

Announcements

Thursday Extras,

None next week

Extra Credit Opportunity

Play: Love and Information in Bucksbaum/Flanagan

Fri, Sat @ 7:30 pm

Sun @ 2:00 pm

CSC 207 Notes

An idea is gelling for an extra credit Supplementary Problem 2

A new problem is likely to be posted over the weekend

Pictorial Loop Invariants and Test 2

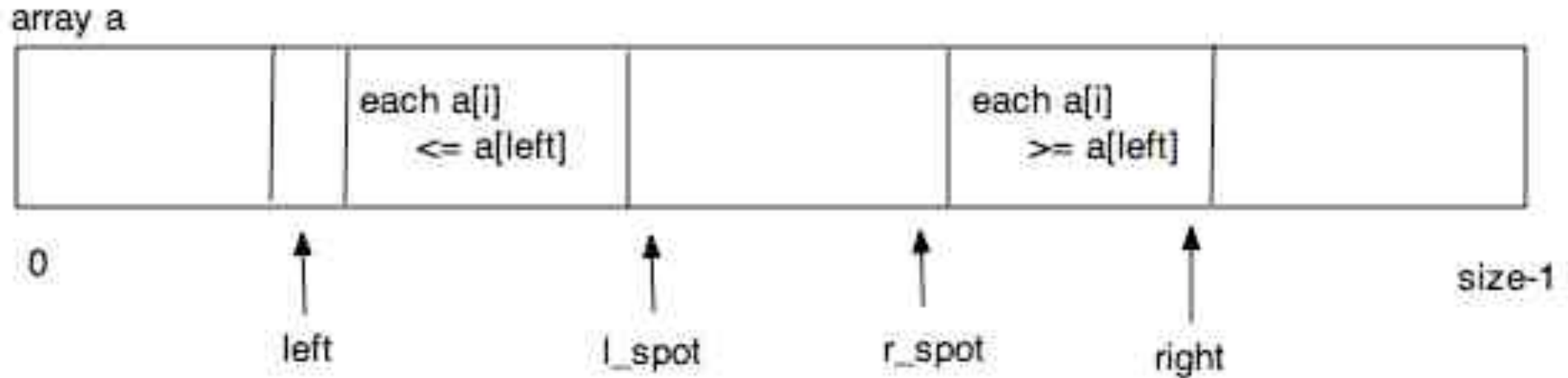
The lab was returned Wed., when Test 2 Part A was due

Some grading seemed quite generous

We'll talk further about the lab today

If you want to revise Test 2, Problem 1, based on today's discussion or conversations in the office, a revision will be accepted if submitted by Wednesday's class meeting

Consider the following loop invariant for partition (from the lab):

