

排序+递归： 特殊的二进制序列

困难/排序、递归

学习目标

- 冒泡排序思想及代码实现
- 插入排序思想及代码实现
- 快速排序思想及代码实现
- 递归算法思想的应用



题目描述

特殊的二进制序列是具有以下两个性质的二进制序列：

- 0 的数量与 1 的数量相等。
- 二进制序列的每一个前缀码中 1 的数量要大于等于 0 的数量。

给定一个特殊的二进制序列 S ，以字符串形式表示。定义一个操作 为首先选择 S 的两个连续且非空的特殊的子串，然后将它们交换。（两个子串为连续的当且仅当第一个子串的最后一个字符恰好为第二个子串的第一个字符的前一个字符。）

在任意次数的操作之后，字符串中的字符按照字典序排列的最大的结果是什么？

输入: $S = "11011000"$

输出: $"11100100"$

解释: 将子串 "10"（在 $S[1]$ 出现）和 "1100"（在 $S[3]$ 出现）进行交换。这是在进行若干次操作后按字典序排列最大的结果。

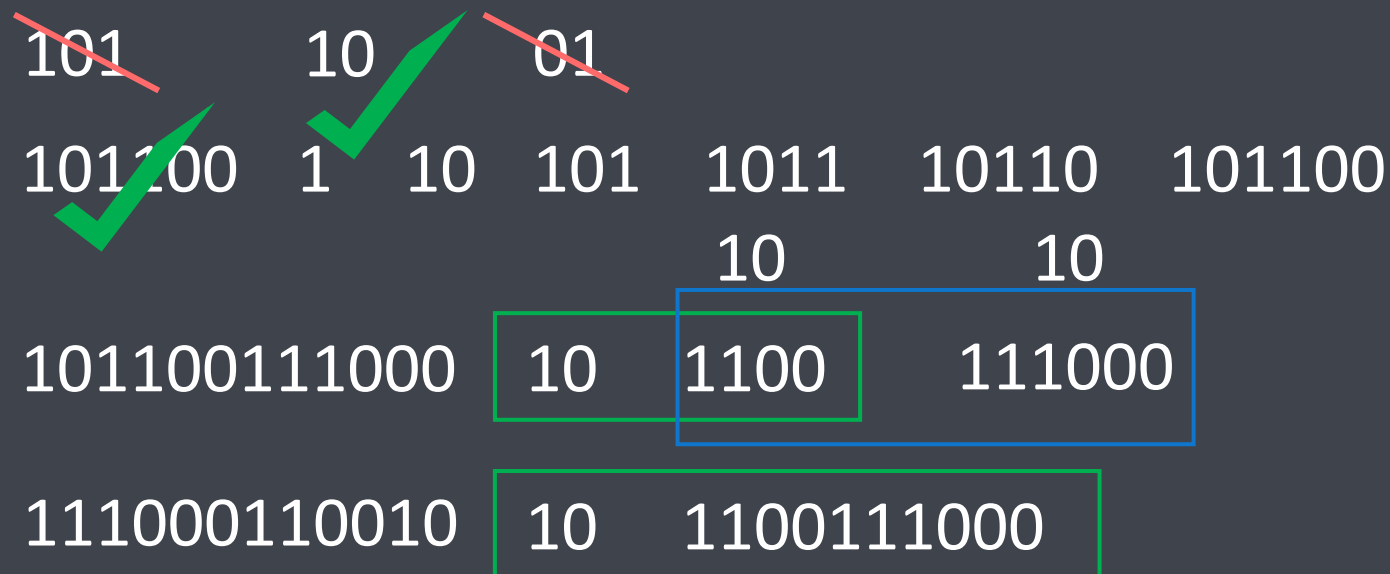
说明:

1. S 的长度不超过 50。
2. S 保证为一个满足上述定义的特殊 的二进制序列。

一. Comprehend 理解题意

题目主干要求

- 特殊二进制序列
 - 0的数量与1的数量相等
 - 每一个前缀码中1的数量 \geq 0的数量
- 交换
 - 两个连续且非空的特殊的子串交换
 - 任意次操作
 - 字典序最大的结果



