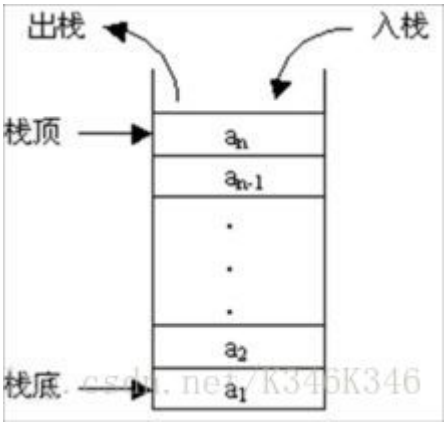
C语言第三讲

基本数据结构

栈

在栈（stack）中，被删除的是最近插入的元素：
栈实现的是一种后进先出策略（而**队列**则是先进先出）



栈上insert操作称为**压入**（push），而无元素参数的delete操作称为**弹出**（pop）。
餐馆里的一摞盘子，“弹出”的顺序和压入的顺序相反，因为只有最上面的盘子才能够被取下来。

堆

通常是一个可以被看做一棵树的数组对象.

算法

算法（Algorithm）是规则的有限集合，是为解决特定问题而规定的一系列操作。
算法设计的目标是正确、可读、健壮、高效、低耗
正确

1. 对于几组输入数据能够得出满足结果的要求
2. 对于精心选择的典型、苛刻的输入数据能够得出满足要求的结果
3. 对于一切合法的输入数据都能够产生满足要求的结果

一般情况下，至少应以第二层含义的正确性作为衡量一个算法是否正确的标准。

```
//求n个数的最大值问题，给出核心处理的示意算法
int max=0;
for(i=1;i<n;i++){
  scanf("%f",&x);
   if(x>max)
      max=x;
}
```