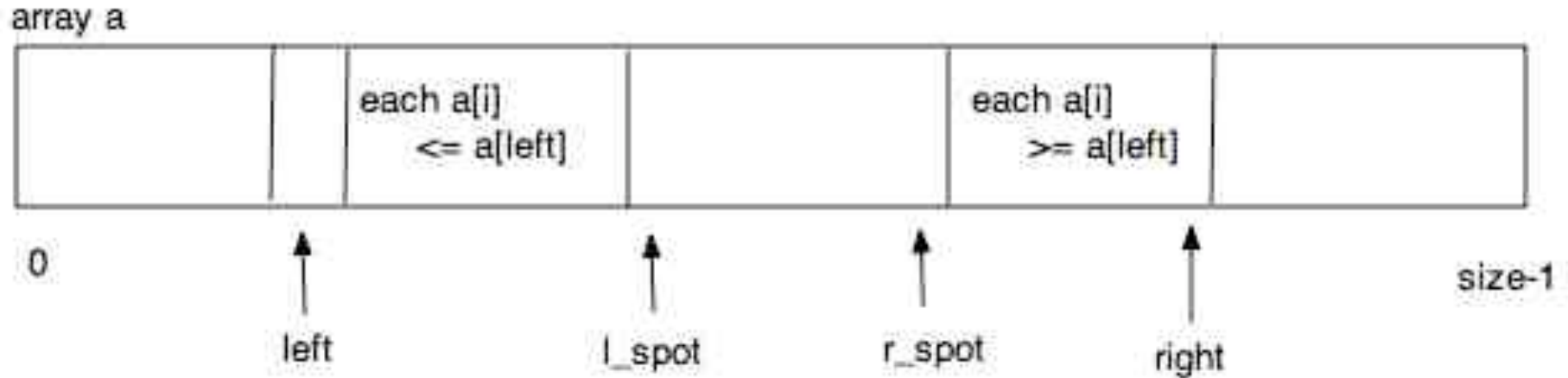


Three processing approaches are considered

1. Initialize $\text{l_spot} = \text{left}$, $\text{r_spot} = \text{right}$
2. Initialize $\text{l_spot} = \text{left} + 1$, $\text{r_spot} = \text{right}$
3. Store left , $[\text{right}]$ as local variables old_left , $[\text{old_right}]$, store the pivot value in variable pivot
increment left , and use left and right to mark the boundaries of the unknown middle section

A. 1 only B. 2 only C. 3 only D. 2 or 3 E. other

Consider the following loop invariant for partition (from the lab):



Is the following code appropriate for the main loop?

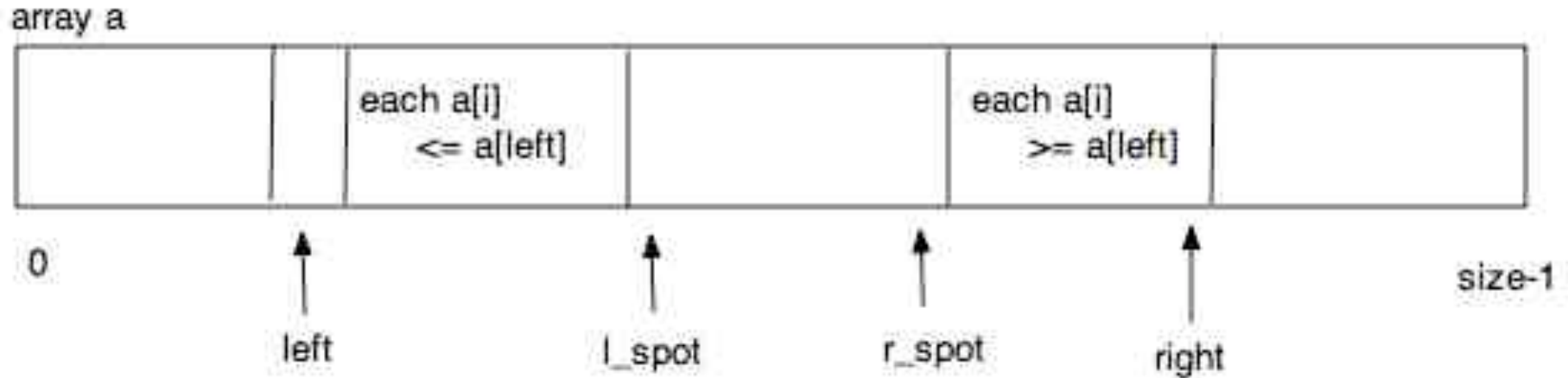
```
while (l_spot <= r_spot)
{
    while ((l_spot <= r_spot) && a[left] <= a[r_spot]))
        r_spot--;
    while ((l_spot <= r_spot) && a[left] > a[l_spot]))
        l_spot++;
    if (a[r_spot] < a[l_spot])
        swap a[r_spot--], a[l_spot++]
}
```

A. Yes, great

B. No, inefficient

C. No, wrong alg.

Consider the following loop invariant for partition (from the lab):



Is the following code appropriate for the main loop?

