



中国科学院大学
University of Chinese Academy of Sciences

计算机科学导论实验

编程基础-2

切片与递归

简版快速排序程序 **fastsort.go**

编程基础实验安排

周次	日期	实验课	课程内容	作业提交内容	截止时间
2	3月8日	编程基础-1	从hello到 Name2Number.go	name_to_number-0.go文件 与程序结果	2024/3/24 23:30
3	3月15日	编程基础-2	基本数据类型、基本语 句、循环	Name2Number.go文件与程序 结果	
			切片与递归 简版快速排序程序 fastsort.go	fastsort.go文件与程序中间结 果	
4	3月22日	编程基础-3		修改函数签名后的fastsort.go 文件与程序中间结果	

- 作业提交后会立刻返回成绩。
- 在截止时间前四次作业均可反复多次提交。

提纲

- 目标：
 - 修改Name2Number.go，掌握for循环
 - 开发出fastsort.go排序程序，理解递归与切片
- 步骤
 - 理解并重现Name2Number.go，用你自己的姓名拼音和循环
 - 使用8元素例子理解fastsort排序算法
 - 理解fastsort.go程序框架
 - 现场实现课程提供的fastsort.go程序
 - 验证你的fastsort.go程序的正确性
 - 提交你的fastsort.go程序

课件中包含教科书未包括的素材引用，特此致谢

1 采用占位符%c实现格式动词%d

name_to_number-0.go

```
package main
import "fmt"
func main() {
    var name string = "Alan Turing"
    sum := 0
    for i := 0; i < 11; i++ {
        sum = sum + int(name[i])
    }
    fmt.Printf("%d\n", sum)
}
```

```
> ./name_to_number-0
> 1045
>
```

Name2Number.go

```
package main
import "fmt"
func main() {
    var name string = "Alan Turing"
    sum := 0
    for i := 0; i < 11; i++ {
        sum = sum + int(name[i])
    }
    var sum_bytes [4]byte
    var j int
    for j = 3; sum != 0; j-- {
        sum_bytes[j] = byte(sum%10) + '0'
        sum = sum / 10
    }
    fmt.Printf("%c", sum_bytes[0])
    fmt.Printf("%c", sum_bytes[1])
    fmt.Printf("%c", sum_bytes[2])
    fmt.Printf("%c", sum_bytes[3])
    fmt.Printf("\n")
}
```

```
> ./Name2Number
> 1045
>
```

1 采用占位符%c实现格式动词%d

name_to_number-0.go



Name2Number.go

fmt.Printf("%d\n", sum)



```
var sum_bytes [4]byte
var j int
for j = 3; sum != 0; j-- {
    sum_bytes[j] = byte(sum%10) + '0'
    sum = sum / 10
}
fmt.Printf("%c", sum_bytes[0])
fmt.Printf("%c", sum_bytes[1])
fmt.Printf("%c", sum_bytes[2])
fmt.Printf("%c", sum_bytes[3])
fmt.Printf("\n")
```

```
> ./name_to_number-0
> 1045
>
```

```
> ./Name2Number
> 1045
>
```

如何用%c实现%d?

```
package main
import "fmt"
func main() {
    var name string = "Alan Turing"
    sum := 0
    for i := 0; i < 11; i++ {
        sum = sum + int(name[i])
    }
    var sum_bytes [4]byte
    var j int
    for j = 3; sum != 0; j-- {
        sum_bytes[j] = byte(sum%10) + '0'
        sum = sum / 10
    }

    fmt.Printf("%c", sum_bytes[0])
    fmt.Printf("%c", sum_bytes[1])
    fmt.Printf("%c", sum_bytes[2])
    fmt.Printf("%c", sum_bytes[3])
    fmt.Printf("\n")
}
```

用字符占位符%c
实现十进制占位符 %d
fmt.Printf("%d\n", sum)

如何用%c实现%d?

```
for j = 3; sum != 0; j-- {  
    sum_bytes[j] = byte(sum%10) + '0'  
    sum = sum / 10  
}
```

```
fmt.Printf("%c", sum_bytes[0])  
fmt.Printf("%c", sum_bytes[1])  
fmt.Printf("%c", sum_bytes[2])  
fmt.Printf("%c", sum_bytes[3])
```

使用整数除法 / 和求余 % 操作，
从数值1045依次摘取出数字5, 4, 0, 1

```
sum % 10  
sum = sum / 10
```

sum_bytes[3]	1045 % 10 = 5
sum	1045 / 10 = 104
sum_bytes[2]	104 % 10 = 4
sum	104 / 10 = 10
sum_bytes[1]	10 % 10 = 0
sum	10 / 10 = 1
sum_byte[0]	1 % 10 = 1
sum	1 / 10 = 0

sum_bytes	=	['1', '0', '4', '5']
数组索引		0 1 2 3

四个打印语句

```
fmt.Printf("%c", ...)
```

依次打印出1, 0, 4, 5四个字符

为什么要做类型转换

`sum_bytes[j] = byte(sum%10) + '0'`
而不是 `sum_bytes[j] = sum % 10`

```
package main
import "fmt"
func main() {
    var name string = "Alan Turing"
    sum := 0
    for i := 0; i < 11; i++ {
        sum = sum + int(name[i])
    }
    // 此时, sum == 1045
    var sum_bytes [4]byte
    var j int
    for j = 3; sum != 0; j-- {
        sum_bytes[j] = byte(sum%10) + '0'
        sum = sum / 10
    }
    fmt.Printf("%c", sum_bytes[0])
    fmt.Printf("%c", sum_bytes[1])
    fmt.Printf("%c", sum_bytes[2])
    fmt.Printf("%c", sum_bytes[3])
    fmt.Printf("\n")
}
```

`sum_bytes` 是一个 `byte` 数组
`sum_byte[j]` 的类型是 `byte`,
但是 `sum` 的类型是 `int`

假设 `j=3` (最初)
`sum` 是 1045,
`sum % 10` 是 $1045 \% 10 = 5$, 为 `int`
00000000.....00000101

`byte(sum%10)`, i.e., `byte(5)`
把这个 64-bit 的值转换成一个 `uint8`
类型的数字, 值为 00000101

`byte(5) + '0'`
 $= 5 + 48$
 $= 00000101 + 00110000$
 $= 00110101 = 53 = '5'$

因此, `sum_bytes[3]` 是 '5'

请每位同学提交Name2Number.go

```
package main // Name2Number.go
import "fmt"
func main() {
    var name string = "Alan Turing"

    sum := 0
    for i := 0; i < len(name); i++ {
        sum = sum + int(name[i])
    }
    var sum_bytes [4]byte
    var j int
    for j = 3; sum != 0; j-- {
        sum_bytes[j] = byte(sum%10) + '0'
        sum = sum / 10
    }
    fmt.Printf("%c", sum_bytes[0])
    fmt.Printf("%c", sum_bytes[1])
    fmt.Printf("%c", sum_bytes[2])
    fmt.Printf("%c", sum_bytes[3])
    fmt.Printf("\n")
}
```

使用你自己的姓名拼音，如"Zhang San"

使用一个循环替换此处的四个打印语句
`fmt.Printf("%c", ...)`

作业提交要求：

1. 替换成姓名的拼音：请与国科大邮箱一致。例如邮箱是`zhangsan24@mailsucas.ac.cn`，则姓名拼音是"Zhang San"（每个字拼音首字母大写，之间用1个空格隔开）
2. 使用一个循环替换四个打印语句
`fmt.Printf("%c", ...)`
3. 其他部分代码**不要修改！**

基础编程

第2次作业 ▾

请提交

• Name2Number.go 未选择文件

• 请输入你的姓名编码

作业提交链接：https://course.things.ac.cn:10088/exp/basic_programming

作业提交截止时间：[2024/3/24 23:30](#)

2 课堂上讲的简版快速排序算法fastsort

- **输入**：8元素整数数组d = [8 3 6 7 2 1 4 5]（假定没有重复元素）
- **输出**：8元素整数数组d = [1 2 3 4 5 6 7 8]
 - 元素从小到大排好序了
- **步骤**：调用fastsort(d)。