> 显然，当n个数全为负时，最大值max为0 ，这个算法的正确性是不够标准的

可读
鲁棒性
对非法输入的抵抗能力，即使输入非法数据，也能够识别并加以处理。
高效率、低储存量

一个算法的执行时间是指算法中所有语句执行时间的总和。
每条语句的执行时间等于该条语句的执行次数乘以执行一次所需实际时间。

语句频度是指该语句在一个算法中重复执行的次数，一个算法的时间耗费就是该算法中所有语句频度之和。

> 算法中语句总的执行次数$f(n)$是问题规模n的函数
> $T(n)=O(f(n))$，其中O是数量级

它表示随问题规模n的宏大，算法的执行时间的增长率和$f(n)$的增长率相同，称作算法的渐进时间复杂度，简称**时间复杂度**

常数阶

```
m=i;
i=j;
j=temp;
```

该程序段的执行时间是一个与问题规模n无关的常数。
$T(n)=O(1)$

线性阶

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

其时间复杂度为$O(n)$

```
for (i=1;1<=n;i++){
   for(j=1;j<=n;j++){
    y++;
   }
}
```

其时间复杂度为$O(n^2)$

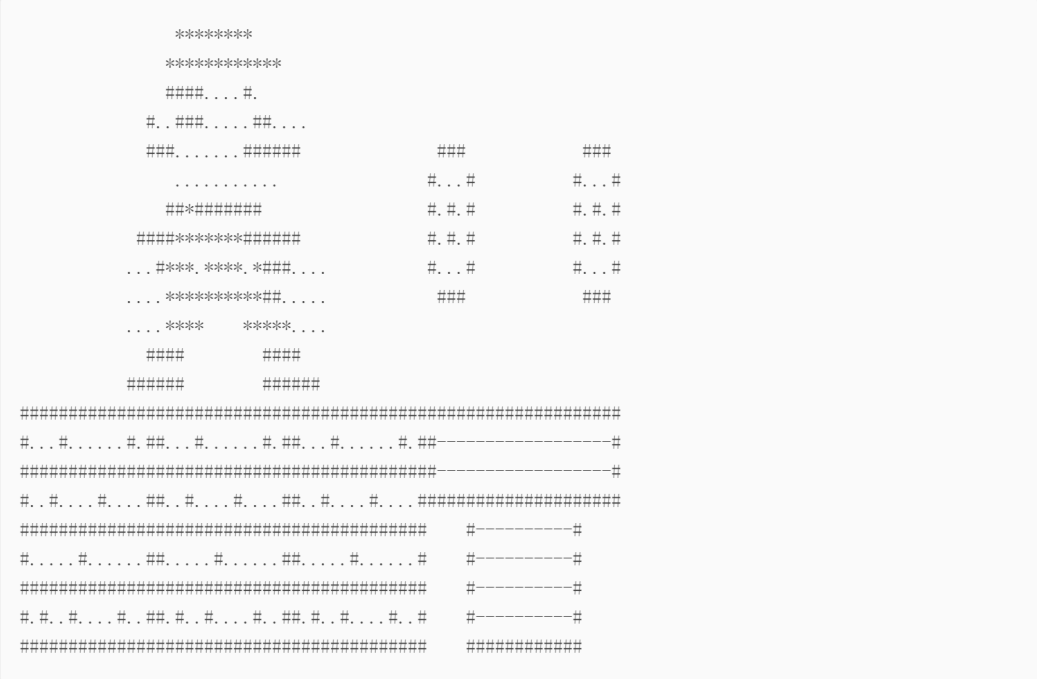
模拟

超级玛丽

模拟的过程就是对真实场景尽可能的模拟，然后通过计算机强大的计算能力对结果进行预测。

题目描述

超级玛丽是一个非常经典的游戏。请你用字符画的形式输出超级玛丽中的一个场景。



输入格式

无

输出格式

题解：

> https://www.luogu.com.cn/problem/solution/P1000

```
#include<stdio.h>
int main() {
    printf(
    "                 ********\n"
    "                **********\n"
```

```
    "                ####....#.\n"
    "              #..###.....##....\n"
    "              ###.......######              ###
###\n"
    "                 ...........                #...#
#...#\n"
    "                ##*#######                 #.#.#
#.#.#\n"
    "              ################              #.#.#
#.#.#\n"
    "            ...#***.****.*###....           #...#
#...#\n"
    "            ....**********##.....            ###
###\n"
    "            ....****    *****....\n"
    "              ####        ####\n"
    "            ######        ######\n"

 "##############################################################\n"
    "#...#......#.##...#......#.##...#......#.##-----------------
#\n"
    "#############################################-----------------
#\n"

 "#..#....#....##..#....#....##..#....#....####################\n"
    "########################################    #----------#\n"
    "#.....#......##.....#......##.....#......#    #----------#\n"
    "########################################    #----------#\n"
    "#.#..#....#..##.#..#....#..##.#..#....#..#    #----------#\n"
    "########################################    ############\n"
    );
    return 0;
}
```

多项式输出

题目描述

复制Markdown 展开

一元 n 次多项式可用如下的表达式表示：

$$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0,a_n\neq 0$$

其中，a_ix^i 称为 i 次项，a_i 称为 i 次项的系数。给出一个一元多项式各项的次数和系数，请按照如下规定的格式要求输出该多项式：

1. 多项式中自变量为 x，从左到右按照次数递减顺序给出多项式。
2. 多项式中只包含系数不为 0 的项。
3. 如果多项式 n 次项系数为正，则多项式开头不出 `+` 号，如果多项式 n 次项系数为负，则多项式以 `-` 号开头。
4. 对于不是最高次的项，以 `+` 号或者 `-` 号连接此项与前一项，分别表示此项系数为正或者系数为负。紧跟一个正整数，表示此项系数的绝对值（如果一个高于 0 次的项，其系数的绝对值为 1，则无需输出 1）。如果 x 的指数大于 1，则接下来紧跟的指数部分的形式为“x^b”，其中 b 为 x 的指数；如果 x 的指数为 1，则接下来紧跟的指数部分形式为 x；如果 x 的指数为 0，则仅需输出系数即可。
5. 多项式中，多项式的开头、结尾不含多余的空格。

