

# 第三章-作业

## 1

以下是快速排序中的一种 *PARTITION* 方法的伪代码及过程：

*PARTITION*(*A*, *p*, *r*)

```
1 x ← A[r]
2 i ← p-1
3 for j ← p to r-1
4     do if A[j] ≤ x
5         then i ← i+1
6             exchange A[i] ↔ A[j]
7 exchange A[i+1] ↔ A[r]
8 return i+1
```

	<i>i</i>	<i>p, j</i>		<i>r</i>
a)		28713564		
		<i>p, i</i> <i>j</i>		<i>r</i>
b)		28713564		
		<i>p, i</i> <i>j</i>		<i>r</i>
c)		28713564		
		<i>p, i</i> <i>j</i>		<i>r</i>
d)		28713564		
		<i>p</i> <i>i</i> <i>j</i>		<i>r</i>
e)		21783564		
		<i>p</i> <i>i</i> <i>j</i>		<i>r</i>
f)		21387564		
		<i>p</i> <i>i</i> <i>j</i>		<i>r</i>
g)		21387564		
		<i>p</i> <i>i</i>		<i>r</i>
h)		21387564		
		<i>p</i> <i>i</i>		<i>r</i>
i)		21347568		

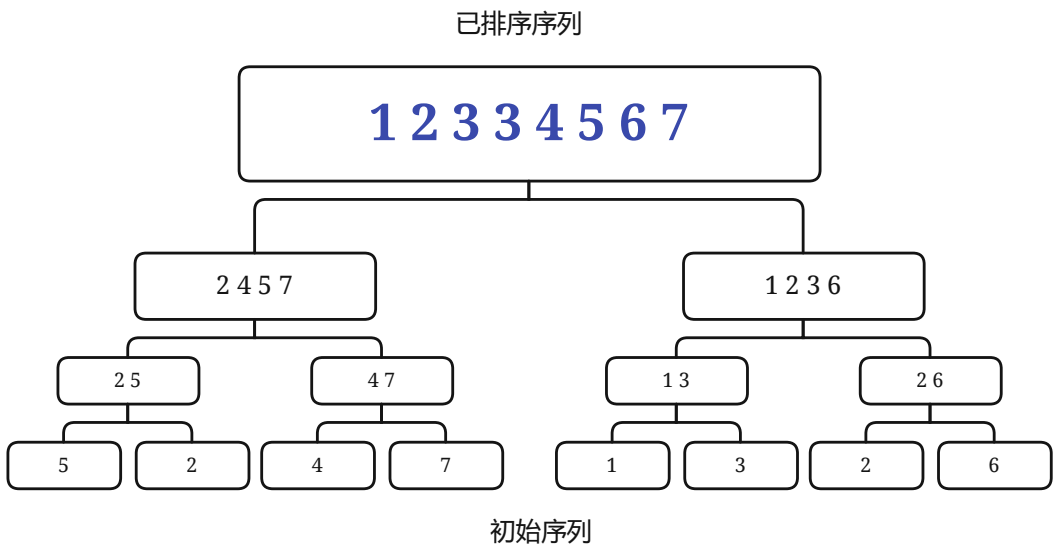
仿照上图说明 *PARTITION* 过程作用于数组  $A = \langle 13, 19, 9, 5, 12, 4, 7, 8 \rangle$  的过程。