函数fastsort(A)的计算过程如下：

1. 基线情况（base case）：

如果A只包含0个元素（如[]）或1个元素（如[3]），fastsort(A)结束

2. 选择A的最后一个元素作为标杆元素（pivot）

3. 调用划分函数partition

● 将小于标杆元素的元素放入lowerA子数组（称为小数组）中

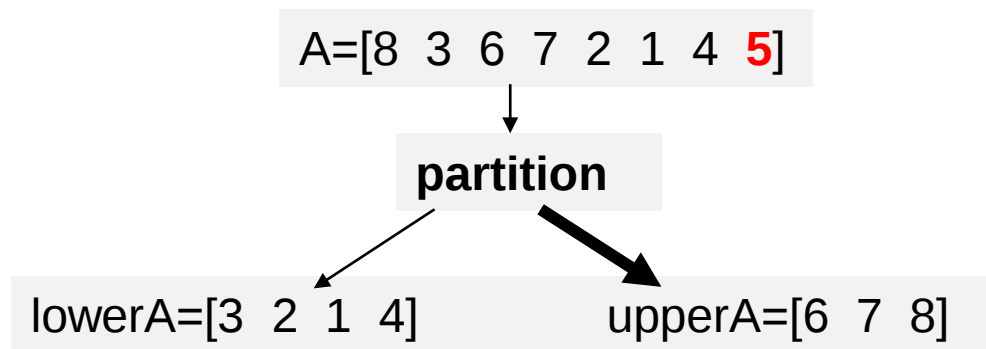
● 将大于标杆元素的元素放入upperA子数组（称为大数组）中

4. 递归调用fastsort(lowerA)

5. 递归调用fastsort(upperA)

lowerA和upperA中元素的顺序是由partition函数的算法唯一确定的，目前先不考虑

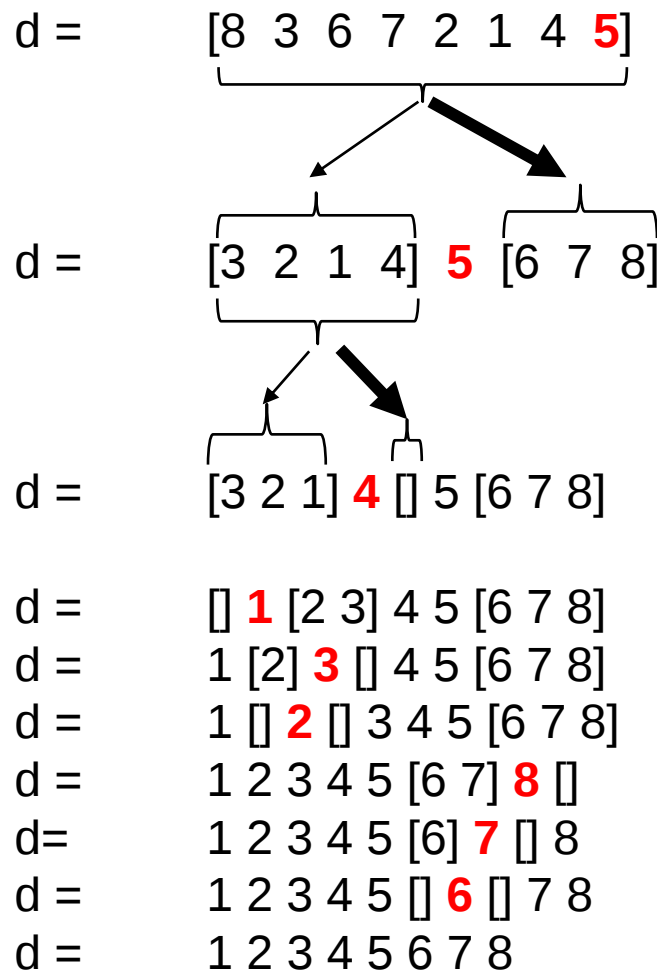
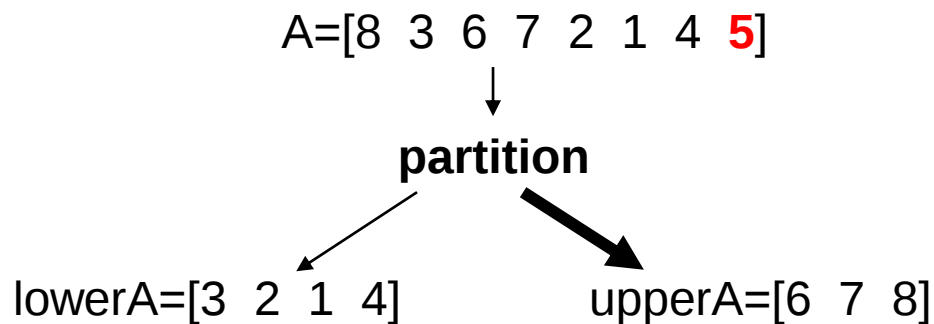
如何体现递归与切片



2 课堂上讲的简版快速排序算法fastsort

函数fastsort(A)的计算过程如下：

1. 基线情况（base case）：如果A只包含0个元素（如[]）或1个元素（如[3]）， fastsort(A)结束
2. 选择A的最后一个元素作为标杆元素（pivot）
3. 调用划分函数partition
将小于标杆元素的元素放入lowerA子数组（称为小数组）中，将大于标杆元素的元素放入upperA子数组（称为大数组）中
4. 递归调用fastsort(lowerA)
5. 递归调用fastsort(upperA)



如何体现递归与切片

执行过程的简略表示可能引起误解

$\text{len}(A) < 2$ 时, $\text{fastsort}(A)$ 直接退出

无省略的执行过程

d = [8 3 6 7 2 1 4 5]

d = [3 2 1 4] 5 [6 7 8]

d = [3 2 1] 4 [] 5 [6 7 8]

d = [] 1 [2 3] 4 5 [6 7 8]

d = 1 [2] 3 [] 4 5 [6 7 8]

d = 1 [] 2 [] 3 4 5 [6 7 8]

d = 1 2 3 4 5 [6 7] 8 []

d = 1 2 3 4 5 [6] 7 [] 8

d = 1 2 3 4 5 [] 6 [] 7 8

d = 1 2 3 4 5 6 7 8

fs([8 3 6 7 2 1 4 5]) d =

fs([3 2 1 4])

fs([3 2 1])

fs([])

fs([2 3])

fs([2])

fs([])

fs([])

fs([6 7 8])

fs([6 7])

fs([6])

fs([])

fs([])

d = [8 3 6 7 2 1 4 5]

d = [3 2 1 4] 5 [6 7 8]

d = [3 2 1] 4 [] 5 [6 7 8]

d = [] 1 [2 3] 4 [] 5 [6 7 8]

d = 1 [2 3] 4 [] 5 [6 7 8]

d = 1 [2] 3 [] 4 [] 5 [6 7 8]

d = 1 2 3 [] 4 [] 5 [6 7 8]

d = 1 2 3 4 [] 5 [6 7 8]

d = 1 2 3 4 5 [6 7 8]

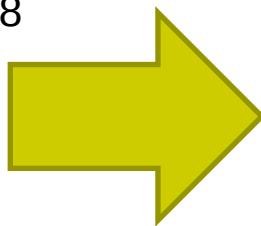
d = 1 2 3 4 5 [6 7] 8 []

d = 1 2 3 4 5 [6] 7 [] 8 []

d = 1 2 3 4 5 6 7 [] 8 []

d = 1 2 3 4 5 6 7 8 []

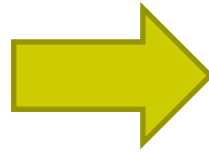
d = 1 2 3 4 5 6 7 8



3. 从算法导出程序fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)



- 函数fastsort(A)的计算过程如下:
 1. ...

- 函数partition(A)的计算过程如下:
 1. ...

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

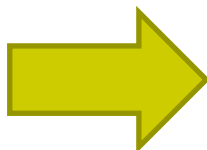
func fastsort(A []int) { ... }

func partition(A []int) ([]int, []int)
{ ... }
```

3. 从算法导出程序fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)



- 函数fastsort(A)的计算过程如下:

1. ...

- 函数partition(A)的计算过程如下:

1. ...

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) { ... }

func partition(A []int) ([]int, []int)
{ ... }
```

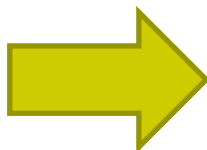
理解切片和数组:

- 在main调用fastsort(s)前, s[0]的值是多少?