输入格式

输入共有 2 行

第一行 1 个整数，n，表示一元多项式的次数。

第二行有 $n+1$ 个整数，其中第 i 个整数表示第 $n-i+1$ 次项的系数，每两个整数之间用空格隔开。

输出格式

输出共 1 行，按题目所述格式输出多项式。

输入输出样例

输入 #1 复制

```
5
100 -1 1 -3 0 10
```

输出 #1 复制

```
100x^5-x^4+x^3-3x^2+10
```

输入 #2 复制

```
3
-50 0 0 1
```

输出 #2 复制

```
-50x^3+1
```


题解：

> https://www.luogu.com.cn/problem/solution/P1067

枚举

三连击

题目背景

复制Markdown 展开

本题为提交答案题，您可以写程序或手算在本机上算出答案后，直接提交答案文本，也可提交答案生成程序。

题目描述

将 $1,2,\ldots,9$ 共 9 个数分成 3 组，分别组成 3 个三位数，且使这 3 个三位数构成 $1:2:3$ 的比例，试求出所有满足条件的 3 个三位数。

输入格式

无

输出格式

若干行，每行 3 个数字。按照每行第 1 个数字升序排列。

输入输出样例

输入 #1 复制

```
无
```

输出 #1 复制

```
192 384 576
* * *
...

* * *
（剩余部分不予展示）
```



```
192 384 576
219 438 657
273 546 819
327 654 981
//第一个数为a
```

> 从123（最小）枚举，abc;abc2;abc3
> 当然，还有一种方法：
> 三个数 之比分别为1:2:3，也可以将第一个数设为a，从1开始（123，246，369）依次枚举，直到满足条件
> *可以用三个数字的每一位相加为45作为限制*
> 题解：
> https://www.luogu.com.cn/problem/solution/P1008?page=1

递归

如果函数调用它本身，那么此函数就是递归的。

递归对要求函数调用自身两次或多次的复杂算法非常有帮助。

实际上，递归经常作为*分治法(divide-and-conquer)*技术的结果自然地出现。这种称为分治法的算法设计方法把一个大问题划分成多个较小的问题，然后采用相同的算法分别解决这些小问题。

分治法的经典示例就是流行的排序算法—快速排序(quicksort)。

汉诺塔

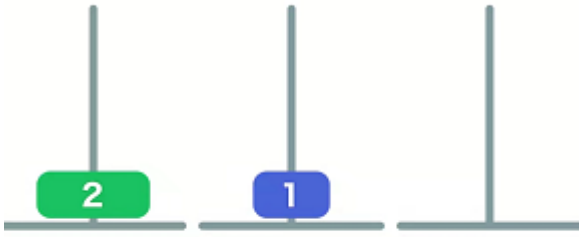
法国数学家爱德华·卢卡斯曾编写过一个印度的古老传说：在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着三根宝石针。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上地穿好了由大到小的64片金片，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小片必须在大片上面。僧侣们预言，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。