限制要求：无

宽松条件：S长度不超过50、S满足特殊二进制序列

一. Comprehend 理解题意

细节问题

- 子串需符合规则（特殊二进制序列）
- 连续子串才可交换
- 使最终结果字典序最大

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

数据结构选择

- 输入和输出的参数类型为字符串
- 符合规则的特殊子串必定是1开头、0结尾
- 以第一个字符开头的合法特殊子串
 - 要么是整个字符串本身
 - 要么紧随其后的就是第二个合法特殊子串

111000110010

111000

110010

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

算法思维选择

- 首先要找到连续的子串——顺序查找
- 对连续子串按照字典序从大到小排列——排序问题
- 而子串中可能还会包含连续子串——递归问题

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

关键知识点：查找连续特殊子串

- 对1、0出现的次数计算差值

(球类运动中的盯人战术)

- 出现1，次数+1
- 出现0，次数-1



- 除起始计数0外，再次出现计数0就表示找到一个特殊子串
 - 计数为0的地方就是本次特殊子串的结束位置
 - 上个特殊子串结束位置的后一个字符是本次特殊子串的起始位置

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

关键知识点：排序

- 概念
 - 把一批任意序列的数据记录，按关键字重新排成一个有序的序列
- 冒泡排序（时间复杂度 $O(n^2)$ ）
- 插入排序（时间复杂度 $O(n^2)$ ）



二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

冒泡排序

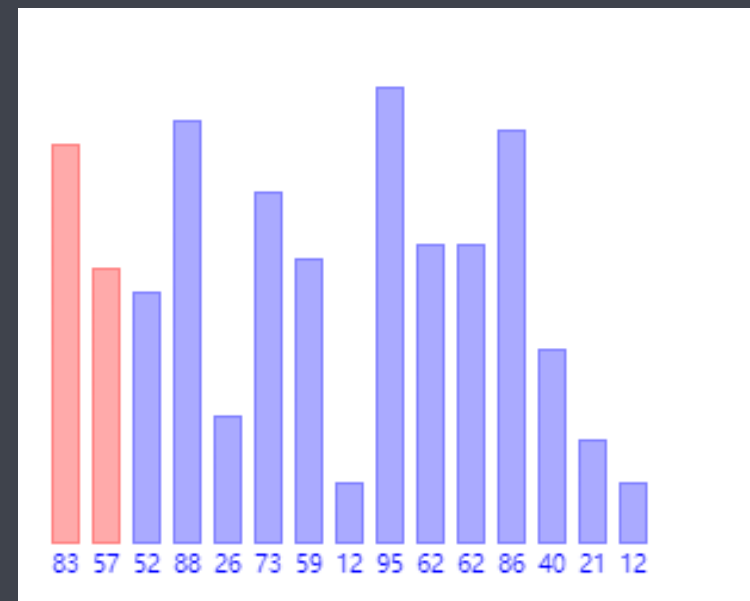
1. 对每一对相邻元素做比较，从开始的第一对到结尾的最后一对
 - 如果前面的大于后面的就交换元素
 - 最后的元素就是最大的数
2. 第二轮对除最后一个元素外的其余元素重复第1步
3. 第三轮对除最后两个元素外的其余元素重复第1步
4. 直到只剩下一个元素排序结束

关键字较小的记录像气泡逐趟向上飘浮

关键字较大的记录像石块往下沉

每一趟"最大"的石头沉到水底

重点



二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

冒泡排序

```
public static void bubbleSort(int arr[]) {  
    for (int i = 0; i < arr.length - 1; i++) {  
        for (int j = 0; j < arr.length - 1 - i; j++) {  
            if (arr[j] > arr[j + 1]) {  
                int temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
            }  
        }  
    }  
}
```



重点

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

插入排序

- 采用比较操作和移动操作交替地进行
- 待插入值 $R[i]$ 从右向左依次与有序区中记录 $R[j](j=i-1, i-2, \dots, 1)$ 比较

(1)若 $R[j] > R[i]$ ，则将 $R[j]$ 后移一个位置；

(2)若 $R[j] \leq R[i]$ 的关键字，则查找过程结束， $j+1$ 即为 $R[i]$ 的插入位置。

此时比 $R[i]$ 大的值均已后移， $j+1$ 位置已经腾空，将 $R[i]$ 直接插入此位置完成一趟插入排序

每步将一个待排序记录按关键字大小插入到有序区记录中适当位置

重点

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

插入排序

初始	[48]	<u>35</u>	18	45	12	68	33
i=1	[35	48]	<u>18</u>	45	12	68	33
i=2	[18	35	48]	<u>45</u>	12	68	33
i=3	[18	35	45	48]	<u>12</u>	68	33
i=4	[12	18	35	45	48]	<u>68</u>	33
i=5	[12	18	35	45	48	68]	<u>33</u>
i=6	[12	18	33	35	45	48	68]