3. 从算法导出程序fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)



- 函数fastsort(A)的计算过程如下:

1. ...

- 函数partition(A)的计算过程如下:

1. ...

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) { ... }

func partition(A []int) ([]int, []int)
{ ... }
```

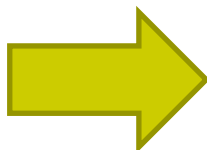
理解切片和数组:

- 执行fastsort(s)之后, d[0]的值是多少?

3. 从算法导出程序fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)



- 函数fastsort(A)的计算过程如下:

1. ...

- 函数partition(A)的计算过程如下:

1. ...

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) { ... }

func partition(A []int) ([]int, []int)
{ ... }
```

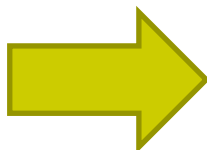
理解切片和数组:

- 若把“fastsort(s)”改成“fastsort(d)”，该程序还能正确执行吗？
 - 若不能，会产生编译错误还是运行时错误？

3. 从算法导出程序fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)



- 函数fastsort(A)的计算过程如下:

1. ...

- 函数partition(A)的计算过程如下:

1. ...

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) { ... }

func partition(A []int) ([]int, []int)
{ ... }
```

理解切片和数组:

- 若把“fmt.Println(s)”改成“fmt.Println(d)”，程序打印的内容还跟原来相同吗？

3. 理解fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)
- 函数fastsort(A)的计算过程如下:
 1. 基线情况: 如果A只包含0个元素 (如[]) 或1个元素 (如[3]), fastsort(A)结束
 2. 选择A的最后一个元素作为标杆元素 (pivot)
 3. 调用函数partition(A)
 4. 递归调用fastsort(lowerA)
 5. 递归调用fastsort(upperA)
- 函数partition(A)的计算过程如下:
 1. ...



```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是, 退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int)
{ ... }
```

3. 理解fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)

- 函数fastsort(A)的计算过程如下:

1. 基线情况: 如果A只包含0个元素 (如[]) 或1个元素 (如[3]), fastsort(A)结束
2. 选择A的最后一个元素作为标杆元素 (pivot)
3. 调用函数partition(A)
4. 递归调用fastsort(lowerA)
5. 递归调用fastsort(upperA)

- 函数partition(A)的计算过程如下:

1. ...

- 若删除此块代码, 程序还能正确打印结果吗?
 - 若不能, 会产生编译错误还是运行时错误?

```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是, 退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int)
{ ... }
```

3. 理解fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)
- 函数fastsort(A)的计算过程如下:
 1. 基线情况: 如果A只包含0个元素 (如[]) 或1个元素 (如[3]), fastsort(A)结束
 2. 选择A的最后一个元素作为标杆元素 (pivot)
 3. 调用函数partition(A)
 4. 递归调用fastsort(lowerA)
 5. 递归调用fastsort(upperA)
- 函数partition(A)的计算过程如下:
 1. ...



```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是, 退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int)
{ ... }
```

- 这两行代码是对lowerA和upperA排序, 为什么执行完后A会变得有序?

3. 理解fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)
- 函数fastsort(A)的计算过程如下:
 1. 基线情况: 如果A只包含0个元素 (如[]) 或1个元素 (如[3]), fastsort(A)结束
 2. 选择A的最后一个元素作为标杆元素 (pivot)
 3. 调用函数partition(A)
 4. 递归调用fastsort(lowerA)
 5. 递归调用fastsort(upperA)
- 函数partition(A)的计算过程如下:
 1. ...



```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是, 退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int)
{ ... }
```

- 为什么fastsort代码中找不到“选择A的最后一个元素作为标杆元素 (pivot)” 对应的代码?

简版快速排序算法fastsort

- 输入：8元素整数数组d = [8 3 6 7 2 1 4 5]
- 输出：8元素整数数组d = [1 2 3 4 5 6 7 8]
 - 元素从小到大排好序了
- 步骤：调用fastsort(d)。

函数fastsort(A)的计算过程如下：

1. 基线情况（base case）

- 如果A只包含0个或1个元素（如[], [3]），fastsort(A)结束

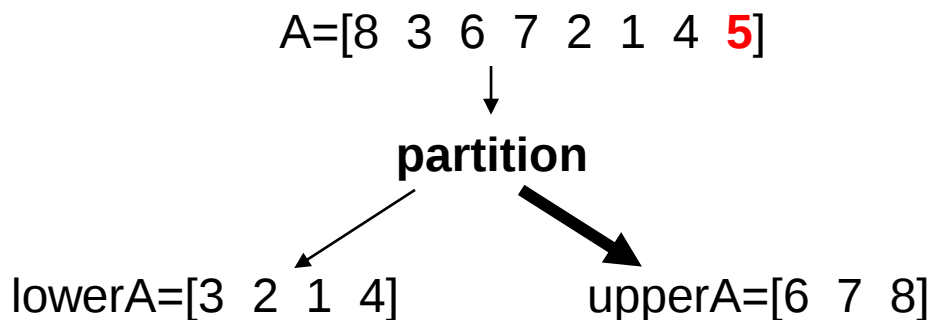
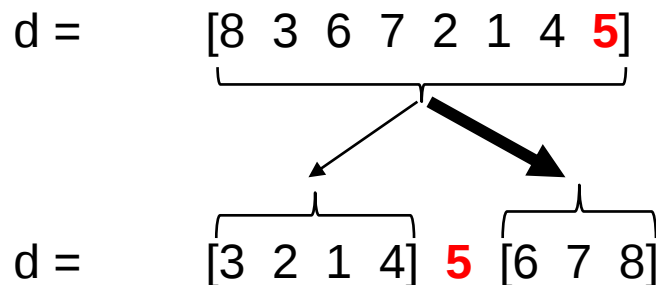
编程假设：选择A的最后一个元素作为标杆元素

2. 调用划分函数partition

- 将小于标杆值的元素放入lowerA子数组（称为小数组）中，
将大于标杆值的元素放入upperA子数组（称为大数组）中

3. 递归调用fastsort(lowerA)

4. 递归调用fastsort(upperA)



简版快速排序算法fastsort

- 输入：8元素整数数组d = [8 3 6 7 2 1 4 5]
- 输出：8元素整数数组d = [1 2 3 4 5 6 7 8]
 - 元素从小到大排好序了
- 步骤： 调用fastsort(d)。

函数fastsort(A)的计算过程如下：

1. 基线情况（base case）

- 如果A只包含0个或1个元素（如[], [3]），fastsort(A)结束

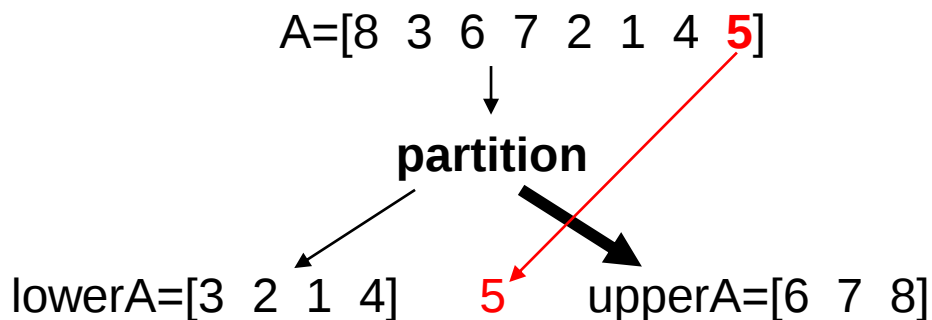
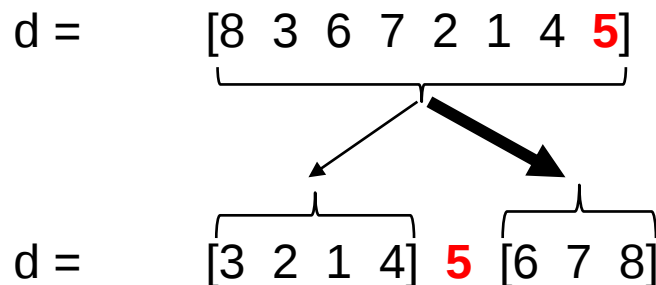
2. 调用划分函数partition

- 将小于标杆值的元素放入lowerA子数组（称为小数组）中，
将大于标杆值的元素放入upperA子数组（称为大数组）中

- 注意：标杆的位置变了

3. 递归调用fastsort(lowerA)

4. 递归调用fastsort(upperA)



3. 理解fastsort.go

简版快速排序算法

- **输入**: 8元素整数数组d = [8 3 6 7 2 1 4 5]
- **输出**: 8元素整数数组d = [1 2 3 4 5 6 7 8]
- **步骤**: 调用fastsort(d)
- 函数fastsort(A)的计算过程如下:
 1. 基线情况: 如果A只包含0个元素 (如[]) 或1个元素 (如[3]), fastsort(A)结束
 2. 选择A的最后一个元素作为标杆元素
 3. 调用函数partition(A)
 4. 递归调用fastsort(lowerA)
 5. 递归调用fastsort(upperA)
- 函数partition(A)的计算过程如下:
 1. 将小于标杆元素的A的元素放入lowerA子数组, 将大于标杆元素的A的元素放入upperA子数组
 2. 返回两个子数组lowerA和upperA

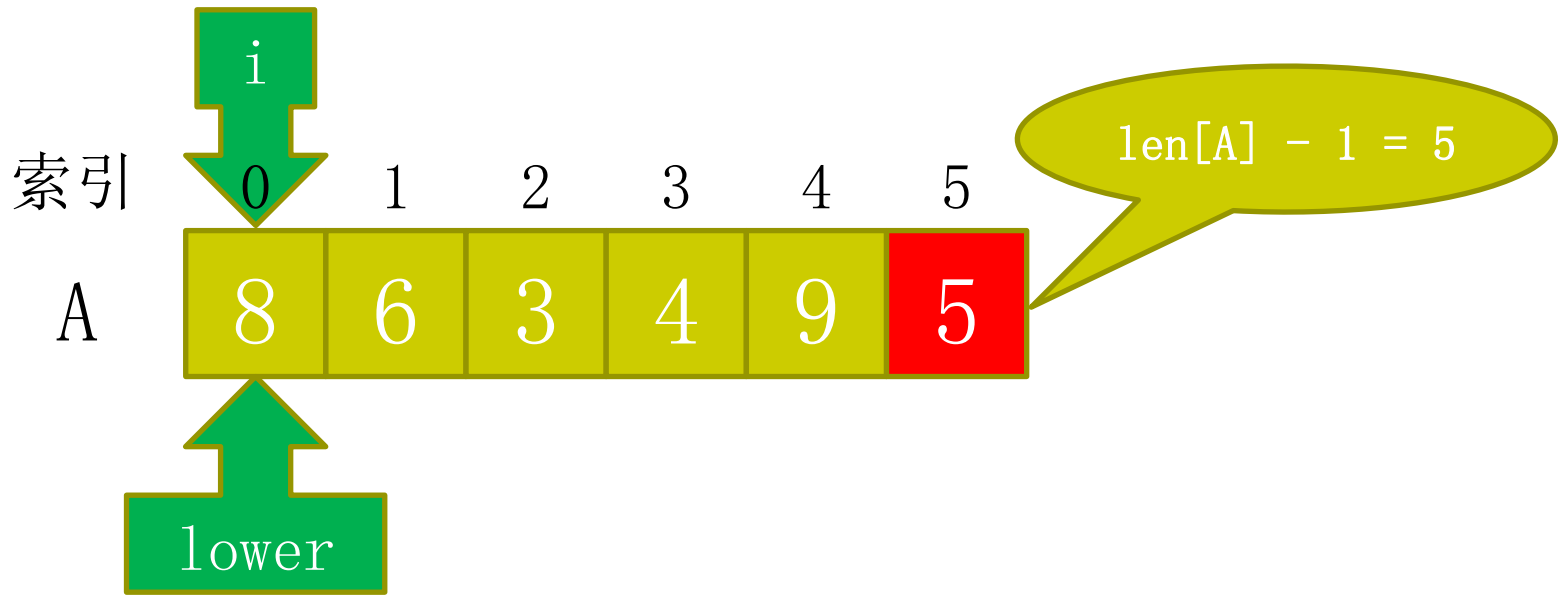


