合：100

计算机学院《算法设计与分析》第一次作业

计算机学院　20373673　于敬凯

September 25, 2022

1　请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明，可以假定 n 是 2 的整数次幂

21

1.1

答案： $O(n)$

证明：

情况一：n 为奇数

以下证明当 $n=2k-1, k\in Z^+$ 即 n 为奇数时，$T(n)=\frac{n+1}{2}$
由数学归纳法：
1. $n=1$ 时，$T(n)=1=\frac{1+1}{2}$
2. 假设 $n>1$ 即 $n\geq 3$ 时，$T(n)=\frac{n+1}{2}$
3. 对于 $n+2$，则 $T(n+2)=T(n)+1=\frac{n+1}{2}+1=\frac{(n+2)+1}{2}$

情况二：n 为偶数

以下证明当 $n=2k, k\in Z^+$ 即 n 为偶数时，$T(n)=\frac{n}{2}$
由数学归纳法：
1. $n=2$ 时，$T(n)=1=\frac{2}{2}$
2. 假设 $n>2$ 即 $n\geq 4$ 时，$T(n)=\frac{n}{2}$
3. 对于 $n+2$，则 $T(n+2)=T(n)+1=\frac{n}{2}+1=\frac{(n+2)}{2}$

综上，当 n 为奇数时 $T(n)=\frac{n+1}{2}$，当 n 为偶数时，$T(n)=\frac{n}{2}$，因此，$T(n)$ 的尽可能紧凑上界是 $O(n)$

$$T(n)=T(n-2)+3n=T(n-4)+3n+3(n-2)=T(n-6)+3n+3(n-2)+3(n-4)=...$$
$$=3n+3(n-2)+3(n-4)+...+9+1=\frac{1}{2}(3n+9)(\frac{3n-9}{6}+1)+1=O(n^2)$$

由此可知，原式渐进上界为 $O(n^2)$。

1.2

答案： $O(log\ n)$

证明： 由主定理中递归式 $T(n)=aT(\frac{n}{b})+f(n)$ 知，本题中 $a=1, b=2, f(n)=1$，因此 $log_b a=log_2 1=0$，因此 $f(n)=1=\Theta(nlog_b a)$，因此由主定理知 $T(n)=\Theta(n^{log_b a}log\ n)=\Theta(log\ n)$。因此紧凑渐进上界为 $log\ n$。

1.3

答案： $O(n)$

证明： 由主定理中递归式的简化形式 $T(n)=aT(\frac{n}{b})+n^k$ 知，本题中 $a=1,b=2,f(n)=n,k=1$ ，因此 $log_ba=log_21=0,k=1>log_ba$。因此判断知 $T(n)=\Theta(n^1)=O(n)$。

1.4

答案： $O(n)$

证明： 由主定理中递归式 $T(n)=aT(\frac{n}{b})+f(n)$ 知，本题中 $a=2,b=2,f(n)=1$ ，因此 $log_ba=log_22=1$ ，存在正常数 $\varepsilon=\frac{1}{2}$ 使得 $f(n)=O(n^{log_ba-\varepsilon}=O^{\frac{1}{2}})$ ，因此 $T(n)=\Theta(n^{log_ba})=O(n)$

1.5

答案： $O(n^2)$

证明： 由主定理中递归式 $T(n)=aT(\frac{n}{b})+f(n)$ 知，本题中 $a=4,b=2,f(n)=1$ ，因此 $log_ba=log_24=2$ ，存在正常数 $\varepsilon=1$ 使得 $f(n)=1=O(n^{log_ba-\varepsilon})=O(n)$ ，因此 $T(n)=\Theta(n^{log_ba})=O(n^2)$

1.6

答案： $O(log^2n)$

证明： 当 $n=2^k$ 时，$T(n)=T(\frac{n}{2})+log\ n$。

令 $c_k=n=2^k$，$T(c_k)=T(c_{k-1})+log(c_k)$。

令 $S(k)=T(c_k)$，因此 $S(k)=S(k-1)+log(2^k)=S(k-1)+k$，因此 $S(k)-S(k-1)=k$，由等差数列求和得 $S(m)-S(0)=\sum_{i=1}^{k}i$ ，因此 $S(m)=1+\frac{(k+1)(k)}{2}$，因此代回原表达式知 $T(2^k)=1+\frac{(k+1)(k)}{2}$，因此 $T(n)=1+\frac{(log\ n+1)(log\ n)}{2}=O(log^2n)$。

