

**《算法分析与设计实训》**

**实训报告**

|  |  |
| --- | --- |
|  | |
|  |  |
| 学生姓名 | 欧阳闻奕 |
| 学 号 | 2022102069 |
| 所在专业 | 软件工程 |
| 所在班级 | 软件2201 |
| 指导教师 | 霍婷婷 |
| 提交时间 |  |
| 评阅情况 |  |
| 成 绩 |  |

# 实验一 蛮力法

## 一、实验内容

掌握蛮力的概念和基本思想，并结合具体的问题学习如何用相应策略进行求解的方法。

## 二、 实验步骤

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

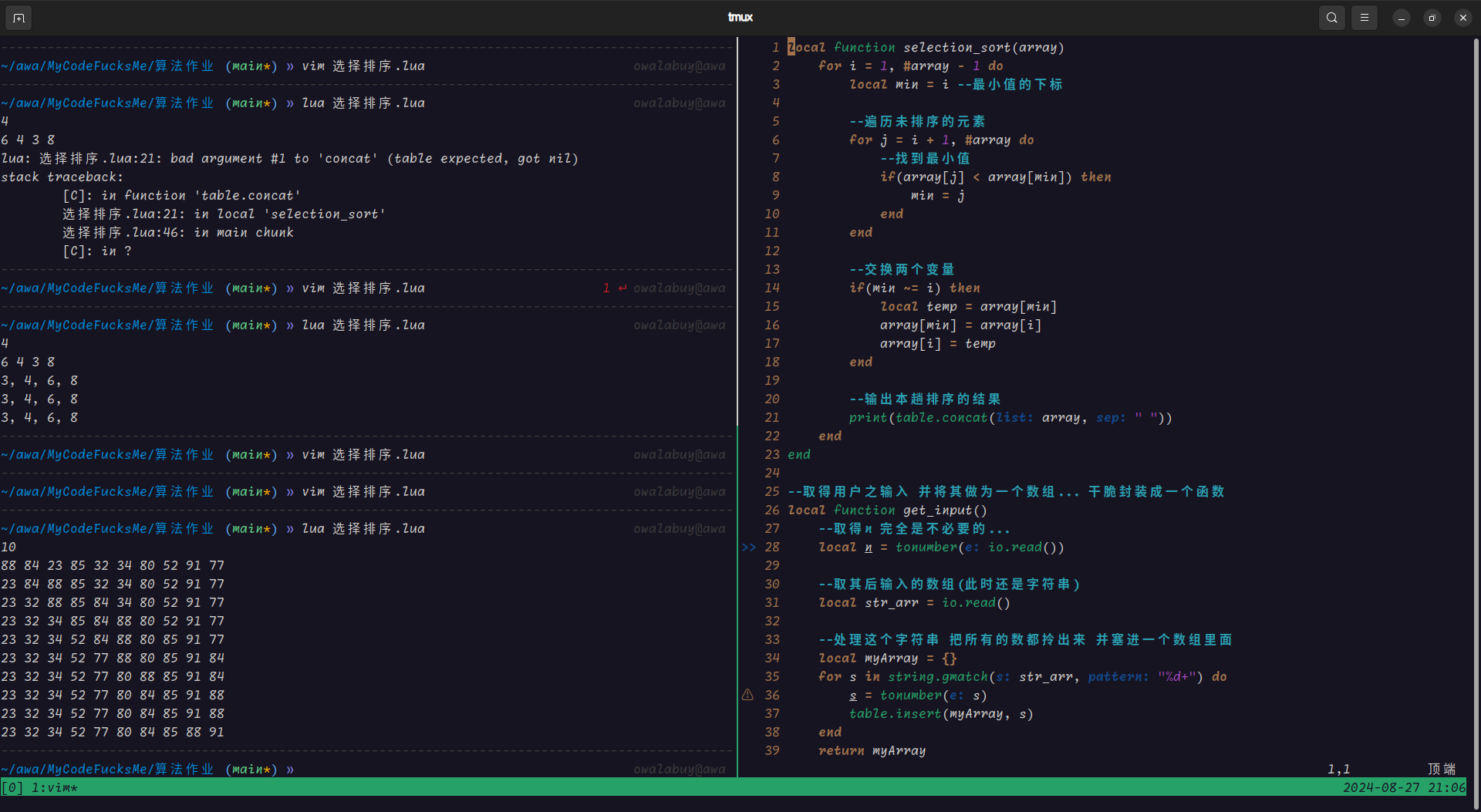
## 三、实验题目及过程

1. 【选择排序】第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。选择排序是不稳定的排序方法。

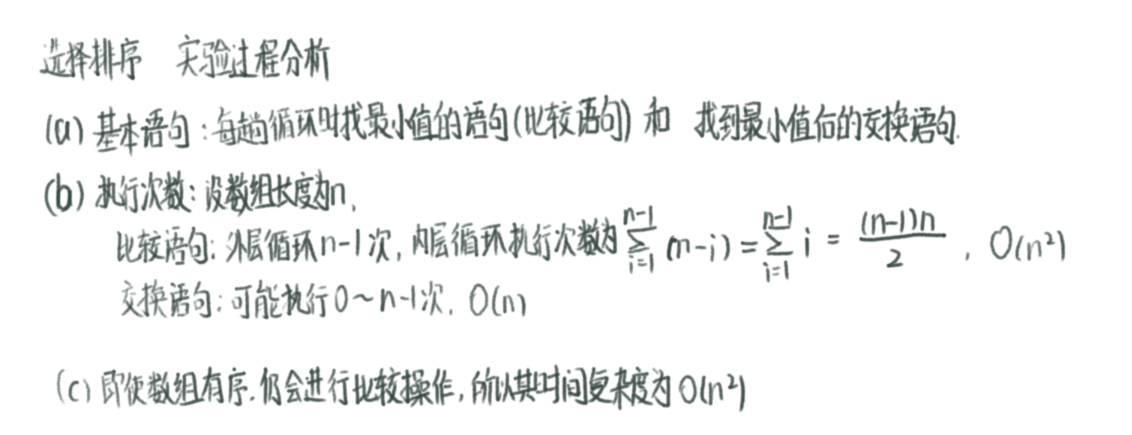
Sample Input：10 15 24 6 12 35 40 98 55

Sample Output：6 10 12 15 24 35 40 55 98

1. 实验源码
2. 实验结果



1. 实验过程分析：
2. 请给出该算法的基本语句？
3. 该算法的基本语句的执行次数是多少？
4. 请对该算法的时间复杂度进行分析。



2. 【冒泡排序】重复地走访过要排序的元素列，依次比较两个相邻的元素，如果顺序（如从大到小、首字母从Z到A）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。