```
package main // fastsort.go
import "fmt"
var d [8]int = [8]int {8,3,6,7,2,1,4,5}
var s []int = d[0:8]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是, 退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int) { // len(A)>1
    lower := 0 // lower对应lowerA的长度, 初始值为0
    for i:= 0; i<len(A); i++ { // 逐个扫描A的元素A[i]
        // 此处插入你的代码
        // 建议: 如果A[i]<标杆值,
        // A[i]与A[lower]交换并更新lower
    }
    // 此时lower的值是lowerA的长度
    // 交换A[lower]与标杆, 使得标杆紧跟在lowerA之后
    A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
    // 返回切片lowerA和upperA
    // 注意两个切片的起始索引和结束索引
    return A[0:lower], A[lower+1:len(A)]
}
```

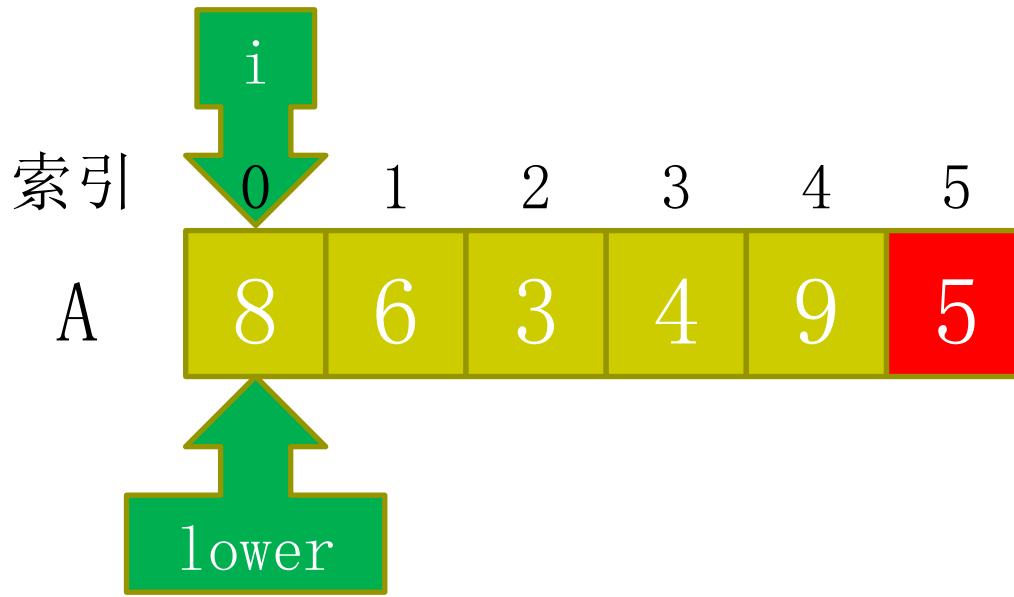
Partition ([8 6 3 4 9 5])



- $lower$ 、 i 的初始值为0
- $pivot$ 为切片的最后一个