答：

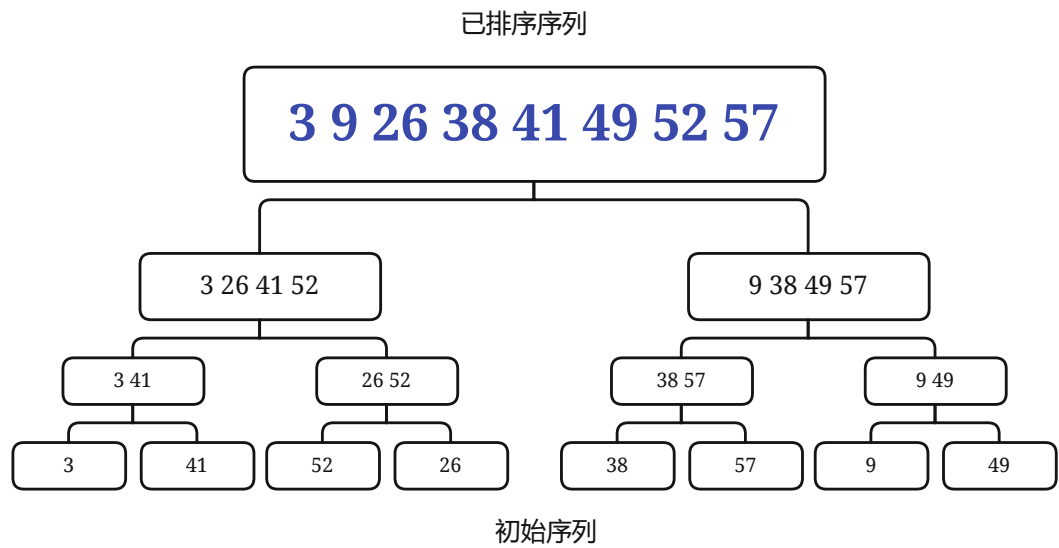
	$i$	$p, j$							$r$
a)		13	19	9	5	12	4	7	8
	$i$	$p \qquad j$							$r$
b)		13	19	9	5	12	4	7	8
	$i$	$p \qquad j$							$r$
c)		13	19	9	5	12	4	7	8
	$i$	$p \qquad j$							$r$
d)		13	19	9	5	12	4	7	8
		$p, i$				$j$		$r$	
e)		5	19	9	13	12	4	7	8
		$p, i$				$j$		$r$	
f)		5	19	9	13	12	4	7	8
		$p \qquad i$		$j$					$r$
g)		5	4	9	13	12	19	7	8
		$p \qquad i$			$r$				
h)		5	4	7	13	12	19	9	8
		$p \qquad i$			$r$				
i)		5	4	7	8	12	19	9	13

2

以下图为模型，说明合并排序在输入数组  $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$  上的执行过程。



答:



### 3

假设  $A$  和  $B$  是长度为  $n$  排好序的数组，且数组中每个数都是不同的。

#### 3.1

设计一个算法，在  $O(\log n)$  时间里找出这  $2n$  个数的中位数，其中  $2n$  个数的中位数为从小到大排序的第  $n$  个数。

答:

描述

假设序列  $A$ 、 $B$  是升序序列。分别求两个升序序列  $A$ 、 $B$  的中位数，设为  $a$  和  $b$ 。若  $a = b$ ，则  $a$  或  $b$  即为所求的中位数；否则，舍弃  $a$ 、 $b$  中较小者所在序列之较小一半，同时舍弃较大者所在序列之较大一半，要求两次舍弃的元素个数相同。在保留的两个升序序列中，重复上述过程，直到两个序列中均只含一个元素时为止，则较小者即为所求的中位数。

伪代码

$MidSearch(A, B, n)$

```
1 firstA <- 0
2 lastA <- n - 1
3 firstB <- 0
4 lastB <- n - 1
5
6 while firstA != lastA || firstB != lastB
7
8     midA <- firstA + (lastA - firstA)/2
9     midB <- firstB + (lastB - firstB)/2
10
11     if A[midA] == B[midB] then return A[midA]
12     else if A[midA] < B[midB]
13         then lastB <- midB // 舍弃B中间点以后的部分且保留中间点
14         firstA <- midA // 舍弃A中间点以前的部分且保留中间点
```

```

15         if !((lastA - firstA + 1) % 2) then firstA ++ //偶数则不保留中
    间点
16         else then lastA <- midA
17             firstB <- midB
18             if !((lastB - firstB + 1) % 2) then firstB ++
19
20     return min(A[firstA], B[firstB])

```

## 3.2

证明你的算法复杂度为  $O(\log n)$ 。

答：

- 分解：分解步骤仅仅计算并比较两个中位数，需要常量时间，因此， $D(n) = \Theta(1)$ 。
- 解决：我们递归地解决一个规模均为  $n/2$  的子问题，将贡献  $T(n/2)$  的运行时间。
- 合并：我们已经注意到这里不需要合并。

给出最坏情况运行时间  $T(n)$  的递归式：

$$T(n) = \begin{cases} \Theta(1) & \text{若 } n = 1 \\ T(n/2) + \Theta(1) & \text{若 } n > 1 \end{cases}$$