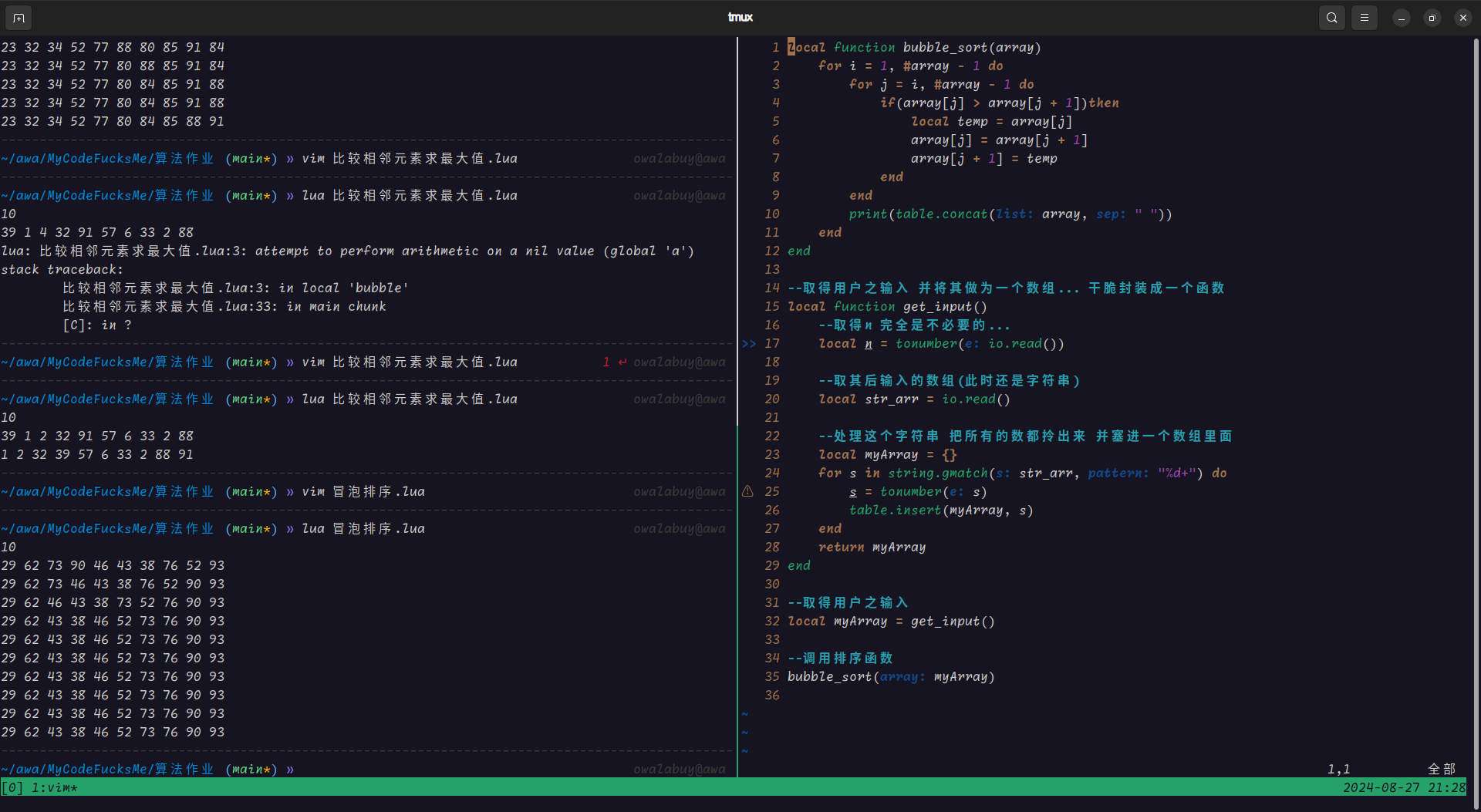
Sample Input：10 15 24 6 12 35 40 98 55

Sample Output：6 10 12 15 24 35 40 55 98

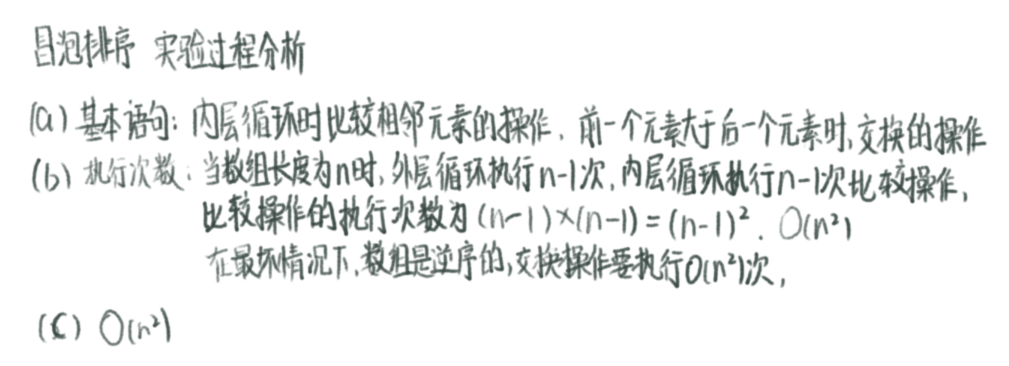
实验步骤：

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

1. 实验源码
2. 实验结果

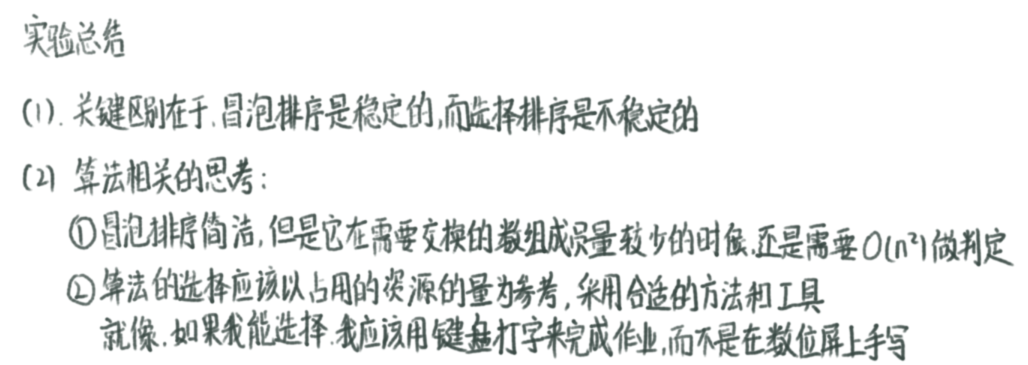
\*内层循环应该从1开始 在压缩文件中更正\*

1. 实验过程分析：
2. 请给出该算法的基本语句？
3. 该算法的基本语句的执行次数是多少？
4. 请对该算法的时间复杂度进行分析。



## 四、 实验总结

1. 选择排序和冒泡排序的关键区别在哪里？
2. 请给出实验过程中你对该类算法的思考。



# 实验二 分治法

## 一、实验内容

掌握分治法的概念和基本思想，并结合具体的问题学习如何用相应策略进行求解的方法。

## 二、 实验步骤

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

