

一:

递归深度为 $\log n$ ，每层需要 n 的时间，因此总的时间复杂度为 $O(n \log n)$ 。

二:

Algorithm 1 `binary_search`

Input: array sorted in ascending order, target, lo, hi

Output: index of the target if found, -1 if not found

```
1:  $mid = (lo + hi) // 2$ 
2: while  $lo \leq hi$  do
3:   if  $arr[mid] == target$  then
4:     return  $mid$ 
5:   else if  $arr[mid] < target$  then
6:     return  $binary\_search(arr, target, mid + 1, hi)$ 
7:   else
8:     return  $binary\_search(arr, target, lo, mid - 1)$ 
9:   end if
10: end while
11: return -1
```

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

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

由 Master 定理， $1 = \Theta(n^{\log_2 1}) = \Theta(1)$ ，故 $T(n) = \Theta(n^{\log_2 1} \log n) = \Theta(\log n)$ ，即此算法的时间复杂度为 $\Theta(\log n)$ 。

三:

1. 在 S 中找出其中位数 $median$ 。
2. 将 S 中每个数与 $median$ 作差取绝对值, 得到数组 d 。
3. 在 d 中找出第 k 个小的数 $kmin$ 。
4. 遍历 d , 找出小于等于 $kmin$ 对应的下标 i , 则所有的 $S[i]$ 构成最接近 $median$ 的 k 个数。

伪代码如下:

Algorithm 2 Search for k numbers closest to the median of S

Input: S, k

Output: k numbers in S that are closest to the median of S

```
1:  $n = \text{len}(S)$ 
2:  $median = \text{Select}(S, n/2)$ 
3:  $d = []$ 
4: for  $i = 0$  to  $n - 1$  do
5:    $d[i] = \text{abs}(S[i] - median)$ 
6: end for
7:  $kmin = \text{Select}(d, k)$ 
8:  $\text{return}S = []$ 
9: for  $i = 0$  to  $n - 1$  do
10:  if  $d[i] \leq kmin$  then
11:     $\text{return}S.append(S[i])$ 
12:  end if
13: end for
14: return  $\text{return}S$ 
```

下面是 *Select* 算法的伪代码，实现在 $O(n)$ 的时间复杂度下找出一组数中第 k 个小的数。

Algorithm 3 find the k -th smallest number in arr

Input: arr, k

Output: the $k - th$ smallest number in arr

```
1:  $n = \text{len}(arr)$ 
2: for  $i = 0$  to  $n/5$  do
3:    $\text{insert\_sort}(arr[(i - 1) * 5 + 1 : (i - 1) * 5 + 5])$ 
4:    $\text{swap}(arr[i], arr[(i - 1) * 5 + 3])$ 
5: end for
6:  $x = \text{Select}(arr[: n/5], n/10)$ 
7:  $\text{index} = \text{partition}(arr, x)$ 
8: if  $k == \text{index}$  then
9:   return  $x$ 
10: else if  $k < \text{index}$  then
11:   return  $\text{Select}(arr[: \text{index} - 1], k)$ 
12: else
13:   return  $\text{Select}(arr[\text{index} + 1 :], k - \text{index})$ 
14: end if
```

下面是 *partition* 算法的伪代码，实现在 $O(n)$ 的时间复杂度下将数组进行划分。

Algorithm 4 partition the arr

Input: arr, x

Output: the index of x

```
1:  $n = \text{len}(arr)$ 
2:  $i = 0$ 
3: for  $j = 1$  to  $n - 1$  do
4:   if  $arr[j] \leq x$  then
5:      $i = i + 1$ 
6:      $\text{swap}(arr[i], arr[j])$ 
7:   end if
8: end for
9: return  $i$ 
```

四:

1. 选取两组数的第 $k/2$ 个数 k_1 、 k_2 进行比较。
2. 若 k_1 小于 k_2 , 则将第一组数前 $k/2$ 个数排除, 重新计算 k , 在剩余数中寻找。
3. 否则排除第二组数的前 $k/2$ 个数排除, 重新计算 k , 在剩余数中寻找。
4. 直到 $k = 1$, 此时返回两组数中当前索引下的更小的一个数, 即为第 $k/2$ 大的数。

Algorithm 5 search for two arrays' k -th number

Input: $nums1, nums2, k$

Output: the median of the two arrays

```
1:  $length1 = \text{len}(nums1)$ 
2:  $length2 = \text{len}(nums2)$ 
3:  $index1 = index2 = 0$ 
4: while  $true$  do
5:   if  $index1 == index2$  then
6:     return  $nums2[index2 + k - 1]$ 
7:   end if
8:   if  $index2 == length2$  then
9:     return  $nums1[index1 + k - 1]$ 
10:  end if
11:  if  $k == 1$  then
12:    return  $\min(nums1[index1], nums2[index2])$ 
13:  end if
14:   $new\_index1 = \min(index1 + k/2, length1) - 1$ 
15:   $new\_index2 = \min(index2 + k/2, length2) - 1$ 
16:  if  $nums1[new\_index1] \leq nums2[new\_index2]$  then
17:     $k = k - (new\_index1 - index1 + 1)$ 
18:     $index1 = new\_index1 + 1$ 
19:  else
20:     $k = k - (new\_index2 - index2 + 1)$ 
21:     $index2 = new\_index2 + 1$ 
22:  end if
23: end while
```

上述算法实现了在 $O(\log(m+n))$ 的时间复杂度下找出两组升序排列的数中的第 k 大的数, 若两组数总长度为偶数, 则使用上述算法找出第 $(m+n)/2$ 大和第 $(m+n)/2+1$ 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 $(m+n)/2$ 大的数即可。时间复杂度均为 $O(\log(m+n))$

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 $O(n \log n)$

Algorithm 6 top k numbers

Input: $nums, k$

Output: top k numbers

```
1:  $map = \{\}$ 
2:  $n = \text{len}(nums)$ 
3: for  $i = 0$  to  $n - 1$  do
4:    $key = nums[i]$ 
5:    $map[key] = map[key] + 1$ 
6: end for
7:  $sorted\_map = \text{quick\_sort}(map, map.items())$ 
8:  $return\_nums = []$ 
9: for  $i = 0$  to  $n - 1$  do
10:  if  $i \leq k$  then
11:     $return\_nums.append(sorted\_map[i][0])$ 
12:  end if
13: end for
14: return  $return\_nums$ 
```