```
lower := 0  
pivot := A[len(A)-1]
```

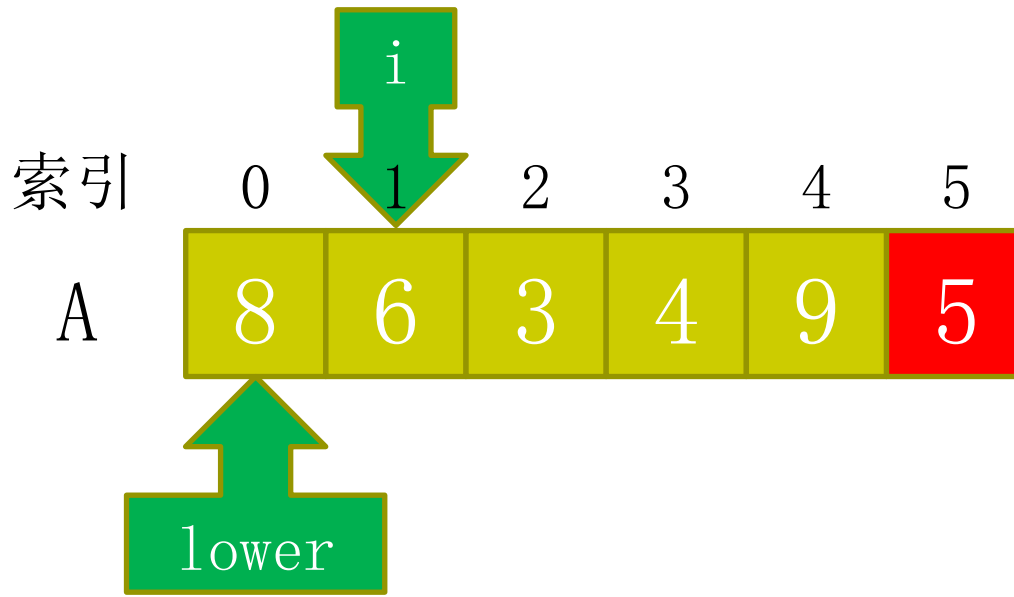
Partition ([8 6 3 4 9 5])



- $A[i] = A[0] \geq 5$
- 不满足, $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

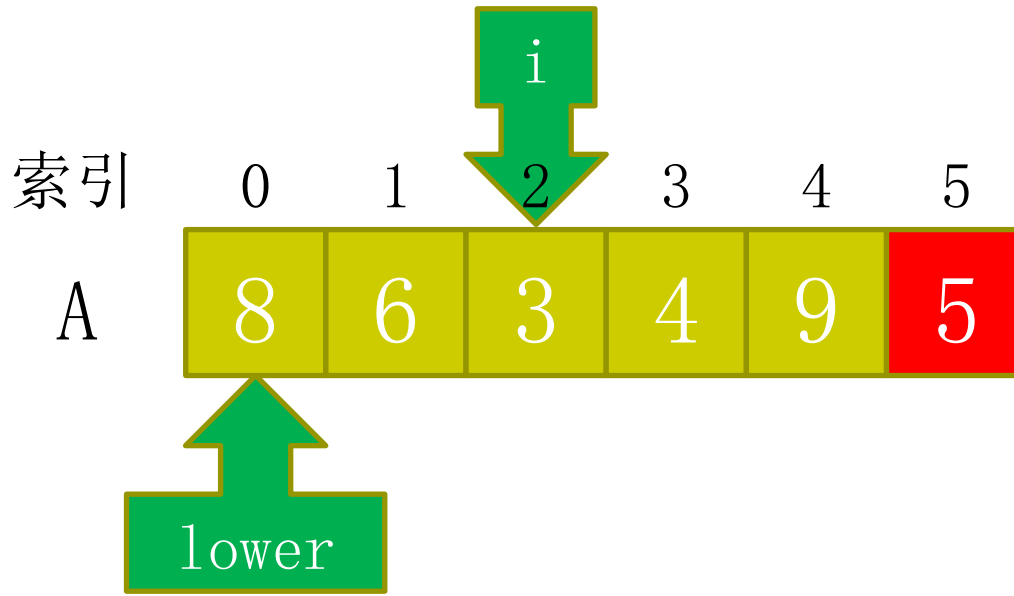
Partition ([8 6 3 4 9 5])



- $A[i] = A[1] \geq 5$
- 不满足, $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

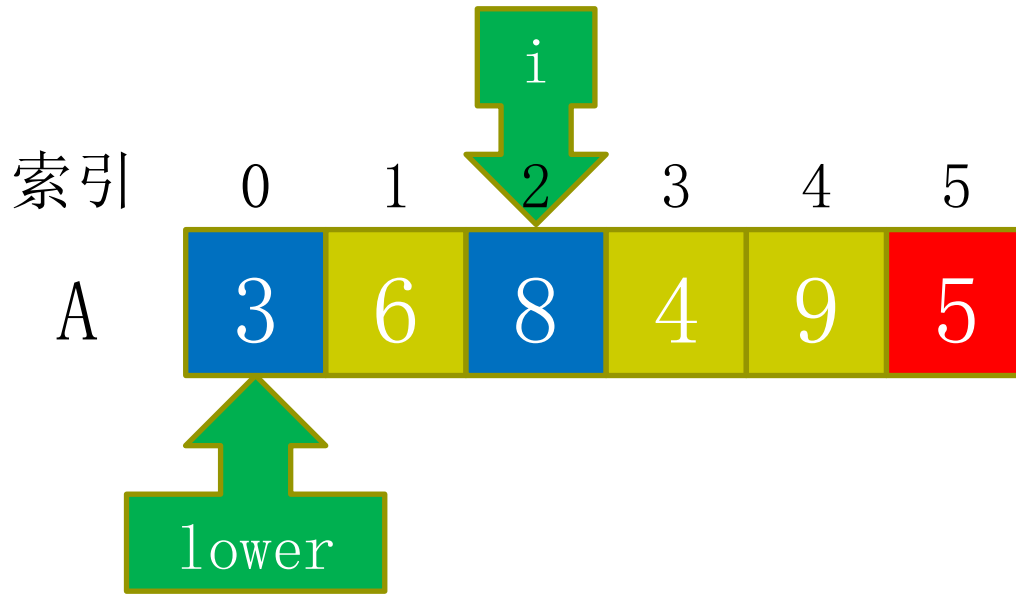
Partition ([8 6 3 4 9 5])



- $A[i] = A[2] < 5$
- 交换 $A[i]$ 与 $A[\text{lower}]$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

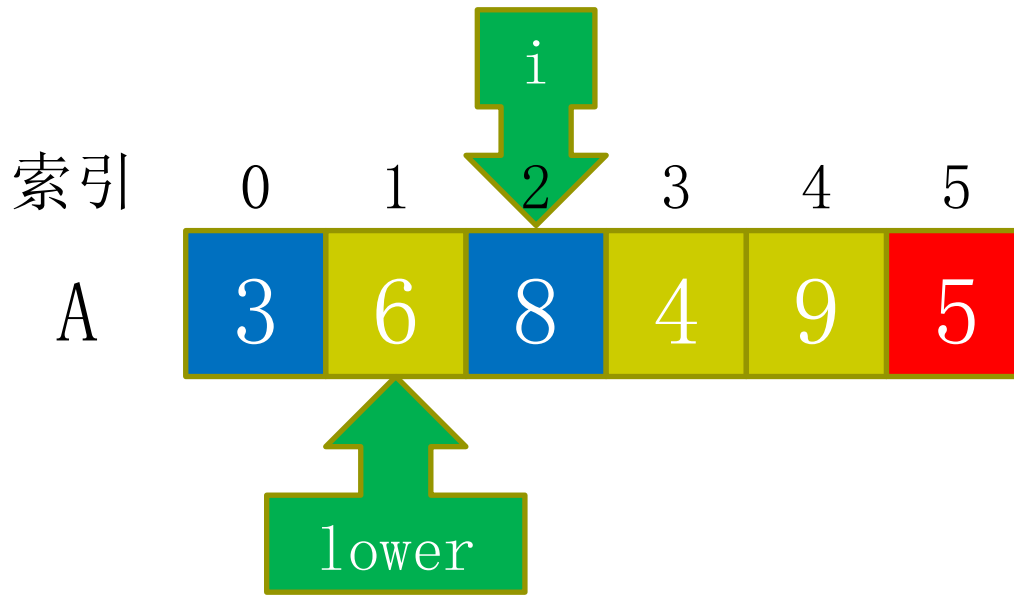
Partition ([8 6 3 4 9 5])



- $A[i] = A[2] < 5$
- 更新 $lower++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

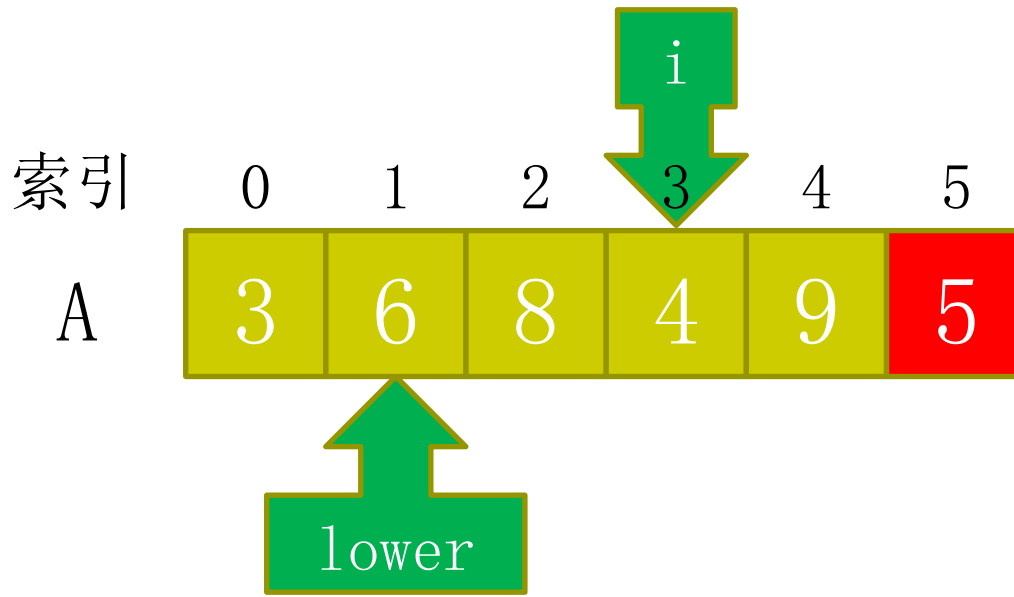
Partition ([8 6 3 4 9 5])



- $A[i] = A[2] < 5$
- $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

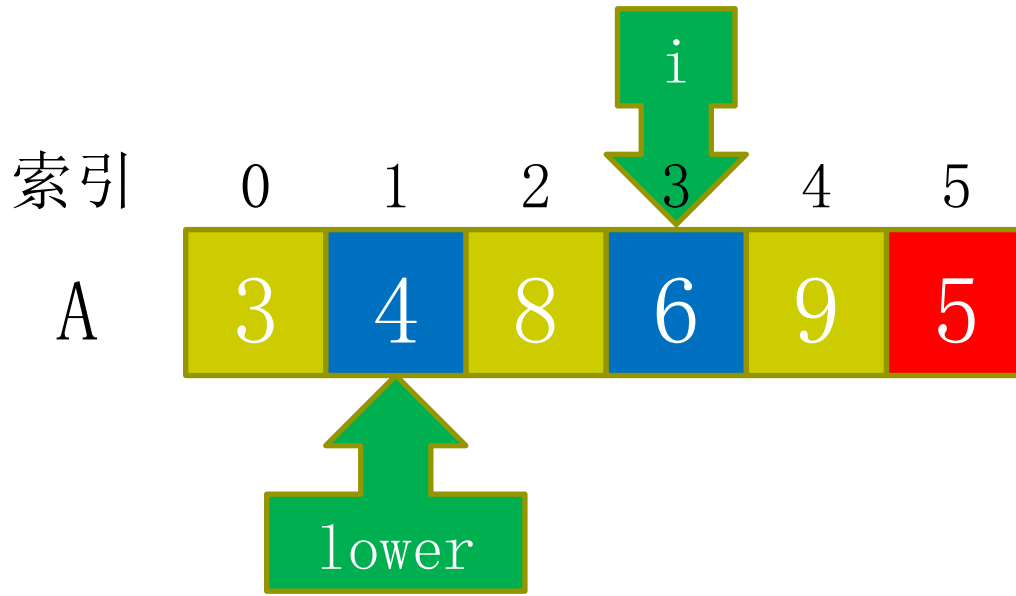