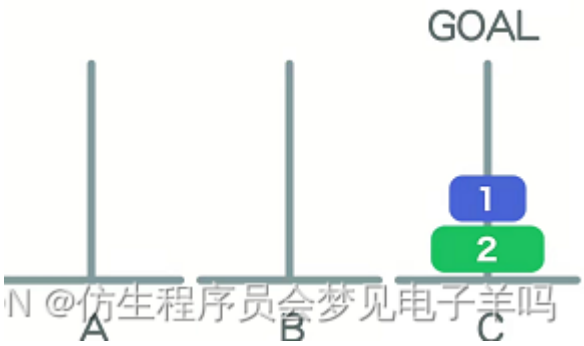
$2^n - 1$

一个：1次

```
1-1;2-3;3-7;4-15;
```

两个：3次





三个:7次

1->C

2->B

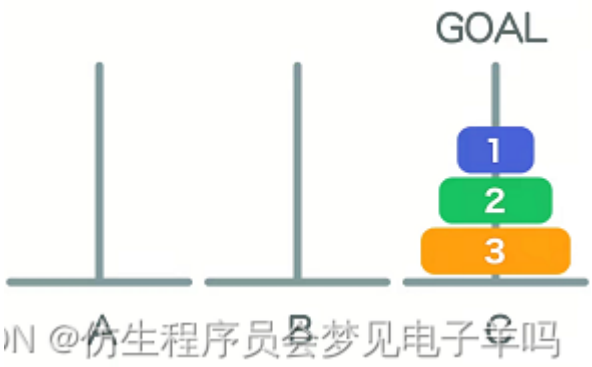
1->B

3->C

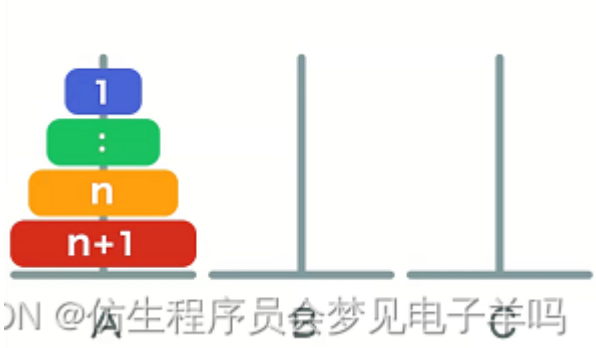
1->A

2->C

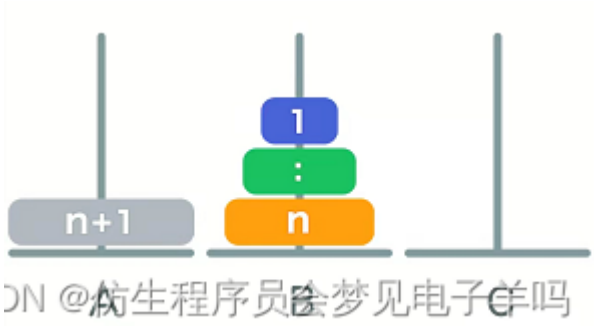
1->C



n个：



移动n个的次数+把最底层移动到C的次数（1次）+移动n个的次数



题解：

> https://blog.csdn.net/Y673789476/article/details/124569813?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-124569813-blog-82025409.t0_searchtargeting_v1&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-124569813-blog-82025409.t0_searchtargeting_v1&utm_relevant_index=2

题目描述

在你的帮助下，小A成功收集到了宝贵的数据，他终于来到了传说中连接通天路的通天山。但是这距离通天路仍然有一段距离，但是小A突然发现他没有地图！！！但是幸运的是，他在山脚下发现了一个宝箱。根据经验判断（小A有经验吗？），地图应该就在其中！在宝箱上，有三根柱子以及在一根柱子上的n个圆盘。小A在经过很长时间判断后，觉得这就是hanoi塔！（这都要琢磨）。但是移动是需要时间的，所以小A必须要通过制造延寿药水来完成这项任务。现在，他请你告诉他需要多少步完成，以便他造足够的延寿药水.。时限1s。

输入格式

一个数n，表示有n个圆盘

输出格式

一个数s，表示需要s步。

输入输出样例

输入 #1	输出 #1
31	2147483647

输入 #2	输出 #2
15	32767

说明/提示

对于所有数据n<=15000

很容易的练手题哦！

排序算法

算法	最坏情况运行时间	平均情况/期望运行时间
插入排序	$\Theta(n^2)$	$\Theta(n^2)$
归并排序	$\Theta(nlgn)$	$\Theta(nlgn)$
堆排序	$O(nlgn)$	—
快速排序	$\Theta(n^2)$	$\Theta(nlgn)$（期望）
计数排序	$\Theta(k+n)$	$\Theta(k+n)$
基数排序	$\Theta(d(k+n))$	$\Theta(d(k+n))$
桶排序	$\Theta(n^2)$	$\Theta(n^2)$（平均情况）