使用主定理：

$$a = 1, b = 2, f(n) = \Theta(1), n^{\log_b a} = n^{\log_2 1} = 1$$

$$f(n) = \Theta(1)$$

所以  $T(n) = f(n) \log n = \Theta(\log n)$ ，即  $T(n) = O(\log n)$

## 4

$n$  枚硬币，其中有一枚是假币，已知假币的重量较轻。现只有一个天平，要求用尽量少的比较次数找出这枚假币。我们用  $f(A, first, last)$  函数来完成上述功能。请写出该函数的伪代码（其中  $A$  表示硬币数组  $[1..n]$ ， $first$ ， $last$  为当前考虑的硬币数组中的第一个和最后一个下标，函数返回值为假币的下标）。

答：

$f(A, first, last)$

```

1  n = last - first + 1 // 定义长度
2  if n == 1 then return first
3  if n == 2 then
4      if A[first] > A[last] then return last
5      else then return first
6  if n % 2 then // 奇数
7      mid = (first + last) / 2
8      for i <- 1 to (n - 1) / 2 do //排除中位数后，两边的数组长度为 (n - 1) / 2
9          sumleft <- sumleft + A[first - 1 + i] // first 对应 1，故从 first - 1
    开始
10         sumright <- sumright + A[mid + i] // mid + 1 对应 1，故从 mid 开始
11         if sumleft == sumright then return mid
12         else if sumleft > sumright then return f(A, mid + 1, last) //右边轻，假币
    在右边
13         else then return f(A, first, mid - 1)
14  if !(n % 2) then // 偶数
15      mid = (first + last - 1) / 2

```

```

16   for i <- 1 to n/2 do // 两边的数组长度为 n/2
17       sumleft <- sumleft + A[first - 1 + i]
18       sumright <- sumleft + A[mid + i]
19   if sumleft > sumright then return f(A,mid + 1,last)
20   else then return f(A,first,mid)

```

## 5

假设给定一个**不同整数**组成的**已经排好序**的数组  $A[1, \dots, n]$ ，我们需要在该数组中查找是否存在索引  $i$ ，使得  $A[i] = i$ 。

### 5.1

尝试用描述分治算法来解决该问题。要求写出伪代码。

答：

描述

假设序列  $A$  是升序序列。计算中位数  $mid$  并进行  $A[mid] = mid$  的判断，如果为真，则返回 *true*；如果  $A[mid] > mid$ ，**因为数组是由不同的整数组成的**，所以  $mid$  之后的所有元素都无法满足  $A[i] = i$ ，可以舍弃；如果  $A[mid] < mid$ ，同理， $mid$  之前的所有元素也无法满足  $A[i] = i$ ，可以舍弃。

eg: 假设数组有相同元素，则：

下标	1	2	3	4	5	6	7	8	9
值	1	2	3	4	6	6	6	7	10

有相同的整数使得寻找中位数这一举动变得相对无意义，因为即使判断了  $A[mid] = mid$  也不能以此把数组划分为有意义的子数组。你可以以该中位数为界将数组分成两个数组，重复上述过程，直到数组的长度为 1 为止。但这样做的时间复杂度和线性的查找没有区别，都是  $O(n)$ 。

不同的值使得情况大为不同。还是使用上面的例子，如果  $A[5] = 6$ ，那么  $A[6]$  必然不可能是 6，只能是大于等于 7 的数，在这样的情况下  $A[7]$  绝对不可能为 7。这样的情况可以一直向后延伸，所以  $mid$  之后的所有元素都无法满足  $A[i] = i$ 。

伪代码

*FindEqual(A, first, last)*

```

1   n = last - first + 1
2   if n == 1 then return A[first] == first
3   if n % 2 then
4       mid = (first + last)/2
5       if A[mid] == mid then return true
6       else if A[mid] > mid then FindEqual(A,first,mid - 1)
7       else then FindEqual(A,mid + 1,last)
8   if !(n % 2) then
9       mid = (first + last - 1)/2
10      if A[mid] == mid then return true
11      else if A[mid] > mid then FindEqual(A,first,mid - 1)
12      else then FindEqual(A,mid + 1,last)

```

## 5.2

使用**主定理**估计第 1 小题中你所描述算法的复杂度。

注意：给出的算法应当保证在  $O(\lg n)$  的运行时间内。

答：

假定原问题规模是 2 的幂，可以简化递归式。这时每个分解步骤将产生规模刚好为  $n/2$  的两个子序列。

- 分解：分解步骤只有赋值和判断表达式几个步骤，需要常量时间，因此， $D(n) = \Theta(1)$ 。
- 解决：我们递归地解决一个规模近似为  $n/2$  的子问题，将贡献  $T(n/2)$  的运行时间。
- 合并：我们已经注意到这里不需要合并。

