1.实现一个带有取最小值min方法的栈，min方法将返回当前栈中的最小值。

你实现的栈将支持**push**，**pop** 和 **min** 操作，所有操作要求都在O(1)时间内完成。

 注意事项

如果堆栈中没有数字则不能进行min方法的调用

您在真实的面试中是否遇到过这个题？

Yes

**样例**

如下操作：**push(1)，pop()，push(2)，push(3)，min()， push(1)，min()** 返回 **1，2，1**

思路：push代表入栈相当于append，pop就是pop，min直接取吧

class MinStack(object):

def \_\_init\_\_(self):

# do some intialize if necessary

self.items=[]

def push(self, number):

# write yout code here

return self.items.append(number)

def pop(self):

# pop and return the top item in stack

return self.items.pop()

def min(self):

# return the minimum number in stack

return min(self.items)

2. 对于一个给定的 source 字符串和一个 target 字符串，你应该在 source 字符串中找出 target 字符串出现的第一个位置(从0开始)。如果不存在，则返回 -1。

您在真实的面试中是否遇到过这个题？

Yes

**说明**

在面试中我是否需要实现KMP算法？

* 不需要，当这种问题出现在面试中时，面试官很可能只是想要测试一下你的基础应用能力。当然你需要先跟面试官确认清楚要怎么实现这个题。

**样例**

如果 source = "source" 和 target = "target"，返回 -1。

如果 source = "abcdabcdefg" 和 target = "bcd"，返回 1。

思路：我要判断单词是不是在目标字符串里，这个容易，下一个是怎么确定第一个，（假设存在，）没办法要想证明存在就意味着我取相同的长度截下，看是否相等，当然需要找到每个首字母出现的位置，这意味着要遍历。

3. 给定一个排序的整数数组（升序）和一个要查找的整数target，用O(logn)的时间查找到target第一次出现的下标（从0开始），如果target不存在于数组中，返回-1。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

在数组 [1, 2, 3, 3, 4, 5, 10] 中二分查找3，返回2。

思路：二分查找按说不难，而且还是排好序的，但是要找第一个可能费事点，因为二分法没办法确定是否是第一个，所以需要对找到的数进行判断。就是挨个前移。

二分出口就是前面序号比后面的序号要大

class Solution:

# @param nums: The integer array

# @param target: Target number to find

# @return the first position of target in nums, position start from 0

def binarySearch(self, nums, target):

# write your code here

length=len(nums)

start=0

end=length-1

t=-2

while(start<=end):

mid=(start+end)//2

if nums[mid]>target:

end=mid-1

elif nums[mid]<target:

start=mid+1

else:

t=mid

break

if t==-2:

return -1

while nums[t]==target:

if t>0:

t-=1

else:

return 0

return t+1

4．给定一个数字列表，返回其所有可能的排列。

 注意事项

你可以假设没有重复数字。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出一个列表[1,2,3]，其全排列为：

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

思路：全排列一般都是循环递归的概念。因为递归能做深度，循环能做广度。

怎么做所谓全排列就是交换次序，我可以递归做深度。循环做交换次数。

mp=[]

length=len(nums)

if length==0 or length==1:

return [nums]

else：

self.digui(nums,0,mp)

def digui(self,nums,index,mp):

if index>len(nums)-1:

b.append(tuple(nums))

return

for i in range(index,len(nums),1):

nums[i],nums[index]=nums[index],nums[i]

digui(nums,index+1,mp)

nums[i],nums[index]=nums[index],nums[i]

5. 给出一个具有重复数字的列表，找出列表所有不同的排列。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出列表 [1,2,2]，不同的排列有：

[

[1,2,2],

[2,1,2],

[2,2,1]

]

思路：直接排得了，要是重复就不加入就可以。或许我可以从源头阻止，比如相等就什么都不做，还是不行最后还要比较，得来别折腾了，直接排，排了在判断是否重复。

class Solution:

"""

@param nums: A list of integers.

@return: A list of unique permutations.