Partition ([8 6 3 4 9 5])



- $A[i] = A[3] < 5$
- 交换 $A[i]$ 与 $A[lower]$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

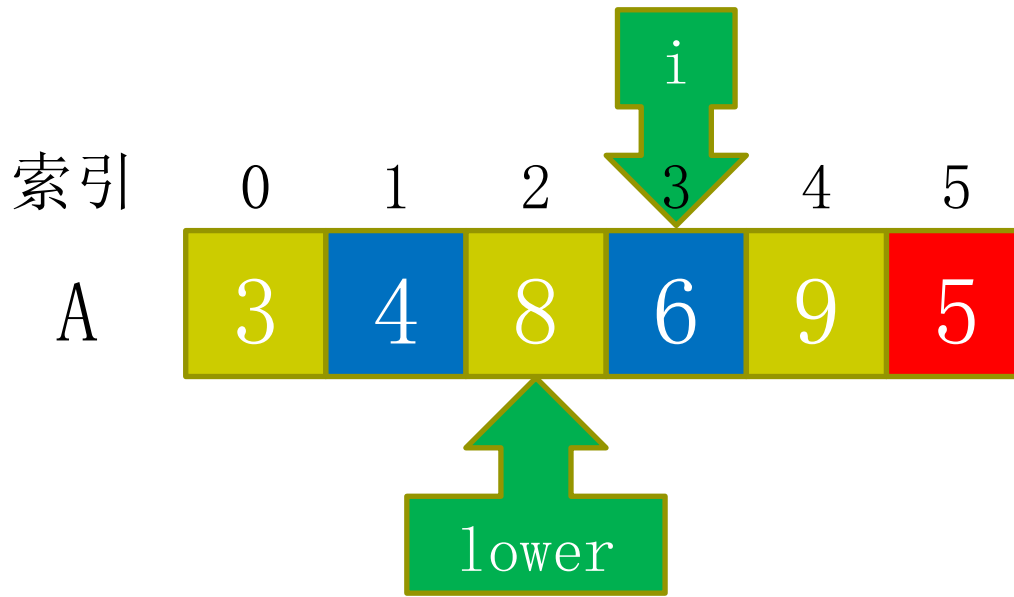
Partition ([8 6 3 4 9 5])



- $A[i] = A[3] < 5$
- 更新 $lower++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

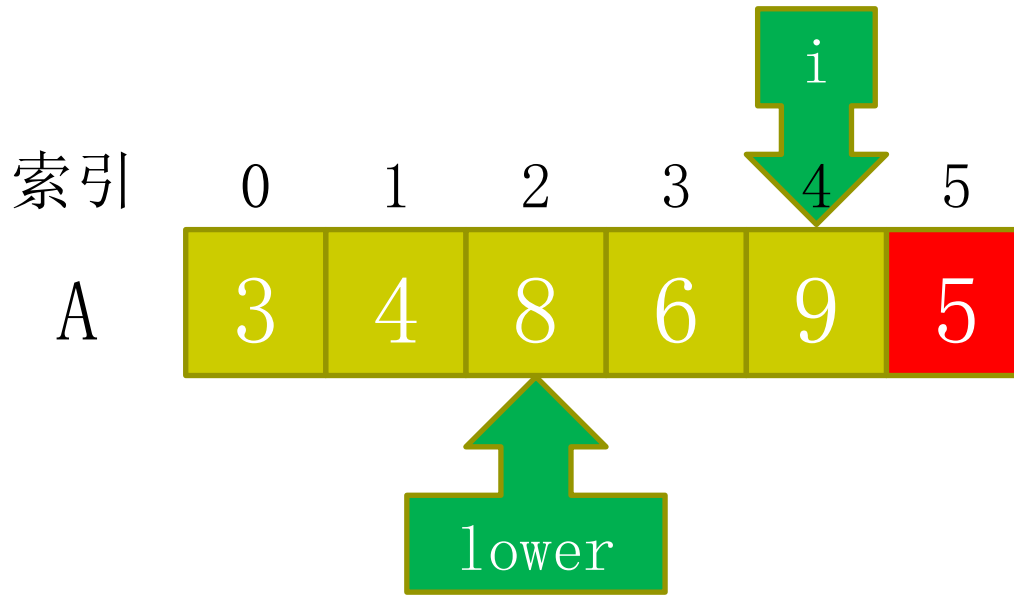
Partition ([8 6 3 4 9 5])



- $A[i] = A[3] < 5$
- $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

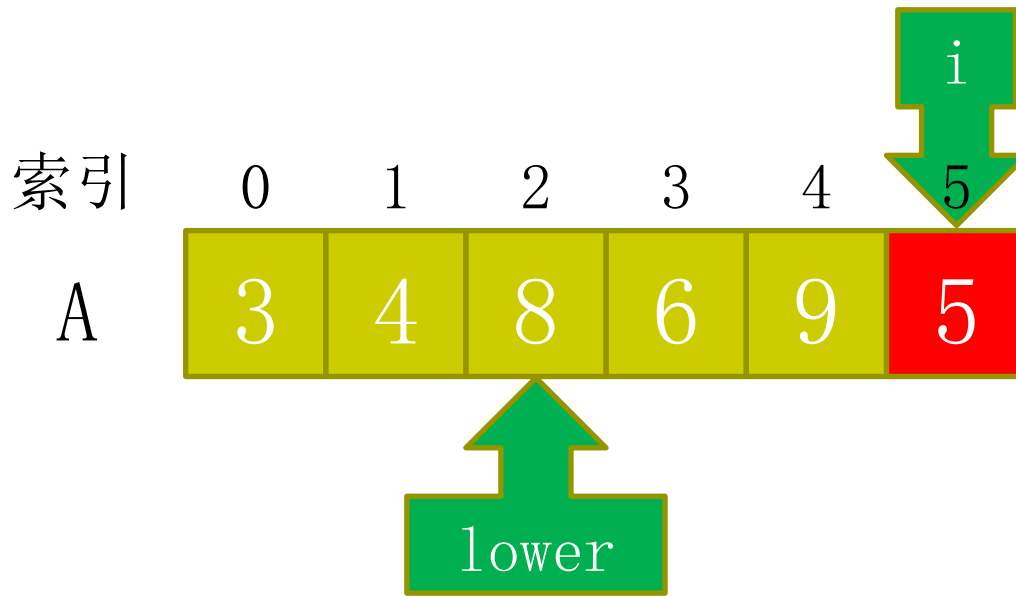
Partition ([8 6 3 4 9 5])



- $A[i] = A[4] \geq 5$
- $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

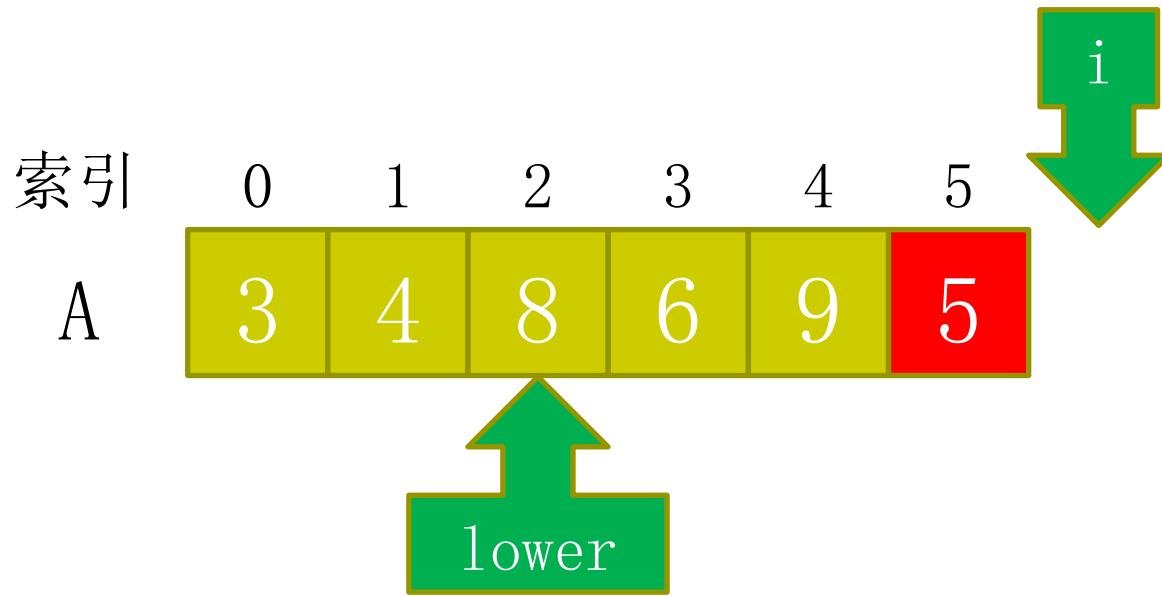
Partition ([8 6 3 4 9 5])



- $A[i] = A[5] \geq 5$
- $i++$

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

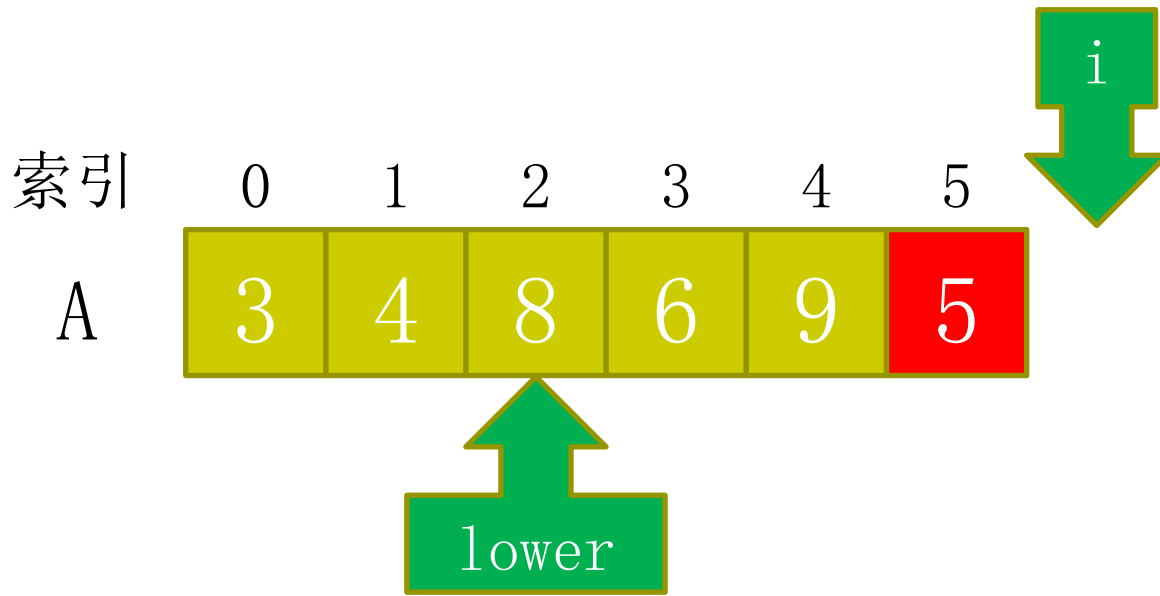
Partition ([8 6 3 4 9 5])



- $i = 6 = \text{len}(A)$
- 停止循环

