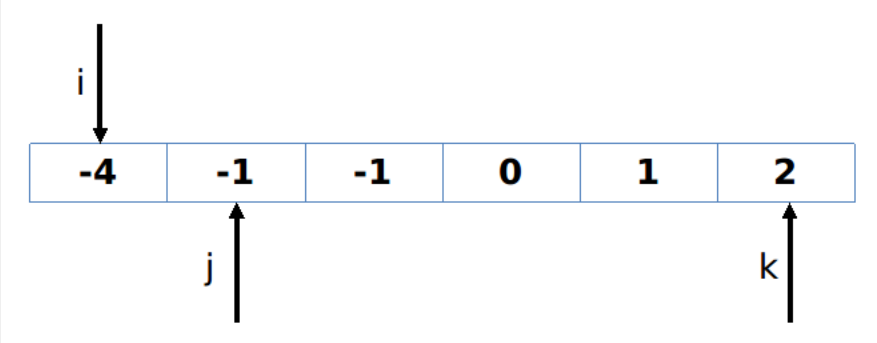
3Sum:

假设数组为 [-1, 0, 1, 2, -1, -4]
首先，把数组排序，复杂度为O(logn)。不先排序的话非常麻烦。同理，想象由三个指针，初始状态是指向如下图所示



Step1， 先保持 i 恒定，开始计算 j 和 k：
1）当 ai + aj + ak = 0时， 把ai, aj, ak, 作为结果之一保存起来。指针 i 和 k 同时向中间移动一个单位。
2）当 ai + aj + ak < 0时， k 保持不动，j 向右移动一个单位。为什么这么做呢？ 我们希望找到ai + aj + ak = 0的组合，而当前状态是ai + aj + ak < 0， 又由于指针 i 恒定不动，只能移动 j 和 k 来寻找，也就是说只有 aj 和 ak 可变。由于数组是升序的，因此 aj < ak, 因此要从小于零变成等于零，只能增加 aj, 因为ak已经是最大的了不能再大了，而要是k往左移动一个的话只能让ai + aj + ak的和更小，更小于零了。因此只能是 把 j 指针向中间移动一个单位，从而曾大ai, aj, ak的和。
3) 移动完成之后继续进行判断，重复第一步。
4) 当 ai + aj + ak > 0，j 保持不动， k 往中间移动一个单元，从而让ai, aj, ak三个数的和更小一点，接着尝试是否能等于零。
5) 再次重复以上几步，一直到指针 i 和 k 在数组中间某个位置相遇位置。可以用while loop实现 while j<k ...
Step2, 把指针 i 往右移动一个单元，重复 Step1中所有步骤。一直到指针 i 移动到倒数第三个元素为止。这是因为当指针 i 在倒数第三个元素的时候， j 和 k 就分别指向最后两个元素了。

Step3, 这里头设计到一个advoid duplicate的问题，因此还要加上一步去重复。先考虑重复是如何出现的，留意数组中由两个 -1. 因此i 会扫过两个 -1 ，因此产生两组 相同的 ai, aj, ak. (-1, 0, 2)对于i=1, 另一组(-1, 0, 2)对应 i = 2.
如何避免这种重复？ 只需要想办法skip掉第一个-1即可，或者更笼统地说让 j 指针只停留在最后一个-1, 而跳过前面所有的 -1.
加入语句 if (i==0 || num[i] != num[i-1]), 如果 i 指向第一个元素，或者 a(i) != a (i-1) 那么就指向上面所有的步骤，否则不做任何判断计算，直接把 i 指针向右移动一格。

完整代码如下：

```python
def ThreeSum(nums):
    nums.sort()
    n = len(nums)
    res = set()

    if n < 3:
        return None

    for i in range(n-2):

        if i == 0 or nums[i] != nums[i-1]:
            j = i + 1
            k = n - 1

            while j < k :
                sum = nums[i] + nums[j] + nums[k]
                if sum == 0:
                    res.add( (nums[i] , nums[j] , nums[k]) )
                    j += 1
                    k -= 1
                elif sum < 0:
                    j+=1
                else:
                    k -=1
    return list(res)
```