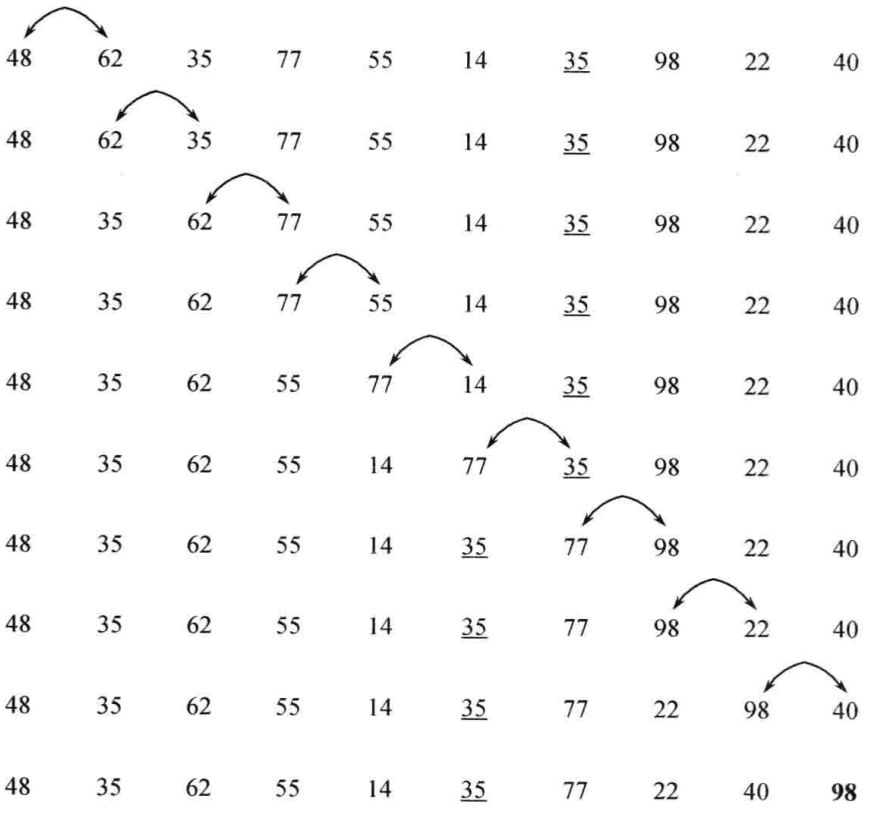
插入法

冒泡排序

冒泡排序（相邻比序法）是一种简单的交换类排序方法，它是通过对相邻的数据元素进行交换，逐步将待排序序列变成有序序列的过程。

> 反复扫描待排序记录序列，顺次比较相邻的两个元素的大小，若逆序就交换位置。
> 在扫描的过程中，不断地将相邻记录中关键字大的记录向后移动，最后必然将待排序记录序列中最大关键字换到序列的末尾，这也是最大关键字记录应在的位置
> 然后进行第二趟冒泡排序，对前n-1个记录进行同样的操作，其结果是使次大的记录被放在第n-1个记录的位置上
> 然后进行第三趟冒泡排序，对前n-2个记录进行同样的操作，其结果是使次大的记录被放在第n-2个记录的位置上
> 如此反复，每一趟冒泡排序都将一个记录排到位，直到剩下一个最小的记录。