## 三、实验题目及过程

1. 【汉诺塔问题】相传在古印度圣庙中，有一种被称为汉诺塔(Hanoi)的游戏。该游戏是在一块铜板装置上，有三根杆(编号A、B、C)，在A杆自下而上、由大到小按顺序放置64个金盘。游戏的目标：把A杆上的金盘全部移到C杆上，并仍保持原有顺序叠好。操作规则：每次只能移动一个盘子，并且在移动过程中三根杆上都始终保持大盘在下，小盘在上，操作过程中盘子可以置于A、B、C任一杆上。

Sample Input： 3

Sample Output：

第1步:把第1个盘子A--->C

第2步:把第2个盘子A--->B

第3步:把第1个盘子C--->B

第4步:把第3个盘子A--->C

第5步:把第1个盘子B--->A

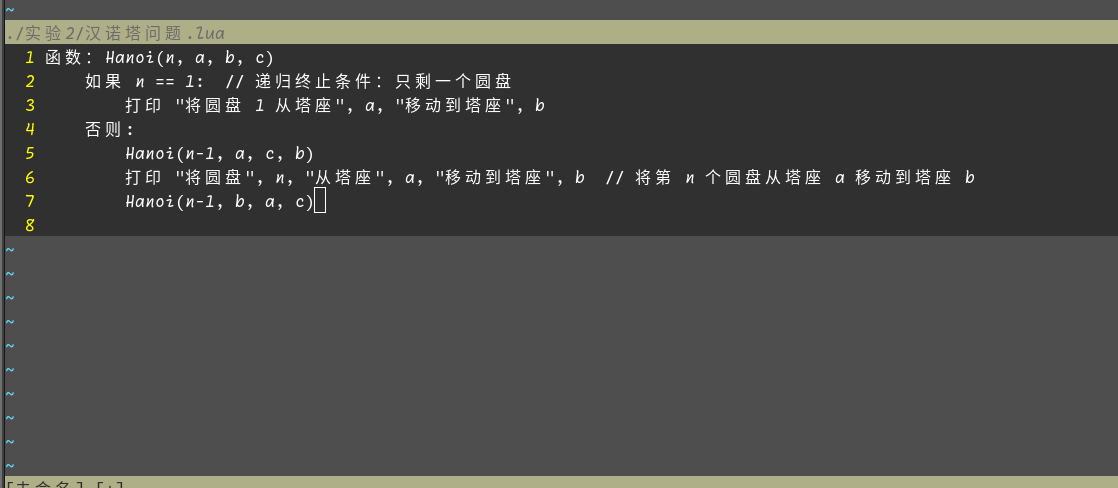
第6步:把第2个盘子B--->C

第7步:把第1个盘子A--->C

1. 实验源码
2. 实验结果



1. 实验过程分析：
2. 请用伪代码描述该算法中盘子的移动过程？



1. 该算法的所采用了哪一种算法的设计思想？  
   分治法(递归)

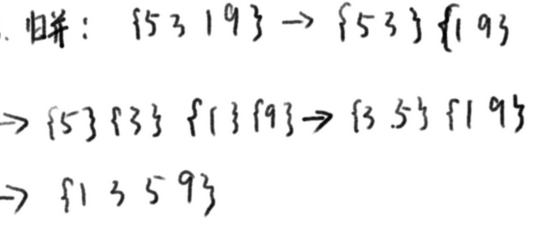
2. 【归并排序】归并排序（Merge Sort）是建立在归并操作上的一种有效，稳定的排序算法，该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