二.Choose 数据结构及算法思维选择

拉勾教育

— 互联网人实战大学 —

插入排序

```
public static void insertionSort(int arr[]) {  
    for (int i = 1; i < arr.length; i++) {  
        int temp = arr[i];  
        for (int j = i - 1; j >= 0; j--) {  
            if ((arr[j] > temp)) {  
                //后移  
                arr[j + 1] = arr[j];  
                if (j == 0) arr[j] = temp;  
            } else {  
                arr[j + 1] = temp;  
                break;  
            }  
        }  
    }  
}
```



重点

三. Code 基本解法及编码实现

解题思路剖析

1. 遍历查找符合规则的连续子串
2. 对连续子串进行排序
3. 按字典序拼接字符串

子串处理 递归（三要素）

- 结束条件：无符合规则的特殊子串
- 函数主功能
- 函数的等价关系式
 - $f(n) = S(f(n1), f(n2), f(n3) \dots)$

(S函数为对子串进行排序拼接 n1、n2、n3为特殊子串)

细节问题

1110110000

1. 递归子串时需去掉头尾的1、0
2. 连续子串的存放（S最大长度为50，连续子串数量最大为25）
3. 升序排序，拼接字符串时要逆序拼接

三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度

- 遍历 $O(n)$, 排序 $O(k^2)$, 拼接 $O(k)$, 其中 k 为特殊子串数量

- 子串递归求解 $k * T(\frac{n}{k} - 2)$ $T(n-2k) = n-2k + T(n-2k-2 \ k') + k'^2 + k'$

- $T(n) = n + k * T(\frac{n}{k} - 2) + k^2 + k = n + T(n-2k) + k^2 + k$

$$= 2n + k^2 - k + T(n-2k-2 \ k') + k'^2 + k' \approx d * n + \sum k * (k - 1) \quad (d \text{ 递归深度, } k \text{ 子串数量})$$

k 最大为 $\frac{n}{2}$ 此时 d 为1, k 最小为1此时 d 为1到 $\frac{n}{2}$

- 总体时间复杂度 $O(n^2)$

- 空间复杂度

- 递归调用的深度 $O(d)$, 每一层递归都创建一个数组

三. Code 基本解法及编码实现

拉勾教育

— 互联网人实战大学 —

Java编码实现

```
public String makeLargestSpecial(String S) {
    if (S.length() <= 1) return S; // 结束条件
    StringBuilder sb = new StringBuilder();
    String[] arr = new String[25]; // 存储连续的子串, 符合要求的字符串必定是1开头0结尾的
    int index = 0; int start = 0; // 符合规则特殊子串的起始位置
    int countOne = 0; // 存放1、0的数量差
    for (int i = 0; i < S.length(); i++) { // 从前往后找, 以start开头是否存在符合要求子串
        countOne += S.charAt(i) == '1' ? 1 : -1; // 计算1、0的数量差
        if (countOne == 0) { // 找到一个特殊子串
            String result = makeLargestSpecial(S.substring(start + 1, i)); // 去掉头尾递归
            arr[index++] = "1" + result + "0"; // 将这个特殊子串放入数组中
            start = i + 1; // 下一个特殊子串的开始位置
        }
    }
    bubbleSort(arr, index - 1); // 对连续可交换的子串进行冒泡排序
    for (int i = index - 1; i >= 0; i--) // 排序后的连续子串 逆序(字典序最大) 拼接成字符串
        sb.append(arr[i]);
    return sb.toString(); // 返回排序后的字符串
}

public void bubbleSort(String arr[], int high) {
    for (int i = 0; i < high; i++) {
        for (int j = 0; j < high - i; j++) {
            if (arr[j].compareTo(arr[j + 1]) > 0) {
                String temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = temp;
            }
        }
    }
}
```

解答成功:

执行耗时: 2 ms, 击败了99.24% 的Java用户

内存消耗: 37.1 MB, 击败了84.69% 的Java用户

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

- 长度为25数组是必须的吗？
 - 如果题目不限定S最大长度该如何实现？
- 冒泡排序存在不必要的比较和移动吗？

是否有更好的算法思维

- 快速排序



四. Consider 思考更优解

关键知识点：快速排序

以某个记录为基准，通过比较、交换记录，将待排序列分成两部分

1. 所有记录小于基准记录的
2. 所有记录大于等于基准记录的

然后对这两部分记录继续快速排序，以达到整个序列有序

选择基准，划分元素，递归排序

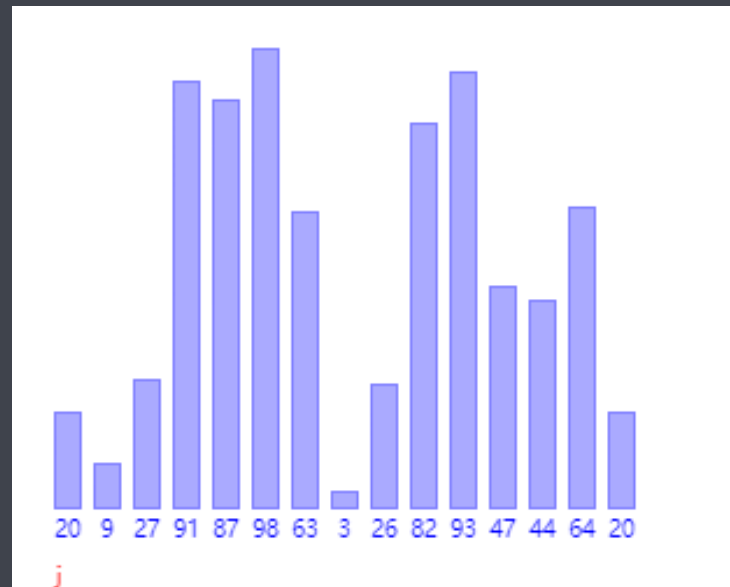


重点

四. Consider 思考更优解


快速排序

1. $i = \text{low}$, $j = \text{high}$, i 向后查找 , j 向前查找 , 取第一个记录为基准 $\text{temp} = r[\text{low}]$;
2. 当 $i < j$ 时进行比较和交换操作 , 具体操作如下 :
 - ① 当 $i < j$ 且 $r[j] \geq \text{基准}$ 时 , j 继续向前查找 ;
 - ② 若 $i < j$ 且 $r[j] < \text{基准}$ 时 , 将小于基准的记录前移 $r[i++] = r[j]$;
 - ③ 当 $i < j$ 且 $r[i] \leq \text{基准}$ 时 , i 继续向后查找 ;
 - ④ 若 $i < j$ 且 $r[i] > \text{基准}$, 将大于基准的记录后移 , $r[j--] = r[i]$;
3. 若 $i = j$, 在 $r[i]$ 处填入基准值 , 左子区和右子区划分结束 ;
4. 递归处理左子区 ;
5. 递归处理右子区。



四. Consider 思考更优解

快速排序



```
public static void quickSort(int arr[], int low, int high) {  
    int i = low; //i是向后搜索指针  
    int j = high; //j是向前搜索指针  
    int temp = arr[i];  
    while (i < j) {  
        while (i < j && arr[j] >= temp) j--; //arr[j] 不小于基准, 不用交换, 继续向前搜索  
        if (i < j) arr[i++] = arr[j]; //比基准小的记录移到前面  
        while (i < j && arr[i] <= temp) i++; //arr[i] 不大于基准, 不用交换, 继续向后搜索  
        if (i < j) arr[j--] = arr[i]; //比基准大的记录移到后面  
    }  
    arr[i] = temp; //确定基准记录位置  
    if (low < i - 1) quickSort(arr, low, i - 1); //递归处理左子区  
    if (high > i + 1) quickSort(arr, i + 1, high); //递归处理右子区  
}
```

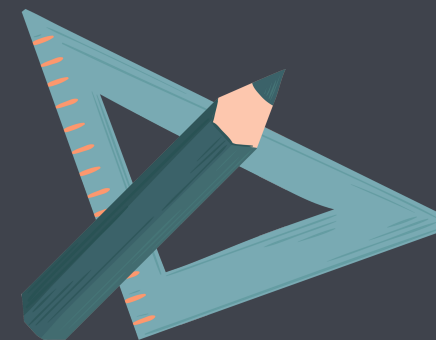
时间复杂度：最好情况 $O(n\log_2 n)$ ，最坏 $O(n^2)$ ，平均 $O(n\log_2 n)$