如果在某一趟冒泡排序的过程中，没有发现一个逆序，则可直接结束整个排序过程，所以冒泡排序过程最多进行n-1次。
给出序列{48，62，35，77，55，14，35，98，22，40}的第一次冒泡排序过程。



(a) 一趟冒泡排序示例

分治算法

> 众所周知，弗兰大学的总图书馆距离寝室非常遥远，来回不便。
> 有一天热爱看书的电子羊同学到图书馆借了 N 本书，出图书馆的时候，警报响了，于是保安把热爱看书的电子羊同学拦下，要检查一下哪本书没有登记出借。热爱看书的电子羊同学正准备把每一本书在报警器下过一下，以找出引发警报的书，但是保安露出不屑的眼神：你连二分查找都不会吗？于是保安把书分成两堆，让第一堆过一下报警器，报警器响；于是再把这堆书分成两堆…… 最终，检测了 logN 次之后，保安成功的找到了那本引起警报的书，露出了得意和嘲讽的笑容。于是阿东背着剩下的书走了。
> 从此，图书馆丢了 N - 1 本书。

许多有用的算法在结构上是**递归**的：为了解决一个给定的问题，算法一次或多次递归地调用其自身以解决若干子问题。
这些算法典型地遵循**分治法**的思想，将原问题分解为几个规模较小但类似于原问题的子问题，递归地求解这些子问题，然后再合并这些子问题的解来建立原问题的解。

二分搜索

假设有n个呈升序排列的数组元素（例如1~100），当我们需要查找一个数a究竟在哪个位置时，可以使用二分搜索：
将数组分为两份，用a的值比较n/2的值，如果大于则在（n/2,n]中，反之则小于，若等于则查找完毕。
将n/2分成两份，重复上述操作，直到找出值相等为止。

快速排序

递归常用于**分治法**中。分治法是将一个大问题划分成多个较小的问题，然后采用相同的算法分别解决这些小问题。

分治法的经典示例就是流行的排序算法：快速排序

假设要排序的数组的下标从1到n。

1. 选择数组元素e(作为“分割元素”)，然后重新排列数组使得元素从1到i-1都是小于或等于元素e的，元素i包含e，而元素从i+1到n都是大于或等于e的
2. 通过递归地采用快速排序方法，对从1到i-1的元素进行排序