Sample Input：10 15 24 6 12 35 40 98 55

Sample Output：6 10 12 15 24 35 40 55 98

1. 实验源码
2. 实验结果



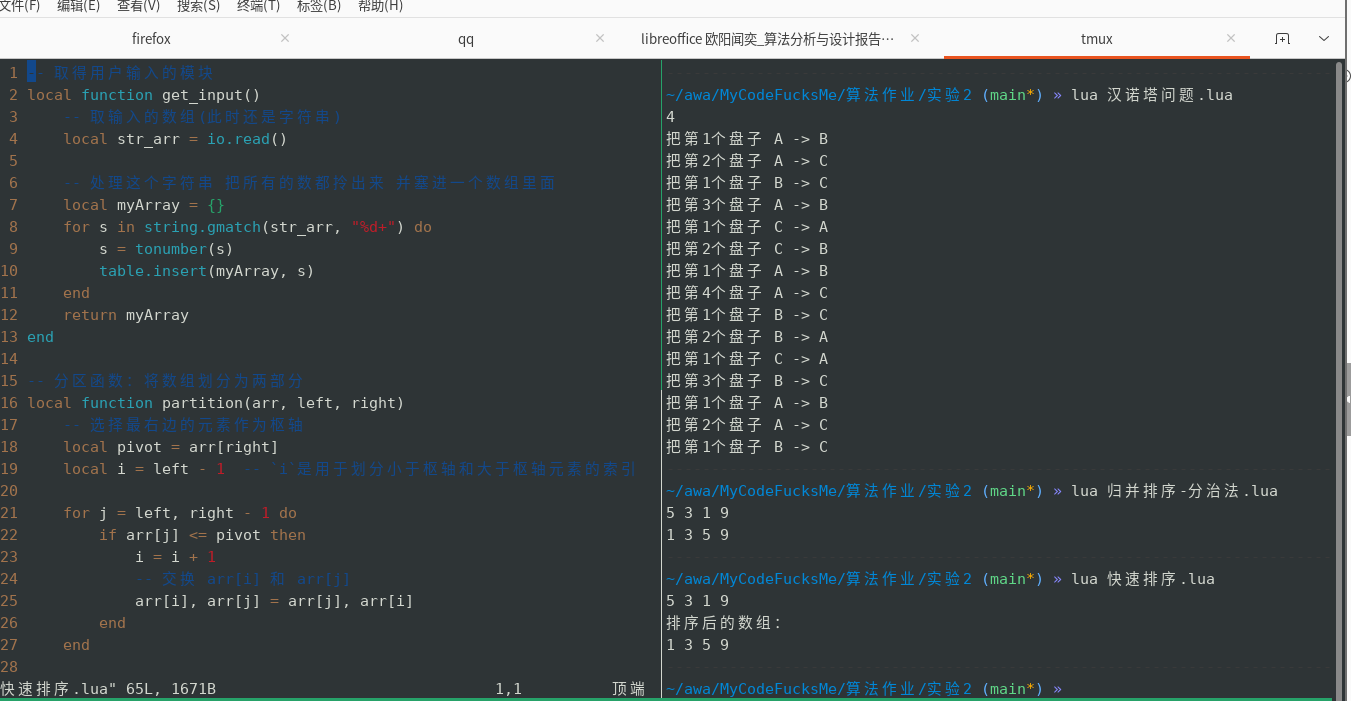
1. 实验过程分析：
2. 请用主定理对该算法进行复杂度分析？  
   O(nlogn)
3. 该用画图的方式给出本实验中输入样例的归并排序的执行过程？

3. 【快速排序】选择一个基准数，通过一趟排序将要排序的数据分割成独立的两部分；其中一部分的所有数据都比另外一部分的所有数据都要小。然后，再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