"""

def permuteUnique(self, nums):

# write your code here

if len(nums)==0:

return [[]]

mp=[]

digui(nums,0,mp)

return mp

def digui(nums, index, mp):

if index>=len(nums)-1:

if tuple(nums) not in mp:

mp.append(tuple(nums))

return

for i in range(index,len(nums), 1):

nums[i],nums[index]=nums[index],nums[i]

digui(nums, index + 1, mp)

nums[i],nums[index]=nums[index],nums[i]

6．给定一个含不同整数的集合，返回其所有的子集

 注意事项

子集中的元素排列必须是非降序的，解集必须不包含重复的子集

您在真实的面试中是否遇到过这个题？

Yes

**样例**

如果 S = [1,2,3]，有如下的解：

[

[3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[]

]

思路：我们平时都是怎么把取子集的，计算机怎么知道我怎么取那个没取，那就是做标记，要取的就标记1，我要做的就是把1做的是不重复的，那就是递归喽，因为只有能做深度，而且能保证到取值。即每个位置都有可能是0和一，因此递归时就有两种情况那就循环就可以。class Solution:

"""

@param S: The set of numbers.

@return: A list of lists. See example.

"""

def subsets(self, S):

# write your code here

b=[]

flag=[0 for i in range(len(S))]

digui(0,b,S,flag)

return b

def digui(n,b,S,flag):

if n==len(S):

temp=[]

for i in range(len(S)):

if flag[i]==1:

temp.append(S[i])

temp.sort()

b.append(temp)

return

for flag[n] in [0,1]:

digui(n+1,b,S,flag)

7. 给定一个可能具有重复数字的列表，返回其所有可能的子集

 注意事项

* 子集中的每个元素都是非降序的
* 两个子集间的顺序是无关紧要的
* 解集中不能包含重复子集

您在真实的面试中是否遇到过这个题？

Yes

**样例**

如果 S = [1,2,2]，一个可能的答案为：

[

[2],

[1],

[1,2,2],

[2,2],

[1,2],

[]

]

思路:感觉不到与上一题的区别唯一区别就是判断是否在里面，估计应该会有更好的解法，但现在不知道。

class Solution:

"""

@param S: A set of numbers.

@return: A list of lists. All valid subsets.

"""

def subsetsWithDup(self, S):

# write your code here

b=[]

flag=[0 for i in range(len(S))]

digui(0,b,S,flag)

return b

def digui(n,b,S,flag):

if n==len(S):

temp=[]

for i in range(len(S)):

if flag[i]==1:

temp.append(S[i])

temp.sort()

if temp not in b:

b.append(temp)

return

flag[n]=0

digui(n+1,b,S,flag)

flag[n]=1

digui(n+1,b,S,flag)

8. 写出一个高效的算法来搜索 *m* × *n*矩阵中的值。

这个矩阵具有以下特性：

* 每行中的整数从左到右是排序的。
* 每行的第一个数大于上一行的最后一个整数。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

考虑下列矩阵：

[

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

给出 target = 3，返回 true

如果按照一维的特点那么就是list就是有lg（n\*m）既然是矩阵就必然会有只比较m和n个怎么样就是m和n都在动，一旦不满足可以排除二微的区域。先找m，后找n

def searchMatrix(self, matrix, target):

#递归算法cost 866 ms

#非递归算法cost864ms

# write your code here

if len(matrix)==0:

return False

if target<matrix[0][0]:

return False

if target>matrix[-1][-1]:

return False

m=len(matrix);n=len(matrix[0])

flag,index=self.getm(matrix,0,m,target)

if flag:

start=0

end=n

while 1:

if start>=end:

print(1)

return False

mid=(start+end)/2

if target==matrix[index][mid]:

return True

if target>matrix[index][mid]:

start=mid+1

else:

end=mid

else:

return False

#递归表达会慢一些

'''

def getm(self,matrix,start,end,target):

if start>=end:

return False,-1

mid=(start+end)/2

if target<=matrix[mid][-1] and target>=matrix[mid][0]:

return True,mid

elif target<matrix[mid][0]:

return self.getm(matrix,start,mid,target)

elif target>matrix[mid][-1]:

return self.getm(matrix,mid+1,end,target)

'''

def getm(self,matrix,start,end,target):

while 1:

if start>=end:

return False,-1

mid=(start+end)/2

if target<=matrix[mid][-1] and target>=matrix[mid][0]:

return True,mid

elif target<matrix[mid][0]:

end=mid

elif target>matrix[mid][-1]:

start=mid+1

9. 给出三个字符串:s1、s2、s3，判断s3是否由s1和s2交叉构成。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

比如 s1 =**"aabcc"** s2 =**"dbbca"**

    - 当 s3 = **"aadbbcbcac"**，返回  true.

