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

$mid = (lo + hi)//2$
while $lo \leq hi$ **do**
  if $arr[mid] == target$ **then**
    return mid
  else if $arr[mid] < target$ **then**
    return $binary\_search(arr, target, mid + 1, hi)$
  else
    return $binary\_search(arr, target, lo, mid - 1)$
  end if
end while
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

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

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

```
n = len(S)
median = Select(S, n//2)
d = [ ]
for i = 0 to n − 1 do
    d[i] = abs(S[i] − median)
end for
kmin = Select(d, k)
returnS = [ ]
for i = 0 to n − 1 do
    if d[i] ≤ kmin then
        returnS.append(S[i])
    end if
end for
return returnS
```

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

$n = len(arr)$
for $i = 0$ to $n/5$ **do**
    $insert\_sort(arr[(i-1)*5+1:\ (i-1)*5+5])$
    $swap(arr[i], arr[(i-1)*5+3])$
end for
$x = Select(arr[:n//5], n/10)$
$index = partition(arr, x)$
if $k == index$ **then**
    return x
else if $k < index$ **then**
    return $Select(arr[:index-1], k)$
else
    return $Select(arr[index+1:], k-index)$
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

$n = len(arr)$
$i = 0$
for $j = 1$ to $n - 1$ **do**
    if $arr[j] \le x$ **then**
        $i = i + 1$
        $swap(arr[i], arr[j])$
    end if
end for
return i

四：
1. 选取两组数的第 $k/2$ 个数 k1、k2 进行比较。
2. 若 k1 小于 k2，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*

```
length1 = len(nums1)
length2 = len(nums2)
index1 = index2 = 0
while true do
    if index1 == index2 then
        return nums2[index2 + k − 1]
    end if
    if index2 == length2 then
        return nums1[index1 + k − 1]
    end if
    if k == 1 then
        return min(nums1[index1], nums2[index2])
    end if
    new_index1 = min(index1 + k/2, length1) − 1
    new_index2 = min(index2 + k/2, length2) − 1
    if nums1[new_index1] ≤ nums2[new_index2] then
        k = k − (new_index1 − index1 + 1)
        index1 = new_index1 + 1
    else
        k = k − (new_index2 − index2 + 1)
        index2 = new_index2 + 1
    end if
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

$map = \{\}$
$n = len(nums)$
for $i = 0$ to $n - 1$ **do**
  $key = nums[i]$
  $map[key] = map[key] + 1$
end for
$sorted\_map = quick\_sort(map,\ map.items())$
$return\_nums = [\,]$
for $i = 0$ to $n - 1$ **do**
  if $i \leq k$ **then**
    $return\_nums.append(sorted\_map[i][0])$
  end if
end for
return $return\_nums$