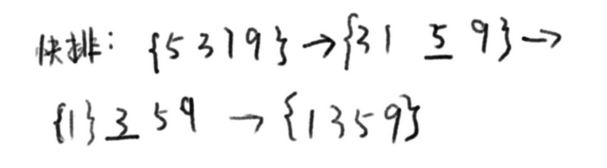
Sample Input：10 15 24 6 12 35 40 98 55

Sample Output：6 10 12 15 24 35 40 55 98

1. 实验源码
2. 实验结果



1. 实验过程分析：
2. 请用主定理对该算法进行复杂度分析？  
   O(nlogn)
3. 该用画图的方式给出本实验中输入样例的归并排序的执行过程？



## 四、 实验总结

1. 给出汉诺塔问题的递归过程详解图例。
2. 请根据实验分析归并排序和快速排序的异同点？

# 实验三 减治法

## 一、实验内容

掌握减治法的概念和基本思想，并结合具体的问题学习如何用相应策略进行求解的方法。

## 二、 实验步骤

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

## 三、实验题目及过程

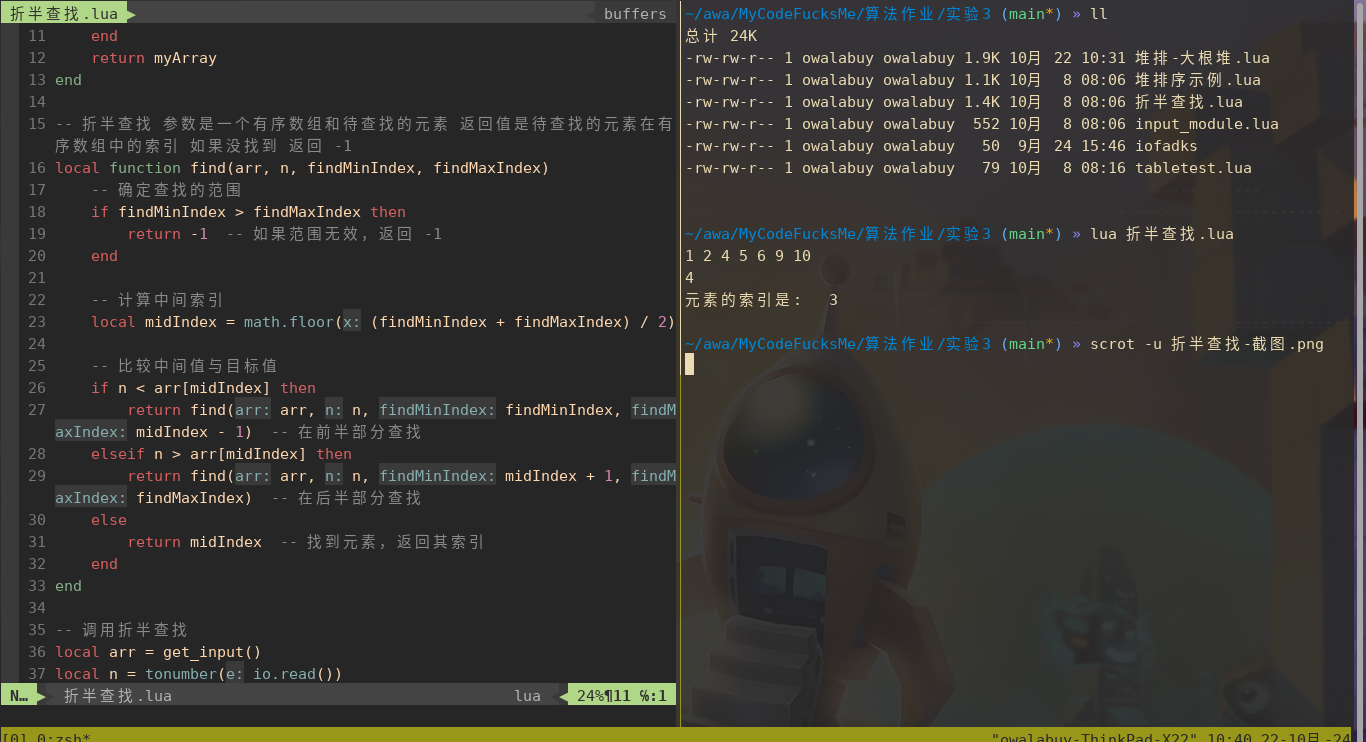
1. 【折半查找】在有序表（升序）中，取中间记录作为比较对象，若给定值与中间记录的关键码相等，则查找成功；若给定值小于中间记录的关键码，则在中间记录的左半区继续查找；若给定值大于中间记录的关键码，则在中间记录的右半区继续查找。不断重复上述过程，直到查找成功，或所查找的区域无记录，查找失败。

Sample Input：1 2 3 4 5 6 7 8 9 10

3

Sample Output：2

1. 实验源码
2. 实验结果



1. 实验过程分析：

请给出该算法的伪代码描述？

local function find(arr, n, findMinIndex, findMaxIndex)

-- 确定查找的范围

if findMinIndex > findMaxIndex then

return -1 -- 如果范围无效，返回 -1

end

-- 计算中间索引

local midIndex = math.floor((findMinIndex + findMaxIndex) / 2)

-- 比较中间值与目标值

if n < arr[midIndex] then

return find(arr, n, findMinIndex, midIndex - 1) -- 在前半部分查找

elseif n > arr[midIndex] then

return find(arr, n, midIndex + 1, findMaxIndex) -- 在后半部分查找

else

return midIndex -- 找到元素，返回其索引

end

end

2. 【堆排序】堆排序是利用堆（假设利用大根堆）的特性进行排序的方法，其基本思想是：首先将待排序的记录序列构造成一个堆，此时，选出了堆中所有记录的最大者即堆顶记录，然后将它从堆中移走（通常将堆顶记录和堆中最后一个记录交换），并将剩余的记录再调整成堆，这样又找出了次大的记录，以此类推，直到堆中只有一个记录为止。

Sample Input：10 15 24 6 12 35 40 98 55

Sample Output：6 10 12 15 24 35 40 55 98

1. 实验源码
2. 实验结果



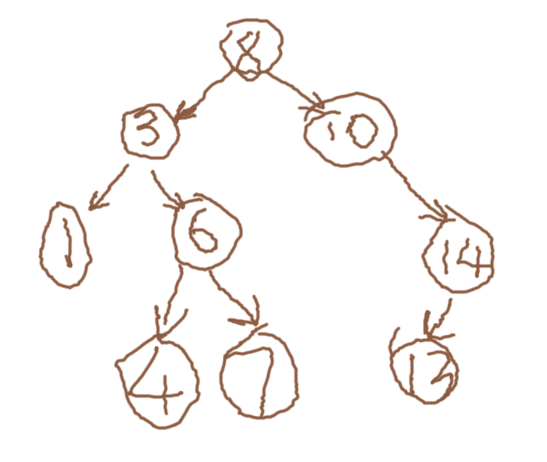
1. 实验过程分析：

请画出输入样例经历一趟堆排序的过程？

## 四、 实验总结

1. 折半查找可否用树的结构来实现，如果可以请画出对应的树的结构？

二叉查找树



1. 请给出堆排序的性能分析过程。

构建堆的过程为O(n) 排序的过程为O(log n) 所以时间复杂度为O(n log n)

由于它是原地进行调整 空间复杂度为O(1)

# 实验四 动态规划法

## 一、实验内容

掌握动态规划法的概念和基本思想，并结合具体的问题学习如何用相应策略进行求解的方法。

## 二、 实验步骤

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

## 三、实验题目及过程

1. 【0/1背包问题】有5个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为10，如何装能使背包的物品价值最大？

Sample Input：

请输入背包的容量：10

请输入物品数量：5

请输入每个物品的重量：2 2 6 5 4

请输入每个物品的价值：6 3 5 4 6

Sample Output：2

背包获得的最大价值是：15

装入背包的物品是：物品：1物品：2物品：5

1. 实验源码

#include <stdio.h>

#include <stdlib.h>

int \*\*V; // 用于存储动态规划表

int \*x; // 用于存储选择了哪些物品

int max(int a, int b);

int KnapSack(int n, int w[], int v[], int C);

int main() {

int C; // 背包容量

int m; // 物品数量

// 读取输入

scanf("%d", &C);

scanf("%d", &m);

int \*weights = (int \*)malloc(sizeof(int) \* m);

int \*values = (int \*)malloc(sizeof(int) \* m);

