最长递增子序列

问题描述：　给出长度为N的数组,找出这个数组的最长递增子序列的长度。(递增子序列是指,子序列的元素是递增的)
例如：5 1 6 8 2 4 5 10,最长递增子序列是1 2 4 5 10长度为5

问题分析：　采用分治法思想

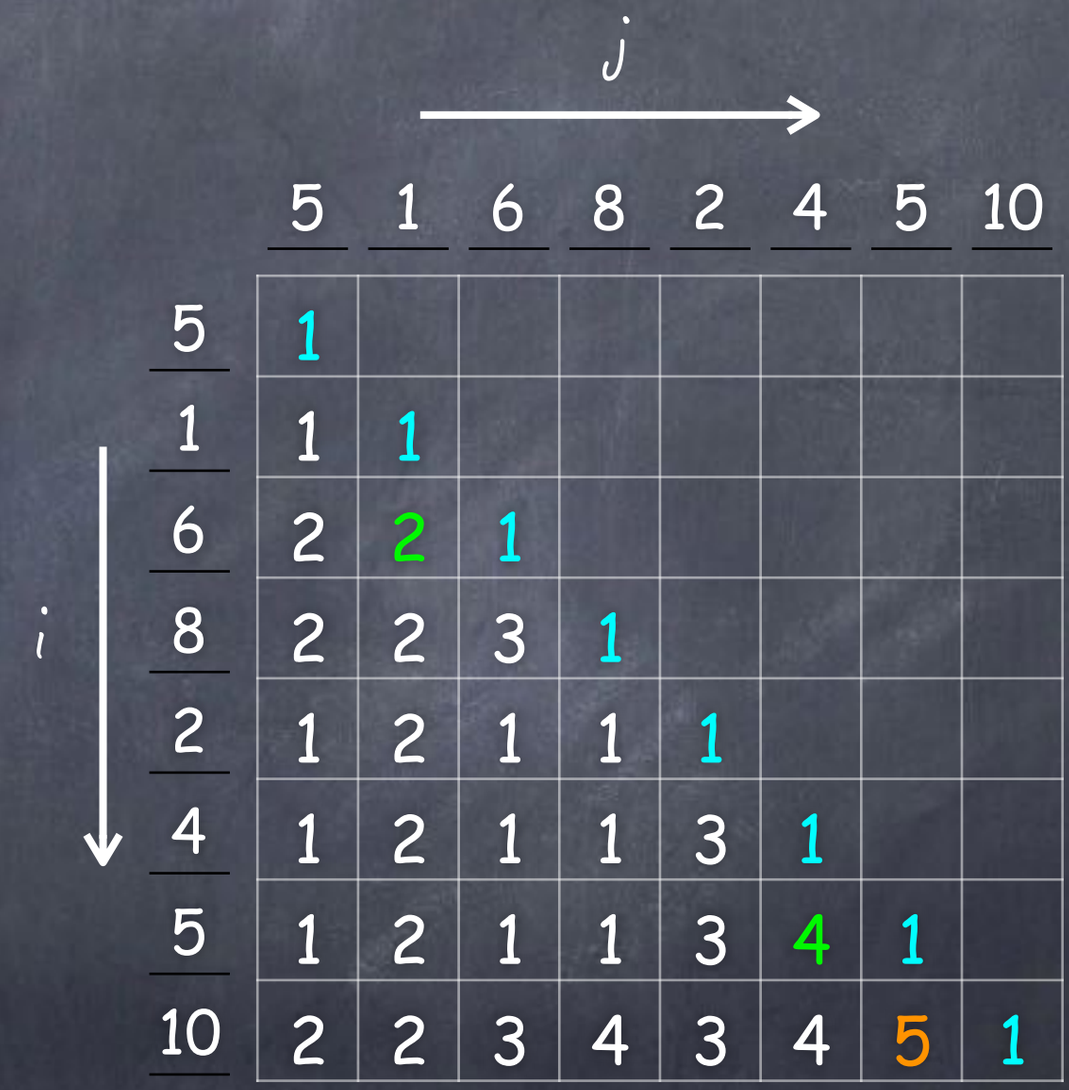
1. 假设先只有一个元素5,那么最长递增子序列为(5)
2. 增加一个元素1,最长递增子序列为(5)或者(1)
3. 再加一个元素6,则为(6),(5,6),(1,6)
4. 加入8: (8),(5,8),(1,8),(6,8),(5,6,8),(1,6,8)
5. 加入2: (2),(1,2)
6. 加入4: (4),(1,4),(2,4),(1,2,4)
7. 加入5: (5),(1,5),(2,5),(1,2,5),(4,5),(1,4,5),(2,4,5),(1,2,4,5)
8. 加入10:
   (10),(5,10),(1,10),(6,10),(5,6,10),(1,6,10),(8,10),(5,8,10),
   (1,8,10),(6,8,10),(1,6,8,10),(2,10),(1,2,10),(4,10),(1,4,10),
   (2,4,10),(1,2,4,10),(5,10),(1,5),(2,5,10),(1,2,5,10),(4,5,10),
   (1,4,5,10),(2,4,5,10),(1,2,4,5,10)