1.7

答案： $O(n^2)$

证明： 由主定理中递归式的简化形式 $T(n)=aT(\frac{n}{b})+n^k$ 知，本题中 $a=3,b=2,k=2$ ，因此 $log_ba=log_23$。由 $k=2>log_ba=log_23$ 知 $T(n)=\Theta(n^k)=O(n^2)$

$$T(n)=\begin{cases}1\ ,\ n=1\\ 8T(n/4)+2n\ ,\ n>1\end{cases}$$

使用主方法计算渐进上界，注意到 $2n=O(n)$，由上式可知 $a=8,b=4,d=1$，由 $d<log_ba$，可得上式的渐进上界为 $O(n^{log_ba})=O(n^{\frac{3}{2}})$。

2 寻找中位数问题

题面： 一组数据 $x=(x_1,\cdots,x_n)$ 从小到大排序后的序列为 x'_1，则这组数据的中位数 M 为

$$M=\begin{cases}x'_{\frac{n+1}{2}}\ ,\ 若n为奇数\\ \frac{1}{2}(x'_{\frac{n}{2}}+x'_{\frac{n}{2}+1})\ ,\ 若n为偶数\end{cases}$$

给定两个长度分别为 n,m 的有序数组 $A[1\ldots n],B[1\ldots m]$ （A 与 B 均已按从小到大排序）。请设计尽可能高效的算法，求出这两个数组中所有数据的中位数，并分析其时间复杂度。

2.1 作业题目 2 分析 19

根据题目中对中位数的定义，题目所求即为 A, B 数组所有数据中第 k 小的数，其中 k 为：

$$k = \begin{cases} \frac{n+m+1}{2}\text{，若}n+m\text{为奇数} \\ \frac{n+m}{2}\text{和}\frac{n+m}{2}+1\text{，若}n+m\text{为偶数} \end{cases}$$

因此，问题转化为求第 k 小的数。

由数组的单调性，联想二分和分治思想，考虑如下子情况：

对于任意两个单调递增数组 A_i, B_i 长度分别为 n_i, m_i，求其中第 k_i 小的数，可以分别取 A_i, B_i 中 $A_i[\lfloor \frac{k_i}{2} \rfloor]$ 和 $B_i[\lfloor \frac{k_i}{2} \rfloor]$ 的值并比较两者大小，如果 $A_i[\lfloor \frac{k_i}{2} \rfloor] < B_i[\lfloor \frac{k_i}{2} \rfloor]$，则说明 A_i 从 1 到 $\lfloor \frac{k_i}{2} \rfloor$ 的值都属于所有数据中最小的 k_i 部分，因此，相当于排除了这部分值，接下来更新数组 $A_{i'} = A_i[\lfloor \frac{k_i}{2} \rfloor + 1 \ldots n_i]$，更新数组 $B_{i'} = B_i$，长度分别为 $n_{i'} = n_i - \lfloor \frac{k_i}{2} \rfloor, m_{i'} = m_i$，目标 $k_{i'} = k_i - \lfloor \frac{k_i}{2} \rfloor$，得到类似本段初始的子情况。同理，我们可以类似地处理 $A_i[\lfloor \frac{k_i}{2} \rfloor] > B_i[\lfloor \frac{k_i}{2} \rfloor]$ 的情况。

对于一些特殊情况和边界情况，我们特殊处理：

1. $k = 1$ 时，直接判断 $A[i]$ 和 $B[j]$ 大小并返回小的。
2. i 或 j 越界时，说明其对应的数组已经遍历结束，直接返回另一数组偏移后的结果。
3. $n + m$ 为偶数时，令 $k = \frac{n+m}{2}$，返回的时候直接取第 k 和第 $k + 1$ 小的数的平均数。

2.2 作业题目 2 时间复杂度分析

可以发现，每轮循环中对于子情况的处理，使得需要查找的问题规模 k 减小了至少一半。

$$T(n+m) = \begin{cases} 1, (n+m) = 1 \\ T(\frac{(m+n)}{2}) + 1, (n+m) > 1 \end{cases}$$

由主定理解得时间复杂度为 $O(log(m+n))$

2.3 作业题目 2 算法与伪代码

Algorithm 1: 寻找中位数

Input: 正整数 n, m 为数组 $A[], B[]$ 长度，长度为 n 的数组 $A[1 \ldots n]$ 和长度为 m 的数组 $B[1 \ldots m]$

