.046
冒泡递归	0.205	2.057	NULL	NULL	NULL
选择递归	0.053	0.485	NULL	NULL	NULL
插入递归	0.039	0.366	NULL	NULL	NULL

对数据进行拟合，可以得到公式，可以看出与次数都是成平方阶的关系。



三、汉诺塔问题

代码

```
void hanoi(int n, char a, char b, char c){
    cout << "hanoi(" << n << ", " << a << ", " << b << ", " << c << ")" << endl;
    if (n == 1){
        cout << a << "-->" << c << endl;
        count++;
    }
    else{
        hanoi(n - 1, a, c, b);
        cout << a << "-->" << c << endl;
        count++;
        hanoi(n - 1, b, a, c);
    }
}
```

复杂度分析：

$h(1) = 1$

$h(2) = 2h(1) + 1 = 2 + 1$

$h(3) = 2h(2) + 1 = 2^2 + 2 + 1$

...

由数学归纳法 $h(n) = 2^{n-1} + 2^{n-2} \ldots + 1 = 2^n - 1$,所以其时间复杂度为$O(2^n)$

无需额外的辅助空间，所以其空间复杂度为$O(1)$

四、角谷猜想

代码

```cpp
#include "iostream"
#include "vector"
using namespace std;
//角谷猜想，并找出最长序列
vector<vector<int>> sequence;
vector<int> tmp;
void jiaogu(int n){
    tmp.push_back(n);
    if (n == 1){
        sequence.push_back(tmp);
        tmp.clear();
    }
    else{
        if (n % 2 == 0){
            jiaogu(n / 2);

        }
        else{
            jiaogu(3 * n + 1);
        }
    }
}
int main(){
    int n;
    for(int i = 2  ; i <= 100 ; i++){
        jiaogu(i);
    }
    //输出sequence中最长的序列
    int max = 0;
    int index = 0;
    for(int i = 0 ; i < sequence.size() ; i++){
        if(sequence[i].size() > max){
            max = sequence[i].size();
            index = i;
        }
    }
    cout << "max sequence: ";
    for(int i = 0 ; i < sequence[index].size() ; i++){
        cout << sequence[index][i] << " ";
    }
    cout << endl;
    return 0;
}
```

97时会出现最长序列

```
cos@LAPTOP-74D9CEQC:/mnt/c/Users/PC/Desktop/乱七八糟/数据结构/hw/hw1$ ./a.out
max sequence: 97 292 146 73 220 110 55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1
186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154
3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```