空间复杂度：划分均匀 $O(\log_2 n)$ ，最坏 $O(n)$

五. Code 最优解思路及编码实现

解题思路剖析

- 使用快速排序，将排序部分的时间复杂度降低到 $O(n\log n)$
- 使用List替代数组，减少无效空间占用



五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度

- 遍历 $O(n)$, 排序 $O(k\log k)$, 拼接 $O(k)$, 其中 k 为特殊子串数量
- 其中递归调用的时间复杂度不变
- 虽然排序部分的时间复杂度降低了, 但总的时间复杂度不变仍为 $O(n^2)$

- 空间复杂度

- 快速排序需要额外的空间消耗 $O(dk) \approx O(n)$, d 为子串递归深度, k 为子串数量
- 减少了数组的无效消耗, 与 n 无关的常数级
- 所以总空间复杂度为 $O(n)$

五. Code 最优解思路及编码实现

拉勾教育

— 互联网人实战大学 —

Java编码实现



执行耗时: 2 ms,击败了99.24% 的Java用户
内存消耗: 37 MB,击败了86.73% 的Java用户

```
public String makeLargestSpecial(String S) {
    StringBuilder sb = new StringBuilder();
    List<String> list = new ArrayList<>(); // 存储连续的子串, 符合要求的字符串必定是1开头0结尾的
    int start = 0; // 符合规则特殊子串的起始位置
    int countOne = 0; // 存放1、0的数量差
    for (int i = 0; i < S.length(); i++) { // 从前往后找, 以start开头是否存在符合要求子串
        countOne += S.charAt(i) == '1' ? 1 : -1; // 计算1、0的数量差
        if (countOne == 0) {
            String str = S.substring(start + 1, i); // 特殊子串去掉头尾进行递归
            String result = makeLargestSpecial(str); // 子串递归求解最大字典序
            list.add("1" + result + "0"); // 将这个特殊子串放到当前list中
            start = i + 1; // 下一个特殊子串的开始位置
        }
    }
    String[] arr = list.toArray(new String[list.size()]);
    quickSort(arr, 0, arr.length - 1); // 对连续可交换的子串进行双基准快速排序
    for (int i = arr.length - 1; i >= 0; i--) sb.append(arr[i]);
    return sb.toString(); // 返回排序后的字符串
}

public void quickSort(String arr[], int low, int high) {
    int i = low, j = high; // i是向后搜索指针 j是向前搜索指针
    String temp = arr[i];
    while (i < j) {
        while (i < j && arr[j].compareTo(temp) >= 0) j--; // arr[j] 不小于基准, 不用交换, 继续向前搜索
        if (i < j) arr[i++] = arr[j]; // 比arr[0]小的记录移到前面
        while (i < j && arr[i].compareTo(temp) <= 0) i++; // arr[i] 不大于基准, 不用交换, 继续向后搜索
        if (i < j) arr[j--] = arr[i]; // 比arr[0]大的记录移到后面
    }
    arr[i] = temp; // 确定基准记录位置
    if (low < i - 1) quickSort(arr, low, i - 1); // 递归处理左子区
    if (high > i + 1) quickSort(arr, i + 1, high); // 递归处理右子区
}
```


六. Change 变形延伸

题目变形

- 二进制序列变成左右括号序列该如何解答

延伸扩展

- 排序在工作场景中应用最广泛的，熟练掌握快速排序
- 快速排序是目前基于比较的内部排序中被认为是最好的方法
- Java中Collections、Arrays的sort()使用双基准快速排序（快速排序的一种变形）

题目来源

- Leetcode 761 <https://leetcode-cn.com/problems/special-binary-string>

总结

- 冒泡排序思想及代码实现
- 插入排序思想及代码实现
- 快速排序思想及代码实现
- 递归算法思想的应用

课后练习

拉勾教育

— 互联网人实战大学 —

1. 按奇偶排序数组 II ([Leetcode 922](#) / 简单)
2. 使用快速排序解答 数组的相对排序 ([Leetcode 1122](#) / 简单)
3. 有效的括号字符串 ([Leetcode 678](#) / 中等)
4. 最大数 ([Leetcode 179](#) / 中等)

拉勾教育

— 互联网人实战大学 —



下载「拉勾教育App」
获取更多内容