Output: 一个浮点数，为两个数组 $A[], B[]$ 中所有数据的中位数

```
i, j ← 1, 1;
// 初始化两个指向A, B数组下标的变量
if (n + m)%2 == 0 then
    k = (n + m)/2;
else
    k = (n + m + 1)/2;
end
return getKthMin(A, B, i, j, k);
```

Algorithm 2: 寻找第 k 小的数

Input: 正整数 n, m 为数组 $A[], B[]$ 长度，长度为 n 的数组 $A[1 \dots n]$ 和长度为 m 的数组 $B[1 \dots m]$，整数 i, j 分别为 $A[], B[]$ 数组的搜索起始下标

Output: 一个浮点数，为两个数组 $A[i \dots n], B[j \dots m]$ 中第 k 小的数（当 $n+m$ 为偶数时为第 k 和第 $k+1$ 小的数的平均数）

function *getKthMin(A[], B[], i, j, k)*:
  $flag \leftarrow (n+m)\%2 = 0$;
  if $i > n$ **then** /* 已遍历A数组，答案在B数组当前下标j开始偏移 */
    return $flag$? $\frac{B[j+k-1]+B[j+k]}{2}$: $B[j+k-1]$;
  end
  if $j > m$ **then** /* 已遍历B数组，答案在A数组当前下标i开始偏移 */
    return $flag$? $\frac{A[i+k-1]+A[i+k]}{2}$: $A[i+k-1]$;
  end
  if $k = 1$ **then** /* 当前还剩1位偏移即可得正确答案，且下标i, j均合法 */
    if *not* $flag$ **then**
      return $min\{A[i], B[j]\}$;
    else
      $temp \leftarrow 0$;
      if $A[i] < B[j]$ **then**
        $temp \leftarrow temp + A[i]$;
        $i \leftarrow i + 1$;
      else
        $temp \leftarrow temp + B[i]$;
        $j \leftarrow j + 1$;
      end
      if $i \leq n$ *and* $j \leq m$ **then** /* 防止数组下标越界 */
        $temp \leftarrow temp + min\{A[i], B[j]\}$;
      else if $i \leq n$ **then**
        $temp \leftarrow temp + A[i]$;
      else $temp \leftarrow temp + B[j]$;
      return $tmep/2$;
    end
  end
  $tempi, tempj \leftarrow min\{i + \frac{k}{2} - 1, n\}, \leftarrow min\{j + \frac{k}{2} - 1, m\}$;
  $tempA, tempB \leftarrow A[tempi], B[tempj]$;
  if $tempA \leq tempB$ **then**
    $k \leftarrow k - (tempi - i + 1)$;
    $i \leftarrow tempi + 1$;
  else
    $k \leftarrow k - (tempj - j + 1)$;
    $j \leftarrow tempj + 1$;
  end
  return $getKthMin(A, B, i, j, k)$;
end

3 双调序列最大值问题 20

题面： 若一个序列 $a[1\dots n]$ 满足一下两个性质之一，则称该序列为双调序列：

- 性质 1：存在 $k \in [1,n]$，使得 $a_1 \le a_2 \le \cdots \le a_{k-1} \le a_k \ge a_{k+1} \ge \cdots \ge a_{n-1} \ge a_n$
- 性质 2：序列经过循环移位后满足性质 1