    - 当 s3 = **"aadbbbaccc"**， 返回 false.

思路：首先我可以判断长度，长度相等的进来，然后pop看看是否相等。最麻烦就是可能时两个的那么就还要往前看是那个。怎么处理这一情况，感觉不好处理想不出来。除非我可以尝试多种情况。是否可以利用递归即深度搜索最后我只要一个or就可以。不如做个二维数组来表示两个字符串的问题，大部分都可以用dp[i][j]表示第一个字符串前i个字符第二个字符串前j个字符的匹配情况来解决，这样代表可以往右和往下走。和我之前想的一样。

1. **class** Solution {
2. **public**:
3. /\*\*
4. \* Determine whether s3 is formed by interleaving of s1 and s2.
5. \* @param s1, s2, s3: As description.
6. \* @return: true of false.
7. \*/
8. **bool** isInterleave(string s1, string s2, string s3) {
9. // write your code here
10. **if**(s3.length()!=s1.length()+s2.length())
11. **return** **false**;
12. **if**(s1.length()==0)
13. **return** s2==s3;
14. **if**(s2.length()==0)
15. **return** s1==s3;
16. vector<vector<**bool**> > dp(s1.length()+1,vector<**bool**>(s2.length()+1,**false**));
17. dp[0][0] = **true**;
18. **for**(**int** i=1;i<=s1.length();i++)
19. dp[i][0] = dp[i-1][0]&&(s3[i-1]==s1[i-1]);
20. **for**(**int** i=1;i<=s2.length();i++)
21. dp[0][i] = dp[0][i-1]&&(s3[i-1]==s2[i-1]);
22. **for**(**int** i=1;i<=s1.length();i++)
23. {
24. **for**(**int** j=1;j<=s2.length();j++)
25. {
26. **int** t = i+j;
27. **if**(s1[i-1]==s3[t-1])
28. dp[i][j] = dp[i][j]||dp[i-1][j];
29. **if**(s2[j-1]==s3[t-1])
30. dp[i][j] = dp[i][j]||dp[i][j-1];
31. }
32. }
33. **return** dp[s1.length()][s2.length()];
34. }
35. };

10. 给出一个**无重叠的**按照区间起始端点排序的区间列表。

在列表中插入一个新的区间，你要确保列表中的区间仍然有序且**不重叠**（如果有必要的话，可以合并区间）。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

插入区间**[2, 5]** 到 **[[1,2], [5,9]]**，我们得到 **[[1,9]]**。

插入区间**[3, 4]** 到 **[[1,2], [5,9]]**，我们得到**[[1,2], [3,4], [5,9]]**。

思路区间位置就哪几种遍历吧，但前面小于后面的大于插入，当前面等于，后面也等于，合并，后面小于，合并，大于合并。没了。

"""

Definition of Interval.

class Interval(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

"""

class Solution:

"""

Insert a new interval into a sorted non-overlapping interval list.

@param intevals: Sorted non-overlapping interval list

@param newInterval: The new interval.

@return: A new sorted non-overlapping interval list with the new interval.

"""

def insert(self, intervals, newInterval):

result= []

# write your code here

#709ms

length=len(intervals)

if length==0:

return [newInterval]

for i in range(length):

flag=0

if intervals[i].end<newInterval.start:

flag=1

result.append( intervals[i])

elif intervals[i].start>newInterval.end:

flag=2

result.append(newInterval)

newInterval=intervals[i]

else:

flag=3

newInterval.start=min(newInterval.start,intervals[i].start)

newInterval.end=max(newInterval.end,intervals[i].end)

if flag==1 or flag==3:

result.append(newInterval)

elif flag==2:

result.append( intervals[-1])

return result