```
for i := 0; i < len(A); i++ {  
    //建议: 如果A[i]小于pivot  
    //则交换A[i]与A[lower]并更新lower  
}
```

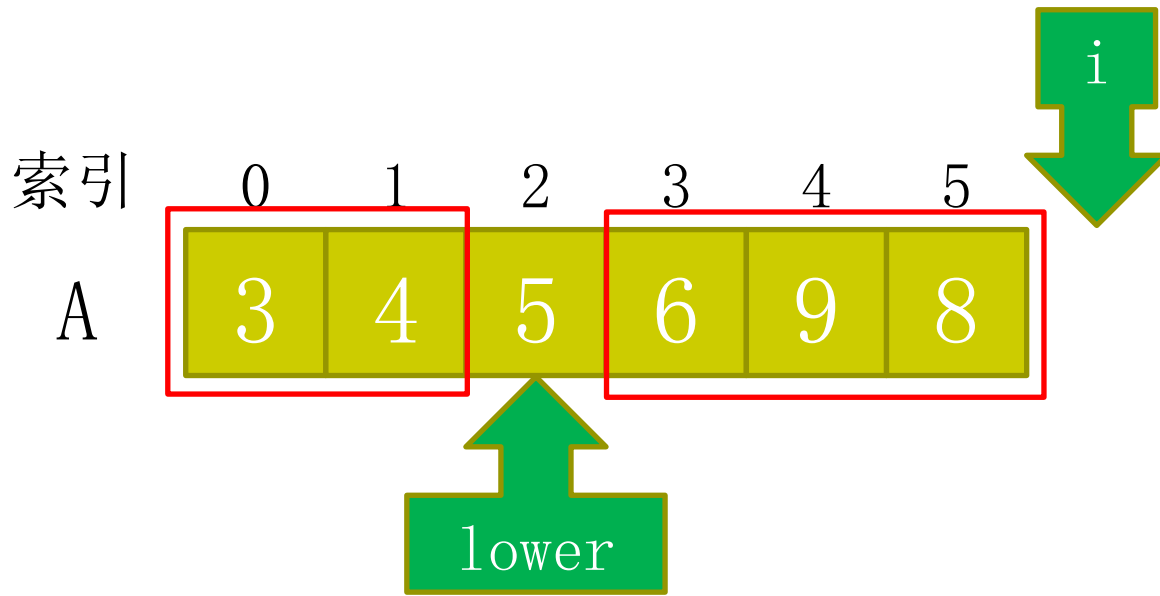
Partition ([8 6 3 4 9 5])



- 交换 $A[lower]$ 与 $A[len(A)-1]$

