关键知识点

冒泡排序

java代码

```
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;
            }
        }
    }
}
```

插入排序

java代码

```
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;
            }
        }
    }
}
```

基本解法

Java代码

```java
public String makeLargestSpecial(String S) {
        //1. 结束条件
        //2. 函数主功能  遍历查找子串、子串排序、子串逆序拼接成字符串
        //3. 等价关系式 在子串排序前，先递归处理子串 f(n)=S(f(n1),f(n2),f(n3)...)
        if (S.length() <= 1) {
            return S;
        }
        StringBuilder sb = new StringBuilder();
        //存储连续的特殊子串，符合要求的字符串必定是1开头0结尾的
        String[] arr = new String[25];
        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

        //从前往后遍历查找，以start开头是否存在符合要求子串
        for (int i = 0; i < S.length(); i++) {
            //计算1、0的数量差
            countOne+=S.charAt(i)=='1'?1:-1;
            //countOne == 0 表示找到一个特殊子串
            if(countOne==0) {
                //对特殊字串去掉头尾1、0然后递归求解字典序最大
                String result=makeLargestSpecial(S.substring(start+1,i));
                //在递归结果上添加头尾1、0放入到字符串数组中
                arr[index++]="1"+result+"0";
                //记录特殊子串下标位置
                start=i+1;
            }
        }
        //对数组中的连续子串进行冒泡排序
        bubbleSort(arr,index-1);
        //排序后的连续子串 逆序（字典序最大）拼接成字符串
        for(int i=index-1;i>=0;i--){
            sb.append(arr[i]);
        }
        //返回排序后的字符串
        return sb.toString();
    }

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

```
    }
```

更优解知识点

快速排序

Java代码

```
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arr[0]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
}
```

Python

```
def partition(arr,low,high):
    i = ( low-1 )         # 最小元素索引
    pivot = arr[high]   #选择最右侧为基准元素
    for j in range(low , high): # 当前元素小于或等于 pivot
        if   arr[j] <= pivot:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]
    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# arr[] --> 排序数组
# low  --> 起始索引
# high  --> 结束索引
# 快速排序函数
def quickSort(arr,low,high):
    if low < high:
        pi = partition(arr,low,high)
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
```

C++

```
//快速排序（从小到大）
void quickSort(int left, int right, vector<int>& arr)
{
  if(left >= right)
    return;
  int i, j, base, temp;
  i = left, j = right;
  base = arr[left];  //取最左边的数为基准数
  while (i < j)
  {
    while (arr[j] >= base && i < j)
      j--;
    while (arr[i] <= base && i < j)
      i++;
    if(i < j)
    {
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
    }
  }
  //基准数归位
  arr[left] = arr[i];
  arr[i] = base;
  quickSort(left, i - 1, arr);//递归左边
  quickSort(i + 1, right, arr);//递归右边
}
```

优化解法

Java代码

```
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快速排序
Arrays.sort(arr,arr.length)
    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
}
```

Python

```
class Solution:
    def makeLargestSpecial(self, S: str) -> str:
        stkc, pos = 0, 0
        new_subs = []
        for i in range(len(S)):
            if S[i] == '0':
                stkc -= 1
            else:
                stkc += 1

            if stkc == 0:
                new_subs.append('1' + self.makeLargestSpecial(S[pos+1:i]) +
'0')
                pos = i + 1
        length=len(new_subs)
        self.quickSort(new_subs,0,length-1) #new_subs.sort(reverse=True)
        result=""
        for i in range(0, length):
```

```
            result+=new_subs[length-i-1]
        return result

    def partition(self,arr,low,high):
        i = ( low-1 )         # 最小元素索引
        pivot = arr[high]    #选择最右侧为基准元素
        for j in range(low , high): # 当前元素小于或等于 pivot
            if   arr[j] <= pivot:
                i = i+1
                arr[i],arr[j] = arr[j],arr[i]
        arr[i+1],arr[high] = arr[high],arr[i+1]
        return ( i+1 )

    # arr[] --> 排序数组
    # low  --> 起始索引
    # high  --> 结束索引
    # 快速排序函数
    def quickSort(self,arr,low,high):
        if low < high:
            pi = self.partition(arr,low,high)
            self.quickSort(arr, low, pi-1)
            self.quickSort(arr, pi+1, high)
```

C++

```
class Solution {
public:
    string makeLargestSpecial(string S) {
        int cnt = 0;
        vector<string> vc;
        int pos = 0;
        for (int i = 0; i < S.size(); i ++) {
            if (S[i] == '0') {
                cnt += -1;
            } else {
                cnt += 1;
            }
            if (cnt == 0) {
                vc.push_back('1' + makeLargestSpecial(S.substr(pos + 1, i - pos
- 1))+ '0');
                pos = i + 1;
            }
        }

        string ret;
        sort(vc.begin(), vc.end(), greater<string>());
```

```
        for (int i = 0; i < vc.size(); i ++) {
            ret += vc[i];
        }
        return ret;
    }
};
```