可以看到每新加一个元素,如果比上一个元素大,就取上一个元素的最大长度加1,如果比上一个元素小,就置1

可以得出递推公式为：L[i]=max(L[j]+1,1)当j<i and A[j]<A[i]

具体打表：

1. 初始化对角线为1
2. 对每一个i,遍历j(0到i-1)
3. 若A[i]<=A[j],置1
4. 若A[i] > A[j],取第j行的最大值加1



说明：

1. 当i=2, j=1时,A[i]=6>A[j]=1
2. 取j行,取第1行最大值1加1=2

1. 当i=7, j=6时,A[i]=10>A[j]=5
2. 取j行,取第6行最大值4加1=5

Python代码实现：

```
# -*- coding: utf-8 -*-
A = [5,1,6,8,2,4,5,10] #原序列

d = [1]*len(A) #用于存放每行的最大长度值
res = 1 #用于记录最终的最大长度
for i in range(len(A)):
    for j in range(i):
#由于一行只记录一个最大值,需要加上判断条件
        if A[i] > A[j] and d[i] < d[j]+1:
            d[i] = d[j]+1
    if d[i] >  res:
        res = d[i]
print res
```

时间复杂度：

1. 由代码可以双重循环共执行 n(n-1)/2次,所以为O(n^2)

空间复杂度：

1. 由代码开辟了一个长度为n的数组用于存储最大值,所以空间复杂度为：O(n)

算法优化：

考虑最终只需输出长度,可以维护一个递增的数组B[]
将A[1]=5放入B里,B[1]=5,当只有一个数字5的时候,长度为1的LIS的最小末尾是5
将A[2]=1放入B里,注意这时LIS有两个(5)和(1),长度为1的LIS的最小末尾就应该是1了,所以在B中用1替换5,B[1]=1
将A[3]=6放入B里,这时LIS有(5),(1),(6),(5,6),(1,6),长度为2的LIS的最小末尾是6,即B[2]=6
将A[4]=8放入B里,因为8>6,长度加1,长度为3的LIS的最小末尾是8,即B[3]=8
将A[5]=2放入B里,这时2在1,6之间,(1,2)可以组成长度为2的LIS的最小末尾为2,所有用2替换6,即B[2]=2,B={1,2,8}
将A[6]=4放入B里,同A[5],得B[3]=4,B={1,2,4}
将A[7]=5放入B里，因为5>4,长度为4的LIS最小末尾是5，B[4]=5,B={1,2,4,5}
将A[8]=10放入B里,10>5,长度为5的LIS最小末尾是10,B[5]=10,B={1,2,4,5,10}
最后B数组的长度即为最长递增子序列的长度5

在往B数组里查找新元素的位置时，由于B数组一直是有序的，可以采用二分查找来优化查找速度

Python代码实现优化：

```
# -*- coding: utf-8 -*-

def BSearch(array,res,item):
#二分查找
    end = res
    start = 0
    while (start <= end):
        mid = start + (end-
start)/2
        if array[mid] > item:
            end = mid -1
        elif array[mid] < item:
            start = mid + 1
        else:
            return mid
#如果找到元素直接返回
    return start
#如果不存在，返回替换元
素的位置

def LIS(array):
    res = 1
    d = [0]*len(array)
    d[0] = array[0]
#将第一个元素放入临时数
组
    for i in array:
        if i > d[res-1]:
#如果大于B中最大的，直
接插入末尾
            d[res] = i
            res = res + 1
        else:
            pos =
BSearch(d,res,i)
#如果小于，二分查找位置
            d[pos] = i

    #print d
    return res

if __name__ == '__main__':
    A = [5,1,6,8,2,4,5,10]
    print LIS(A)
```

时间复杂度：

1. LIS外层：遍历原数组为n
2. 内层为二分查找logn
3. 时间复杂度为：O(nlogn)

空间复杂度：

1. 由代码开辟了一个长度为n的数组用于存储递增序列,所以空间复杂度为：O(n)