给定一个双调序列 $a[1\dots n]$，请设计一个算法来求出这个序列中的最大值，并对该算法的复杂度进行分析。

3.1 作业题目 3 分析

本题可以很容易想到 $O(n)$ 的朴素做法：顺序遍历即可。但这并没有利用到双调序列给出的特殊性质。下面分析 $O(log\ n)$ 的做法。

双调序列给出了某种有序性，可以联想到分治思想。首先，考虑一个满足题意的双调序列可能出现的四种情况，不难发现一个双调序列一定属于其中之一。

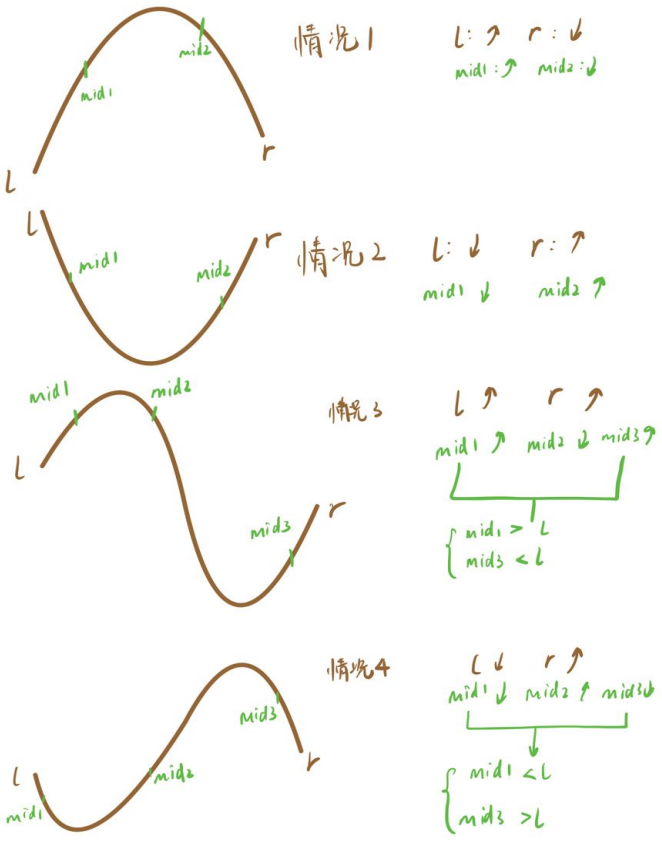

Figure 1: 四种情况示意图

不妨设该序列为 $A[l\dots r]$，取中点下标 $mid = \lfloor \frac{l+r}{2} \rfloor$，通过计算 l, r, mid 及其相邻点的值可计算出该点的增减性趋势（类似导数），通过这三点的增减性可以判断出当前双调序列属于四种情况的哪一种。

对于这四种情况，前两种可以只通过 mid 点的增减性判断答案所在区间；后两种情况中，如第三种情况出现了两类单调递增的点，需要额外通过和左值（或右值）比较判断答案所在区间。

判断出答案所在区间后，我们截取该部分区间，其仍满足双调序列的性质，因此可以递归下去，直至求得正确结果。

3.2 作业题目 3 时间复杂度分析

可以知道，每次执行递归，都减小了一半的搜索区间，类似二分算法：

$$T(n) = \begin{cases} O(1), n = 1 \\ T(\frac{n}{2}) + O(1), n > 1 \end{cases}$$

计算得时间复杂度为 $O(log\ n)$

3.3 作业题目 3 算法与伪代码

Algorithm 3: 双调序列最大值问题

Input: 整数 n 为数组 $a[]$ 长度，长度为 n 的数组 $a[1 \dots n]$

Output: 一个整数，为数组 $a[]$ 中最大值

```
1  l, r ← 1, n;
2  function BitonicSequence(a[], l, r):
3      mid ← ⌊(l+r)/2⌋;
4      l_mono, r_mono ← a[l+1] > a[l] ? UP : DOWN, a[r-i] > a[r] ? DOWN : UP;
5      mid_mono_l ← a[mid] > a[mid-1] ? UP : DOWN;
6      mid_mono_r ← a[mid+1] > a[mid] ? DOWN : UP;
7      switch l_mono, r_mono do
8          case UP, DOWN do
9              if mid_mono_l is UP ansmid_mono_r is DOWN then
10                 return a[mid];
11             else if mid_mono_l is UP and mid_mono_r is UP then
12                 return BitonicSequence(a[], mid+1, r);
13             else
14                 return BitonicSequence(a[], l, mid-1);
15         case DOWN, UP do
16             return max{a[l], a[r]};
17         case UP, UP do
18             if mid_mono_l is DOWN then
19                 return BitonicSequence(a, l, mid-1);
20             else
21                 if a[mid] > a[l] then
22                     return BitonicSequence(a, l, mid-1);
23                 else
24                     return BitonicSequence(a, mid+1, r);
25         case DOWN, DOWN do
26             if mid_mono_l is UP then
27                 return BitonicSequence(a, mid+1, r);
28             else
29                 if a[mid] < a[l] then
30                     return BitonicSequence(a, mid+1, r);
31                 else
32                     return BitonicSequenc(a, l, mid-1);
33     end
34 end
```

4 字符串等价关系判定问题

20

题面： 给定两个长度为 n 的字符串 A 和 B，若称 A 和 B 是等价的，当且仅当它们满足如下关系之一：

1. A 和 B 完全相同
2. 若把 A 分成长度相同的两段 A_1 和 A_2，也将 B 分成长度相等的两端 B_1 和 B_2。且他们之间满足如下两种关系之一：
   - A_1 和 B_1 等价且 A_2 和 B_2 等价；
   - A_1 和 B_2 等价且 A_2 和 B_1 等价。