// 读取物品重量

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

scanf("%d", &weights[i]);

}

// 读取物品价值

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

scanf("%d", &values[i]);

}

// 为动态规划表分配内存

V = (int \*\*)malloc((m + 1) \* sizeof(int \*));

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

V[i] = (int \*)malloc((C + 1) \* sizeof(int));

}

// 用于标记装入了哪些物品

x = (int \*)malloc(sizeof(int) \* m);

// 调用 0/1 背包函数

int maxValue = KnapSack(m, weights, values, C);

// 输出最大价值

printf("%d\n", maxValue);

// 输出装入的物品

int first = 1;

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

if (x[i] == 1) {

if (!first) {

printf(" ");

}

printf("%d", i + 1);

first = 0;

}

}

printf("\n");

// 释放内存

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

free(V[i]);

}

free(V);

free(x);

free(weights);

free(values);

return 0;

}

int max(int a, int b) {

return a > b ? a : b;

}

int KnapSack(int n, int w[], int v[], int C) {

// 初始化动态规划表

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

V[i][0] = 0;

}

for (int j = 0; j <= C; j++) {

V[0][j] = 0;

}

// 动态规划计算

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

for (int j = 1; j <= C; j++) {

if (j < w[i - 1]) {

V[i][j] = V[i - 1][j]; // 不选第 i 个物品

} else {

V[i][j] = max(V[i - 1][j], V[i - 1][j - w[i - 1]] + v[i - 1]); // 选或不选第 i 个物品

}

}

}

// 回溯找到装入背包的物品

for (int j = C, i = n; i > 0; i--) {

if (V[i][j] > V[i - 1][j]) { // 如果选择了第 i 个物品

x[i - 1] = 1;

j -= w[i - 1];

} else {

x[i - 1] = 0;

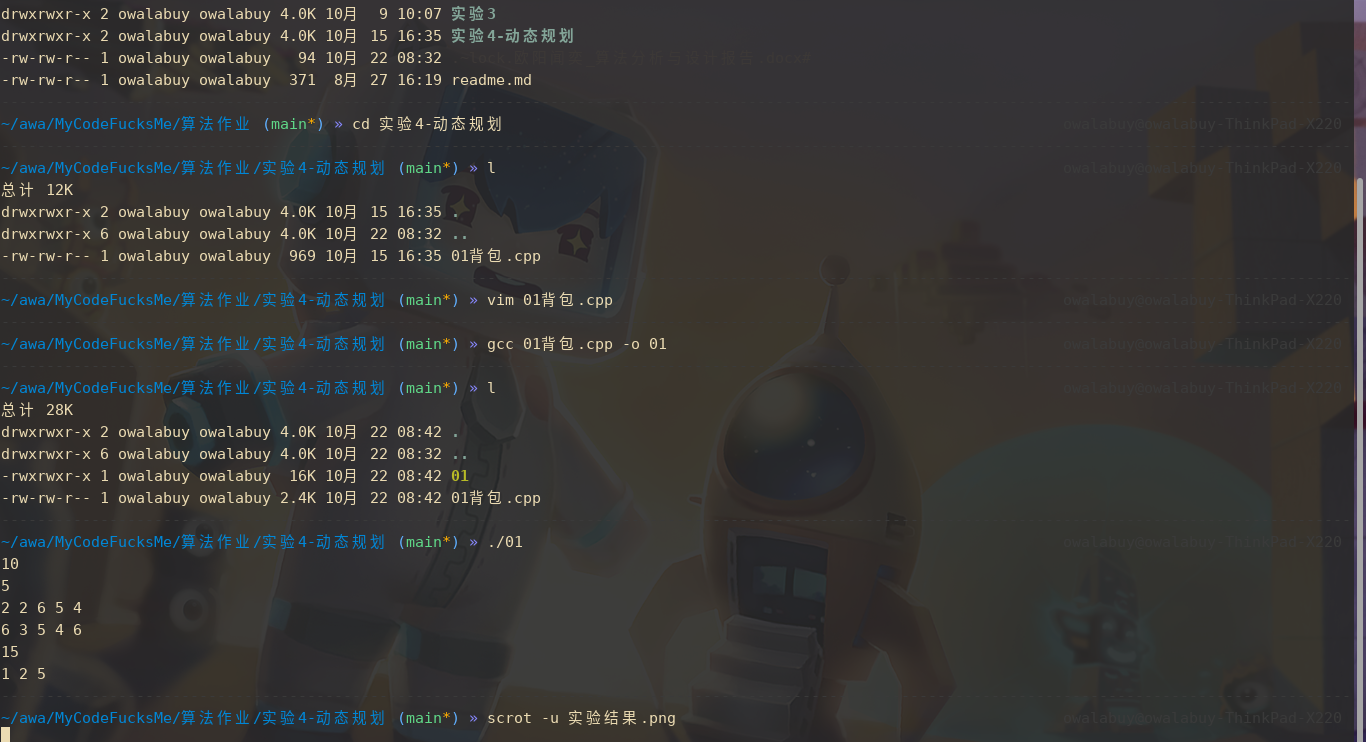
}

}

return V[n][C]; // 返回最大价值

}

1. 实验结果



1. 实验过程分析：
2. 请给出该算法的伪代码描述？

#include <stdio.h>

#include <stdlib.h>

int \*\*V; // 用于存储动态规划表

int \*x; // 用于存储选择了哪些物品

int max(int a, int b);

int KnapSack(int n, int w[], int v[], int C);

int main() {

int C; // 背包容量

int m; // 物品数量

// 读取输入

scanf("%d", &C);

scanf("%d", &m);

int \*weights = (int \*)malloc(sizeof(int) \* m);

int \*values = (int \*)malloc(sizeof(int) \* m);

// 读取物品重量

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

scanf("%d", &weights[i]);