题外：

在 {a, b, c, d, e} 这个字符数组中选取三个，列出所有可能的选法， 显然总共有C(3, 5)=10 种选法。

分别为

1） 以a为首字符；

1.1 以 e 为尾字符；

b, c, d 为中间字符，总共由三种选法

(a, b, e) (a, c, e) (a, d, e)

1.2 以 d 为尾字符：

b, c 为中间字符， 总共由两种选法

(a, b, d) (a, c, d)

1.3 以 c 为尾字符

b为中间字符, 总共由一种选法

(a, b, c)

2) 以 b 为首字符：

2.1 以 e 为尾字符：

c, d 为中间字符， 共两种选法

(b, c, e) (b, d, e)

2.2 以 d 为尾字符：

c 为中间字符， 共一种选法

(b, c, d)

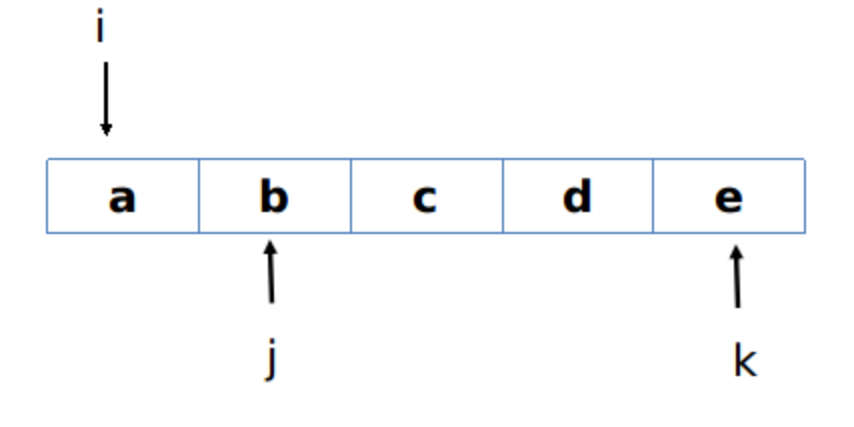
3) 以 c 为首字符：

3.1 以 e 为尾字符：

只能d 为中间字符， 共一种选法

(c, d, e)

至此，我们找到了所有的10种组合方式。如果仔细分析，以上我们在做选择的方法可以有规律可循，想象我们有三个指针，一开始只想第一个，第二个，和最后一个元素，如下图表示：



1）指针 i 恒定，指针 k 恒定，指针j 扫过中间所有元素。(a 为首字符， e为尾字符)

2）指针 i 恒定，指针 k 向左移动一格，指针 j 继续扫描中间的元素 （a为首字符，d为尾字符）

3) 指针 i 继续恒定，重复第二步，知道指针 k 移动到 a[2] 的位置为止。(a 为首字符，c为尾字符)

4）指针 i 向前移动一格，j停在 i + 1位置， k停在最后的位置 ，重复步骤(1) 和步骤 (2)，直到指针 i 移动到 N-2 的位置位置 (b为首字符...)

这样一直到指针 i 扫到第 N-2 个元素为止， 就找到了所有的组合数。这就是夹逼法，其复杂度为 O(n^3). 代码：

```python
def comb(strs):

    n = len(strs)
    res=set()

    for i in range(0, n-2):
        for k in range(n-1, 1, -1):
            for j in range(i+1, k):
                tup = ( strs[i], strs[j], strs[k] )
                res.add(tup)
    return res

strs=['a', 'b', 'c', 'd', 'e', 'f']
res = comb(strs)
print(res)
print (len(res))
```

这里假设元素没有重复。

****

上面的问题是三循环嵌套，现在回到这条题目，我们用类似的方法，但可以简化二循环把复杂度降为O(n^2).