3. 通过递归地采用快速排序方法，对从i+1到n的元素进行排序。

开始，low指向数组中的第一个元素，而high指向末尾元素。

首先把第一个元素（分割元素）复制给其他地方的一个临时存储单元，从而在数组中留出一个“空位”。

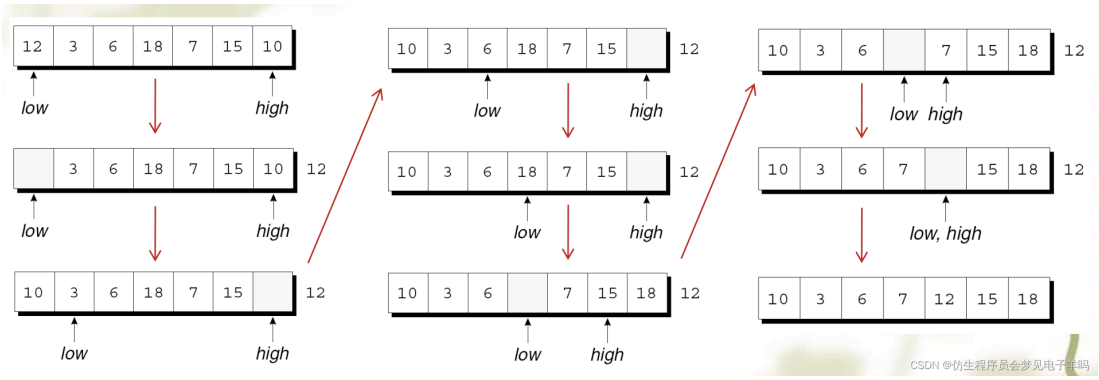
接下来，从右向左移动high，直到high指向小于分割元素的数时停止。

然后把这个数复制给low指向的空位，这将产生一个新的空位（high指向的）。

现在从左向右移动low，寻找大于分割元素的数。在找到时，把这个找到的数复制给high指向的空位。

重复执行此过程，交替操作low和high直到两者在数组中间的某处相遇时停止。

此时，两个标记都将指向空位：只要把分割元素复制给空位就够了。



从最后一个图可以看出，分割元素左侧的所有元素都小于或等于12，而其右侧的所有元素都大于或等于12。

既然已经分割了数组，那么就可以使用快速排序法对数组的前4个元素（10,3,6,和7）和后2个元素（15和18）进行递归快速排序了

让我们先来开发一个名为quicksort的递归函数，此函数采用快速排序算法对整型数组进行排序

程序qsort.c 将10个数读入一个数组，调用quicksort函数来对数组进行排序，然后打印数组中的元素：

```
Enter 10 numbers to be sorted: 9 16 47 82 4 66 12 3 25 51
In sorted order: 3 4 9 12 16 25 47 51 66 82
```

分割数组的代码放置在名为split的独立的函数中。


```c
#include <stdio.h>

#define N 10
```

```c
void quicksort(int a[], int low, int high);
int split(int a[], int low, int high);

int main(void)
{
  int a[N], i;

  printf("Enter %d numbers to be sorted: ", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);

  quicksort(a, 0, N - 1);

  printf("In sorted order: ");
  for (i = 0; i < N; i++)
    printf("%d ", a[i]);
  printf("\n");

  return 0;
}

void quicksort(int a[], int low, int high)
{
  int middle;

  if (low >= high) return;
  middle = split(a, low, high);
  quicksort(a, low, middle - 1);
  quicksort(a, middle + 1, high);
}

int split(int a[], int low, int high)
{
  int part_element = a[low];

  for (;;) {
    while (low < high && part_element <= a[high])
      high--;
    if (low >= high) break;
    a[low++] = a[high];

    while (low < high && a[low] <= part_element)
      low++;
    if (low >= high) break;
    a[high--] = a[low];
  }

  a[high] = part_element;
  return high;
}
```

动态规划

动态规划通常用来解决**最优化问题**。
这类问题有很多可行解，每个解都有一个值，我们希望寻找到具有最优值（最大值或最小值）的解。
我们称这样的解为问题的**一个最优解**，而非**最优解**（因为可能有多个解都达到最优值。）

钢条切割

0-1背包

贪心算法

贪心算法在每一步都做出当时看起来最佳的选择。
总是做出局部最优的选择，寄希望于这样的选择能够导致全局最优解。

运用贪心策略在每一次转化时都取得了最优解。问题的最优子结构性质是该问题可用贪心算法求解的关键特征。贪心算法的每一次操作都对结果产生直接影响。贪心算法对每个子问题的解决方案都做出选择，不能回退。

122. 买卖股票的最佳时机 II

难度 中等 1802 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `prices` ，其中 `prices[i]` 表示某支股票第 `i` 天的价格。

在每一天，你可以决定是否购买和/或出售股票。你在任何时候 **最多** 只能持有 **一股** 股票。你也可以先购买，然后在 **同一天** 出售。

返回 *你能获得的* ***最大*** *利润* 。

示例 1：

```
输入：prices = [7,1,5,3,6,4]
输出：7
解释：在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5 - 1 = 4 。
     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6 - 3 = 3 。
     总利润为 4 + 3 = 7 。
```

示例 2：

```
输入：prices = [1,2,3,4,5]
输出：4
解释：在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5 - 1 = 4 。
     总利润为 4 。
```

示例 3：

```
输入：prices = [7,6,4,3,1]
输出：0
```


> 贪心算法，一次遍历，只要今天价格小于明天价格就在今天买入然后明天卖出，时间复杂度O(n)