给出最坏情况运行时间  $T(n)$  的递归式：

$$T(n) = \begin{cases} \Theta(1) & \text{若 } n = 1 \\ T(n/2) + \Theta(1) & \text{若 } n > 1 \end{cases}$$

使用主定理：

$$a = 1, b = 2, f(n) = \Theta(1), n^{\log_b a} = n^{\log_2 1} = 1 \\ f(n) = \Theta(1)$$

所以  $T(n) = f(n) \lg n = O(\lg n)$ 。

## 附录

### 3.c

```
1  #include <stdio.h>
2  int MidSearch(int A[], int B[], int n)
3  {
4      int firstA = 0;
5      int lastA = n - 1;
6      int firstB = 0;
7      int lastB = n - 1;
8      int midA, midB;
9
10     while ((firstA != lastA) || firstB != lastB)
11     {
12         midA = firstA + (lastA - firstA) / 2;
13         midB = firstB + (lastB - firstB) / 2;
14
15         if (A[midA] == B[midB])
16             return A[midA];
17         else if (A[midA] < B[midB])
18         {
19             lastB = midB;
20             firstA = midA;
21             if (!((lastA - firstA + 1) % 2))
22                 firstA++;
23         }
24         else
25         {
26             lastA = midA;
27             firstB = midB;
28             if (!((lastB - firstB + 1) % 2))
29                 firstB++;
30         }
31     }
32 }
```

```

30     }
31 }
32 if (A[firstA] < B[firstB])
33     return A[firstA];
34 else
35     return B[firstB];
36 }
37
38 int main()
39 {
40     int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
41     int B[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
42     printf("%d", MidSearch(A, B, 10));
43     return 0;
44 }

```

#### 4.c

```

1  #include <stdio.h>
2  int f(int A[], int first, int last)
3  {
4      int n = last - first + 1;
5      int mid;
6      int sumleft = 0;
7      int sumright = 0;
8      if (n == 1)
9          return first;
10     if (n == 2)
11     {
12         if (A[first] > A[last])
13             return last;
14         else
15             return first;
16     }
17     if (n % 2)
18     {
19         mid = (first + last) / 2;
20         for (int i = 1; i <= (n - 1) / 2; i++)
21         {
22             sumleft += A[first - 1 + i];
23             sumright += A[mid + i];
24         }
25         if (sumleft == sumright)
26             return mid;
27         else if (sumleft > sumright)
28             return f(A, mid + 1, last);
29         else
30             return f(A, first, mid - 1);
31     }
32     if (!(n % 2))
33     {
34         mid = (first + last - 1) / 2;
35         for (int i = 1; i <= n / 2; i++)
36         {
37             sumleft += A[first - 1 + i];
38             sumright += A[mid + i];
39         }

```

```

40         if (sumleft > sumright)
41             return f(A, mid + 1, last);
42         else
43             return f(A, first, mid);
44     }
45 }
46
47 int main()
48 {
49     int A[10] = {10, 10, 10, 10, 9, 10, 10, 10, 10, 10};
50     printf("%d", f(A, 0, 9));
51     return 0;
52 }

```

## 5.c

```

1  #include <stdio.h>
2  int FindEqual(int A[], int first, int last)
3  {
4      int n = last - first + 1;
5      int mid;
6      if (n == 1)
7          return A[first] == first;
8      if (n % 2)
9      {
10         mid = (first + last) / 2;
11         if (A[mid] == mid)
12             return 1;
13         else if (A[mid] > mid)
14             return FindEqual(A, first, mid - 1);
15         else
16             return FindEqual(A, mid + 1, last);
17     }
18     if (!(n % 2))
19     {
20         mid = (first + last - 1) / 2;
21         if (A[mid] == mid)
22             return 1;
23         else if (A[mid] > mid)
24             return FindEqual(A, first, mid - 1);
25         else
26             return FindEqual(A, mid + 1, last);
27     }
28 }
29 int main()
30 {
31     //每个都符合
32     int A[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
33     printf("%d\n", FindEqual(A, 0, 9));
34
35     //只有B[0]=0
36     int B[10] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
37     printf("%d\n", FindEqual(B, 0, 9));
38
39     //都不符合
40     int C[10] = {1, 2, 3, 4, 5, 6, 7, 7, 9, 10};
41     printf("%d\n", FindEqual(C, 0, 9));

```



```
42     return 0;  
43 }
```