}

// 读取物品价值

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

scanf("%d", &values[i]);

}

// 为动态规划表分配内存

V = (int \*\*)malloc((m + 1) \* sizeof(int \*));

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

V[i] = (int \*)malloc((C + 1) \* sizeof(int));

}

// 用于标记装入了哪些物品

x = (int \*)malloc(sizeof(int) \* m);

// 调用 0/1 背包函数

int maxValue = KnapSack(m, weights, values, C);

// 输出最大价值

printf("%d\n", maxValue);

// 输出装入的物品

int first = 1;

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

if (x[i] == 1) {

if (!first) {

printf(" ");

}

printf("%d", i + 1);

first = 0;

}

}

printf("\n");

// 释放内存

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

free(V[i]);

}

free(V);

free(x);

free(weights);

free(values);

return 0;

}

int max(int a, int b) {

return a > b ? a : b;

}

int KnapSack(int n, int w[], int v[], int C) {

// 初始化动态规划表

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

V[i][0] = 0;

}

for (int j = 0; j <= C; j++) {

V[0][j] = 0;

}

// 动态规划计算

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

for (int j = 1; j <= C; j++) {

if (j < w[i - 1]) {

V[i][j] = V[i - 1][j]; // 不选第 i 个物品

} else {

V[i][j] = max(V[i - 1][j], V[i - 1][j - w[i - 1]] + v[i - 1]); // 选或不选第 i 个物品

}

}

}

// 回溯找到装入背包的物品

for (int j = C, i = n; i > 0; i--) {

if (V[i][j] > V[i - 1][j]) { // 如果选择了第 i 个物品

x[i - 1] = 1;

j -= w[i - 1];

} else {

x[i - 1] = 0;

}

}

return V[n][C]; // 返回最大价值

}

1. 最优解可能导致背包没装满吗？  
   有可能 因为这是01背包问题 物品不可细分

2. 【袋鼠过河问题】一只袋鼠要从河这边跳到河对岸，河很宽，但是河中间打了很多桩子，每隔一米就有一个，每个桩子上都有一个弹簧，袋鼠跳到弹单簧上就可以跳的更远。每每个弹簧力量不同，用一个数字代表它的力量，如果弹簧力是为5，就代表袋鼠下一跳最多能够跳5米，如果为0，就会陷进去无法继续跳跃。河流一共N米宽，袋鼠初始位置就在第一个弹簧上面，要跳到最后一个弹簧之后就算过河了，给定每个弹簧的力量，求袋鼠最少需要多少跳能够到达对岸。如果无法到达输出-1。

Input

输入分两行、第一行是数组长度N(1=<N=<10000)，第二行是每一项的值，用空格分隔。

Output

输出只有一行，包含一个整数，为最少的跳数，无法到达输出-1。

Sample Input：

5

2 0 1 1 1

Sample Output：4

1. 实验源码

#include <stdio.h>

int minJumps(int n, int arr[]) {

if (n == 1) {

return 0; // 袋鼠已经在终点

}

int jumps = 0; // 记录跳跃次数

int farthest = 0; // 袋鼠能跳到的最远距离

int currentEnd = 0; // 当前这次跳跃的终点

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

// 更新袋鼠能跳到的最远距离

farthest = (i + arr[i] > farthest) ? i + arr[i] : farthest;

// 如果无法再继续跳跃，返回 -1

if (farthest <= i) {

return -1;

}

// 如果已经到达当前这次跳跃的最远距离，增加一次跳跃

if (i == currentEnd) {

jumps++;

currentEnd = farthest;

// 如果袋鼠能跳到终点，返回跳跃次数

if (currentEnd >= n - 1) {

return jumps;

}

}

}

// 如果袋鼠无法到达终点

return -1;

}

int main() {

int n;

// 输入数组的长度

scanf("%d", &n);

int arr[n];

// 输入弹簧的力量数组

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

scanf("%d", &arr[i]);

}

// 调用函数，输出最少的跳跃次数

int result = minJumps(n, arr);

printf("%d\n", result);

return 0;

}

1. 实验结果



1. 实验过程分析：

请分析袋鼠过河问题是否满足最优性原理，并找出动态规划函数的递推式。

用贪心算法实现的

从下标 i=0 开始 选择当前局部最优 能够达到的最远位置 然后和全局最优对比 更新全局最优 最后拿到整体最优解

## 四、 实验总结

动态规划算法使用条件以及求解过程？

使用条件:有重叠子问题和最优子结构并且子问题无后效性的问题可以使用动态规划法来求解

求解过程: 定义状态 找到递推式 设定初始条件 计算顺序(填表) 得到最终解

# 实验五 贪心法

## 一、实验内容

掌握贪心法的概念和基本思想，并结合具体的问题学习如何用相应策略进行求解的方法。

## 二、 实验步骤

理解算法思想和问题要求； 编程实现题目要求； 上机输入和调试自己所编的程序； 验证分析实验结果； 整理出实验报告。

## 三、实验题目及过程

1. 【找零钱问题】设计一个贪心法，使得找的钱币张数最少。

商店售货员找给 1 个顾客 n 元，用以下七种面值的纸币：100 元，50 元，20 元，10 元，5 元，2 元，1 元。

思考：如果商店售货员找给 1 个顾客 140 元，假设钱币的面值有九种：100 元，70 元，50 元，20 元，10 元，7 元，5 元，2 元，1 元。用贪心法得到的是该问题的最优解吗？

