1.1 Merge Sort 归并排序

问题简述

Input: 长度为n的无序数组 arr

Output: 元素相同并保持non-decreasing顺序的 arr

时间复杂度

Merge Sort的时间复杂度为： O(nlogn)

算法思想

Merge sort 使用了 Divide and Conquer的算法思路，属于递归算法啊。

- 对于一个长度为1的算法，他本身就一定是个sorted items
- 对于一个长度不为一的算法，我们可以使用这样的思路：
    - 我们把input中的array一分为二，这样我们得到了两个行的input，我们称他们为 `left` 和 `right`
    - 然后我们对这两个部分分别apply我们的merge sort算法，在保证了我们算法output一直正确的情况下， `left` 和 `right` 会变成两个non-decreasing的array
    - 由于我们确保 `left` 和 `right` 都是non-decreasing的array，我们可以利用这个特性快速的合并原来的array
        - 创建一个array `result` 来储存我们的output
        - 我们比较 `left` 和 `right` 的第一个元素，他们应该分别是 `left` 和 `right` 中最小的元素

- 选择其中较小的一个元素，我们把其中**较小的值**append到 `result` 中，这样我们就有了理想的non-decreasing并包含所有元素的array
- 返回 `result`

Pseudocode

MergeSort

Algorithm: MergeSort

Input : A list A of n comparable items.
Output: A sorted list A.
if $|A| = 1$ **then return** A
$A_1 :=$ MergeSort(Front-half of A)
$A_2 :=$ MergeSort(Back-half of A)
return $Merge(A_1, A_2)$

Algorithm: Merge

Input : Two lists of comparable items: A and B.
Output: A merged list.
Initialize S to an empty list.
while *either A or B is not empty* **do**
  Pop and append $\min\{$front of A, front of $B\}$ to S.
end
return S

代码

下面是一个代码实例，我们发现我们还可以通过修改一下这个方法来计算inversion的次数，如下方代码所示

```
import sys
import math

def merge_sort(arr,size):
    #TODO: return a sorted arr and num of inversions

    #Base case: if there is only one element in the array, return it self,
    # and the number of inversions here is 0
    if size==1:
```

```
10            return arr, 0
11
12        #seperate the list by index
13        mid_point = math.ceil(size/2)
14        left = arr[0:mid_point]
15        right = arr[mid_point:]
16
17        #recursive left half:
18        left_sort_arr,left_inver_count = merge_sort(left,len(left))
19        #recursive right half
20        right_sort_arr,right_inver_count = merge_sort(right,len(right))
21
22        #merge two sorted array
23        sort_arr, merge_inver_count = merge(left_sort_arr,right_sort_arr)
24
25        #calculate total inversions
26        total_inver = left_inver_count+right_inver_count+merge_inver_count
27
28        return sort_arr, total_inver
29
30    def merge(left,right):
31        #TODO: merge two sorted array, it can be zero
32        sorted_arr = []
33        inver_count = 0
34
35        while (len(left)≠0) or (len(right)≠0):
36
37            #if left is empty, pop right
38            if(len(left)==0):
39                sorted_arr.append(right.pop(0))
40            #if right is empty, pop left
41            elif(len(right)==0):
42                sorted_arr.append(left.pop(0))
43            #both not empty, compare first element
44            else:
45                if(left[0]⩽right[0]):
46                    sorted_arr.append(left.pop(0))
47                else:
48                    sorted_arr.append(right.pop(0))
49                    inver_count += len(left)
50        return sorted_arr, inver_count
```