```
A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
return A[0:lower], A[lower+1:len(A)]
```

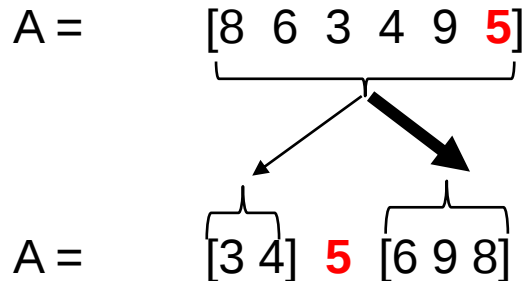
Partition ([8 6 3 4 9 5])



- 分别返回 $lower$ 之前的切片和之后的切片

```
A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
return A[0:lower], A[lower+1:len(A)]
```

Partition示例



索引 0 1 2 3 4 5

i ↓
[8 6 3 4 9 5] // lower, i 初始值为0, len(A)-1为5
lower ↑
// A[0] >= A[5], i = i + 1

i ↓
[8 6 3 4 9 5] // A[1] >= A[5], i = i + 1
lower ↑

i ↓
[8 6 3 4 9 5] // A[2] < A[5], A[i]与A[lower]交换
lower ↑ // lower = lower + 1, i = i + 1

i ↓
[3 6 8 4 9 5] // A[3] < A[5], A[i]与A[lower]交换
lower ↑ // lower = lower + 1, i = i + 1

i ↓
[3 4 8 6 9 5] // A[4] >= A[5], i = i + 1
lower ↑

i ↓
[3 4 8 6 9 5] // A[5] >= A[5], i = i + 1
lower ↑

i ↓
[3 4 5 6 9 8] // i > 5 循环结束, 交换标杆和A[lower]
lower ↑ // lowerA = A[0:lower] = A[0:2] = {3 4}
// upperA = A[lower+1:len(A)]
// = A[3:6] = {6 9 8}

```
func partition(A []int) ([]int, []int) {
    lower := 0
    for i := 0; i < len(A); i++ {
        // 此处插入你的代码
        // 建议: 如果A[i] < 标杆值,
        // A[i]与A[lower]交换并更新lower
    }
    A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
    return A[0:lower], A[lower+1:len(A)]
}
```

现场练习：请逐步排序[8 6 3 4 9 5]

- **输入**：6元素整数数组d = [8 6 3 4 9 5]
- **输出**：6元素整数数组d = [3 4 5 6 8 9]

- 元素从小到大排好序了

- **步骤**：调用fastsort(d)。

函数fastsort(A)的计算过程如下：

1. 基线情况（base case）：如果A只包含0个元素（如[]）或1个元素（如[5]），fastsort(A)结束
2. 选择A的最后一个元素作为标杆元素（pivot）
3. 调用划分函数partition
 - 将小于标杆值的元素放入lowerA子数组（称为小数组）中，将大于标杆值的元素放入upperA子数组（称为大数组）中
4. 递归调用fastsort(lowerA)
5. 递归调用fastsort(upperA)

d = [8 6 3 4 9 5]

d =

d =

d =

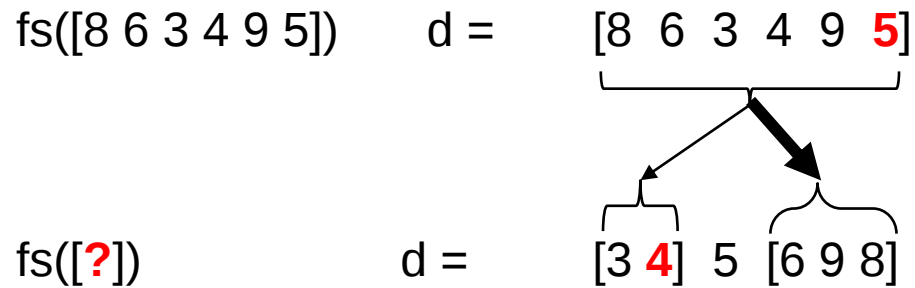
d =

d =

d =

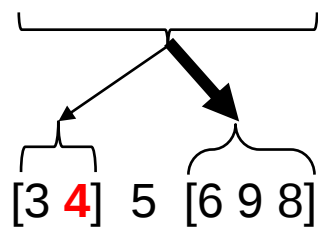
d = 3 4 5 6 8 9

现场练习：请逐步排序[8 6 3 4 9 5]



现场练习：请逐步排序[8 6 3 4 9 5]

fs([8 6 3 4 9 5]) d = [8 6 3 4 9 **5**]



fs([3 4]) d = [3 **4**] 5 [6 9 8]

fs([3]) d = [3] 4 [] 5 [6 9 8]

fs([]) d = 3 4 [] 5 [6 9 8]

fs([6 9 8]) d = 3 4 5 [6 9 **8**]

fs([6]) d = 3 4 5 [6] 8 [9]

fs([9]) d = 3 4 5 6 8 [9]

d = 3 4 5 6 8 9

4. 现场实现 你的fastsort.go

- 实验要求：
 - 利用右侧代码框架
 - 使用你的云平台上的数组**替换待排序数组d的值**
 - 只在**标红处**填充你的代码
 - 如果 **$A[i] < \text{标杆值}$** ，则 **$A[i]$ 与 **$A[\text{lower}]$ 交换并更新 **lower******
 - 提交fastsort.go文件
 - 提交lowerA, pivot, upperA的值
 - 由于采用计算机自动评分，请严格按照云平台示例填写内容

```
package main // fastsort.go
import "fmt"
var d [6]int = [6]int {8,3,6,7,2,1} //替换该数组
var s []int = d[0:6]
func main() {
    fastsort(s)
    fmt.Println(s)
}

func fastsort(A []int) {
    if len(A) < 2 { // 判断是不是基线情况
        return // 是，退出
    }
    lowerA, upperA := partition(A)
    fastsort(lowerA) // 递归排序小数组
    fastsort(upperA) // 递归排序小数组
}

func partition(A []int) ([]int, []int) {
    lower := 0
    for i:= 0; i<len(A); i++ {
        // 此处插入你的代码
        // 建议: 如果 $A[i] < \text{标杆值}$ ,
        //  $A[i]$ 与 $A[\text{lower}]$ 交换并更新 $\text{lower}$ 
    }
    A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
    return A[0:lower], A[lower+1:len(A)]
}
```

4. 现场实现 你的fastsort.go

- 实验要求：
 - 提交fastsort.go文件
 - 手动计算并提交lowerA, pivot, upperA的值
 - 由于采用计算机自动评分，请严格按照云平台示例填写内容

作业提交链接

https://course.things.ac.cn:10088/exp/basic_programming

作业提交截止时间

[2024/3/24 23:30](#)

请提交

请把fastsort.go中的待排序数组设成如下值: [2, 5, 3, 4, 1, 9]

• fastsort.go 未选择任何文件

请使用给定的数组，填写fastsort函数中按partition函数执行顺序依次计算的lowerA, pivot, upperA。由于采用计算机自动评分，请**严格**按如下示例填写内容

假如第1次计算的结果是lower = [2,3,5], pivot=6, upper=[7]，则在第1行的第1个空格填写2,3,5（数组各数字之间使用1个英文逗号隔开）第2个空格填写6 第3个空格填写7

假如第2次计算的结果是lower = [2,3], pivot=5, upper=[], 则第2行的第1个空格填写2,3 第2个空格填写5 第3个空格不填写任何内容

假如partition仅执行了3次或4次，则只填前3行或前4行，其它行数不需要填写任何内容

	Lower	pivot	Upper
1:			
2:			
3:			
4:			
5:			