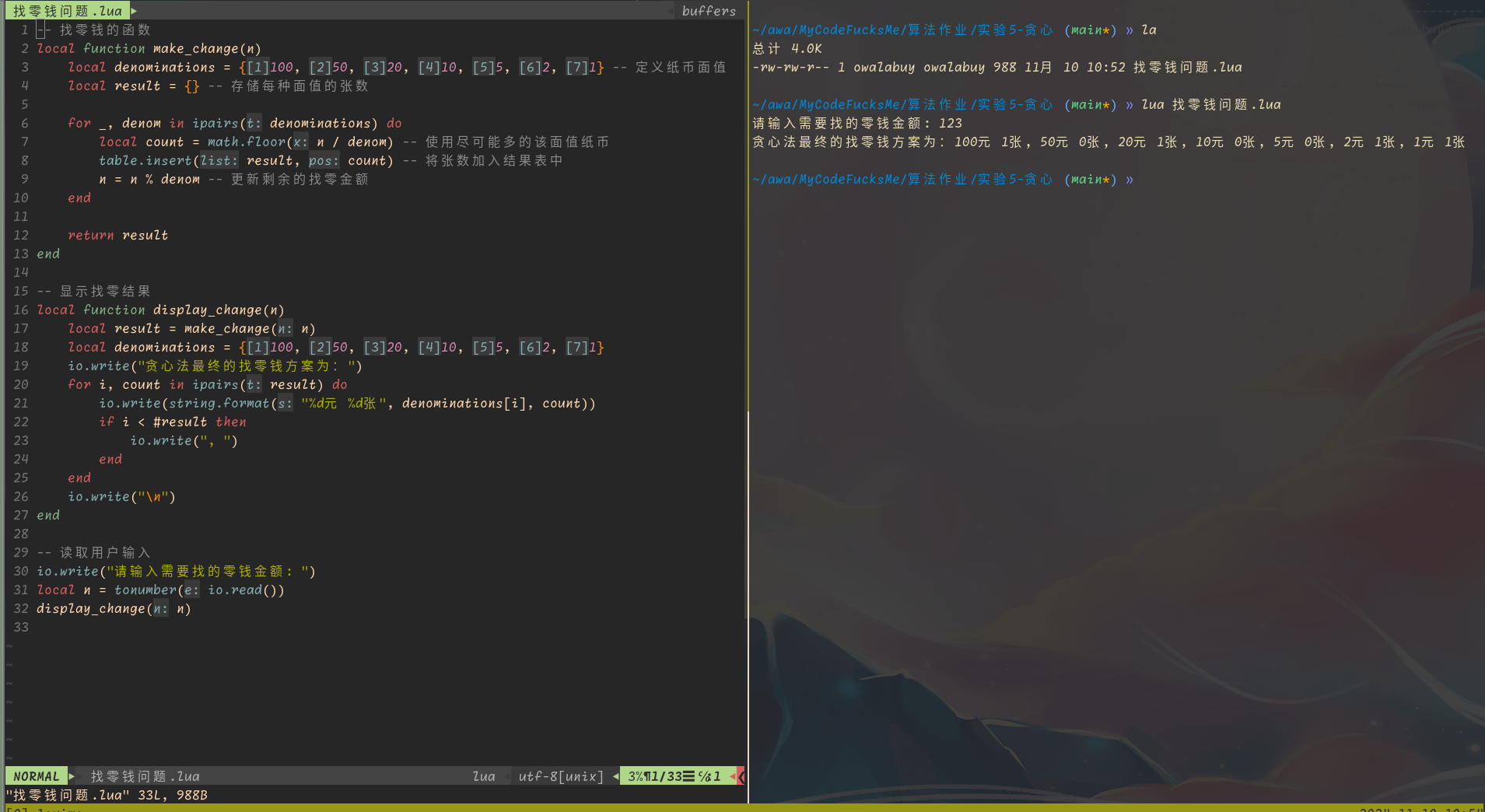
Sample Input：

需要找给顾客的钱 n元：123

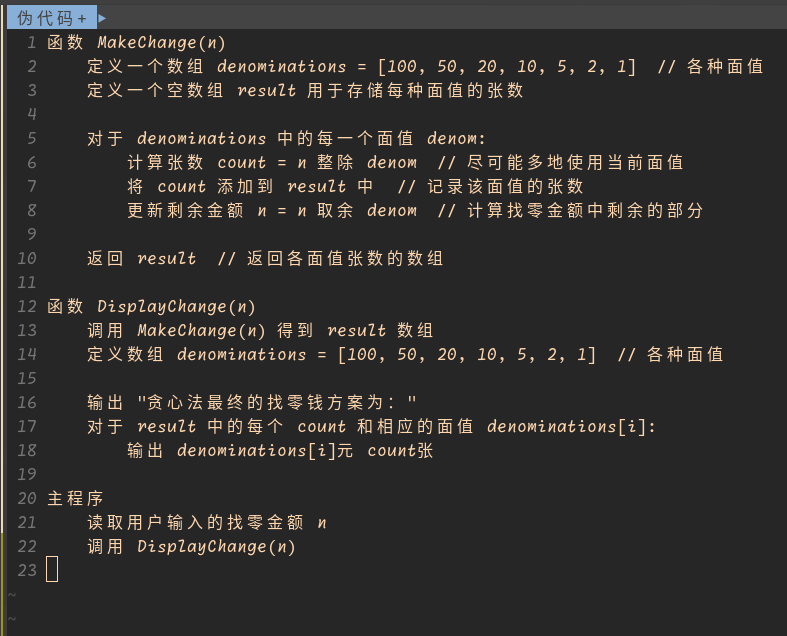
Sample Output：

贪心法最终的找零钱方案为：100元 1张，50元 0张，20元 1张，10元 0张，5元 0张，2元 1张，1元 1张

1. 实验源码
2. 实验结果



1. 实验过程分析：
2. 请给出该算法的伪代码描述？

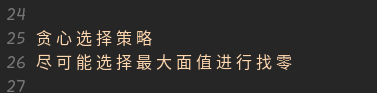


1. 贪心算法得到的是该问题的最优解吗？

可能不是最优解 因为每次都找最大的零钱 有可能出现剩下的钱只能用面值小的钱来找 这样钱的张数反而还更多了

## 四、 实验总结

1. 给出贪心法解决找零钱问题的贪心选择策略。



1. 贪心算法使用条件以及求解过程？

