一：  
递归深度为 ，每层需要 的时间，因此总的时间复杂度为 。  
二：

array sorted in ascending order, target, lo, hi index of the target if found, -1 if not found **return** **return** **return** **return**

上述算法采取二分查找的策略，每次查找比较数组中间的元素， 如果中间元素大于目标值，则目标值在数组的前半部分，对前半部分进行递归调用， 否则在后半部分， 对后半部分进行递归调用。

该算法的时间复杂度的递归方程为

由 Master定理， , 故 , 即此算法的时间复杂度为 。  
三：  
1.在 中找出其中位数 。  
2.将 中每个数与 作差取绝对值，得到数组 。  
3.在 中找出第 个小的数 。  
4.遍历 d， 找出小于等于 对应的下标 ， 则所有的 构成最接近 的 个数。  
伪代码如下：

, numbers in that are closest to the median of **return**

下面是 算法的伪代码，实现在 的时间复杂度下找出一组数中第 个小的数。

, the smallest number in **return** **return** **return**

下面是 算法的伪代码，实现在 的时间复杂度下将数组进行划分。

, the index of **return**

四：  
1.选取两组数的第 个数k1、k2进行比较。  
2.若k1小于k2，则将第一组数前 个数排除，重新计算 , 在剩余数中寻找。  
3.否则排除第二组数的前 个数排除，重新计算 , 在剩余数中寻找。  
4.直到 ，此时返回两组数中当前索引下的更小的一个数，即为第 大的数。

, , **return** **return** **return**

上述算法实现了在 的时间复杂度下找出两组升序排列的数中的第 大的数， 若两组数总长度为偶数，则使用上述算法找出第 大和第 大的数并取两者的平均值；若两组数总长度为奇数，则使用上述算法找出第 大的数即可。 时间复杂度均为

五：  
1.遍历数组，统计每个数字出现的次数，存入字典。  
2.使用快速排序对字典的值进行排序，得到一个递减的序列。  
3.选出递减序列前k个键值对的键。  
时间复杂度为

, top k numbers **return**