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。

4.1 作业题目 4 分析

分析题目中给出的两条等价判定关系，通过对递归节及边界条件的设置，实际上约定了题目中所述等价是一种数学意义上的等价关系，即满足自反性、对称性和传递性的关系。

给予对等价的深刻认识，考虑组合数学知识中划分等价类的思想：给一个等价类规定一个唯一的代表元素，同时涉及一套规则，使得对于任意给定的字符串，都可以按照该规则计算出其等价类的代表元素。

基于以上两点想法，考虑利用字典序定义“最小字符串”，具体而言，对于字符串 $dcba$，其等价于 $cdab$，可以证明这是其对应的最小字符串。对于字符串 $cdba$，也可以计算出其最小字符串为 $cdab$，因此判断出二者同属一个等价类，即二者等价。

4.2 作业题目 4 时间复杂度分析

给定一个长度为 n 的字符串，其等价的最小字符串为将其分别分为长度为 $\frac{n}{2}$ 的两个子字符串拼接而成，因此，其时间复杂度满足：

$$T(n) = \begin{cases} 1, n = 1 \\ 2T(\frac{n}{2}) + O(n), n > 1 \end{cases}$$

算得其时间复杂度为 $O(nlog\ n)$

4.3 作业题目 4 算法与伪代码

Algorithm 4: 字符串等价关系判定问题

Input: 长度为 n 的字符串 $A[1 \dots n], B[1 \dots n]$

Output: 一个布尔值，表示两个字符串是否等价

```
function StringEqual(A[], B[]):
    return Small(A) = Small(B)
end
```

Algorithm 5: 计算最小字符串

Input: 长度为 n 的字符串 $S[1\dots n]$

Output: 长度为 n 的字符串 $Small[1\dots n]$

```
function Small(S[]):
    n ← S.length;
    if n = 1 then
        return S;
    else
        S1 ← Small[S[1 ... n/2]];
        S2 ← Small[S[n/2 + 1 ... n]];
        if S1 > S2 then
            return S2 + S1;
        else
            return S1 + S2;
        end
    end
end
```

5 向量的最小和问题

题面： 给定 n 个二位向量 $v_1, v_2, \dots v_n$。每一个向量 $v_i = (x_i, y_i)$ 都可以变换为如下四种形式：

1. $v_i^1 = (x_i, y_i)$
2. $v_i^2 = (-x_i, y_i)$
3. $v_i^3 = (x_i, -y_i)$
4. $v_i^4 = (-x_i, -y_i)$

请你设计一个高效的算法从 n 个向量中找出两个向量，使得他们以某种形式相加后的模长最小。换言之，请找出两个向量 $v_i, v_j (1 \le i, j \le n$ 且 $i \ne j)$，以及两个整数 $k_1, k_2 (1 \le k_1, k_2 \le 4)$，使 $||v_i^{k_1} + v_j^{k_2}||_2$ 最小。此外，请分析该算法的时间复杂度。

5.1 作业题目 5 分析

由于向量可以取 v^1, v^2, v^3, v^4 四种状态，因此问题可以转化为寻找两个向量的最小差，因此可以将所有向量都转化到第一象限进行求解。

在算法开始之前，使用归并排序将点集分别以 x 坐标和 y 坐标按照从小到大预先排序，时间复杂度为 $O(nlog\ n)$。联想分治思想，考虑一个子情况：将传入的所有点以横坐标 x 为第一关键字排序，以 x 为基准，每次将点集分成左右两个大小相等的子集（如果无法整除可以大小相差 1），分别递归求得其最短距离为 d_1, d_2，因此在该子情况中最短距离最坏为 $d = min\{d_1, d_2\}$。考虑两个子集中各有一个点 p_1, p_2 的情况，可以知道如果他们之间的距离要比 d 更优，那么他们与两个点集中间线 mid 的距离最多为 d，因此可以构造 x 范围在 $[mid - d, mid]$ $[mid, mid + d]$ 的带状区域，利用类似归并排序的思想，遍历 n 个点计算出在该带状区域的点的集合，并遍历其中每个点和其后 7 个点的距离 d_i，若有比 d 更小的则更新 d。

5.2 作业题目 5 时间复杂度分析

在点的数量 n 小于等于 3 的时候，直接双层循环搜索，时间复杂度为 $O(1)$；在点的数量 n 大于 3 时，使用分治：

$$T(n) = \begin{cases} O(1), n \leq 3 \\ 2T(\frac{n}{2}) + O(n), n > 3 \end{cases}$$

由主定理计算得，时间复杂度为 $O(nlog\ n)$。

5.3 作业题目 5 算法与伪代码

Algorithm 6: 向量的最小和问题

Input: n 个二维向量 $v_1, v_2, \ldots, v_n$

Output: 满足题意条件的 v_i, v_j, k_1, k_2

$V\_x, V\_y \leftarrow MergeSort(v_1, \ldots, v_n) in\ FIRST.x\ and\ SECOND.y, MergeSort(v_1, \ldots, v_n) in\ FIRST.y\ and\ SECOND.x;$

function *VectorMin(V_x, V_y)*:
    $n \leftarrow V\_x.size$ **if** $n \leq 3$ **then**
        return $min\{dis\{V\_x\_i, V\_x\_j\}\}, v\_i, v\_j;$
    end
    $V\_x\_l, V\_x\_r \leftarrow V\_x\ seperate\ with\ x.value;$
    $V\_y\_l, V\_y\_r \leftarrow \{v\_i | v\_i\ in\ V\_x\_l\}, \{v_j\ in\ V\_x\_r\};$
    $d1, v1, v2 \leftarrow VectorMin(V\_x\_l, V\_y\_l);$
    $d2, v3, v4 \leftarrow VectorMin(V\_x\_r, V\_y\_r);$
    $d \leftarrow min\{d1, d2\};$
    $v1', v2' \leftarrow d1 < d2\ ?\ v1, v2\ :\ v3, v4;$
    $mid \leftarrow Middle(V\_x, V\_y);$
    $V\_new \leftarrow \{v\_i | mid - d \leq v\_i.x \leq mid + d\};$
    for $v\_i\ in\ V\_new$ **do**
        for $j\ in\ [0, \ldots, 7]$ **do**
            if $dis\{v_i, v_{\{i+j\}}\} < d$ **then**
                $d, v1', v2' \leftarrow dis\{v_i, v_{\{}i + j\}\}, v\_i, v\_\{i + j\};$
            end
        end
    end
    return $d, v1', v2';$
end

6 附录暨部分算法的实现

6.1 寻找中位数问题 Java 实现代码

```java
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int len1 = nums1.length;
        int len2 = nums2.length;
        int[] ans;
        if ((len1 + len2) % 2 == 0) {
            int k = (len1 + len2) / 2;
            boolean flag = true;
            ans = getKMin(nums1, nums2, 0, 0, k, flag);
            return ((double)ans[0] + (double)ans[1]) / (double)2;
        } else {
```

```
            int k = (len1 + len2 + 1) / 2;
            boolean flag = false;
            ans = getKMin(nums1, nums2, 0, 0, k, flag);
            return (double)ans[0];
        }
    }

    private int[] getKMin(int[] nums1, int[] nums2, int i, int j, int k, boolean flag) {
        int len1 = nums1.length;
        int len2 = nums2.length;
        int[] ret = new int[2];
        if (i >= len1) {
            ret[0] = nums2[j + k - 1];
            if (flag) {
                ret[1] = nums2[j + k];
            }
            return ret;
        }
        if (j >= len2) {
            ret[0] = nums1[i + k - 1];
            if (flag) {
                ret[1] = nums1[i + k];
            }
            return  ret;
        }
        if (k == 1) {
            if (nums1[i] < nums2[j]) {
                ret[0] = nums1[i];
                i++;
            } else {
                ret[0] = nums2[j];
                j++;
            }
            if (flag) {
                if (i < len1 && j < len2) {
                    ret[1] = Math.min(nums1[i], nums2[j]);
                } else if (i < len1) {
                    ret[1] = nums1[i];
                } else {
                    ret[1] = nums2[j];
                }
            }
            return ret;
        }
        int tempi = Math.min(len1 - 1, i + k / 2 - 1);
        int tempj = Math.min(len2 - 1, j + k / 2 - 1);
        if (nums1[tempi] < nums2[tempj]) {
            k -= (tempi - i + 1);
            i = tempi + 1;
        } else {
            k -= (tempj - j + 1);
            j = tempj + 1;
        }
        return getKMin(nums1, nums2, i, j, k, flag);
    }
}
```