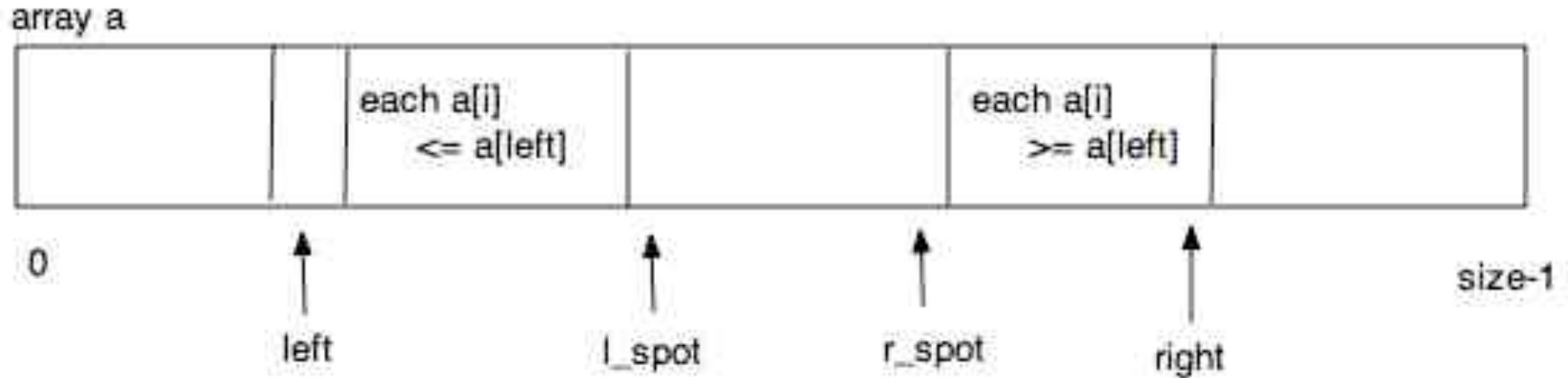
```
while (l_spot <= r_spot)
{
    while ((l_spot <= r_spot) && a[left] <= a[r_spot]))
        r_spot--;
    while ((l_spot <= r_spot) && a[left] >= a[l_spot]))
        l_spot++;
    if (a[r_spot] < a[l_spot])
        swap a[r_spot], a[l_spot]
}
```

A. Yes, great

B. No, inefficient

C. No, wrong alg.

Consider the following loop invariant for partition (from the lab):



Is the following code appropriate for the main loop?

```
while (l_spot <= r_spot)
{
    if (a[left] < a[r_spot])
        r_spot--;
    else if (a[left] > a[l_spot])
        l_spot++;
    else if (a[r_spot] < a[l_spot])
        swap a[r_spot], a[l_spot]
}
```

A. Yes, great

B. No, inefficient

C. No, wrong alg.

Suppose a tree is based on the following TreeNode definition

```
String firstName;  
String lastName;  
TreeNode left;  
TreeNode right;
```

Nodes are ordered by last name

(if last names are the same, first name is used)

What is printed by the following, when given the root of the tree as initial parameter?

```
private void print (TreeNode root) {  
    if (base != null) {  
        print (root.left);  
        System.out.println(root.firstName+" "+first.lastName);  
        print (root.right);  
    }  
}
```

- A. Nodes in ascending order
- B. Nodes in descending order

- C. Nodes in other order
- D. Compile/run-time error

Suppose a tree is based on the following TreeNode definition

```
String firstName;  
String lastName;  
TreeNode left;  
TreeNode right;
```

Nodes are ordered by last name
(if last names are the same, first name is used)

A search is to be done to find the last name that comes last in alphabetical order. How should this be done?

- A. From the root, go left as long as possible
- B. From the root, go right as long as possible
- C. Do a search of the entire tree
- D. Something else

Suppose a tree is based on the following TreeNode definition

```
String firstName;  
String lastName;  
TreeNode left;  
TreeNode right;
```

Nodes are ordered by last name
(if last names are the same, first name is used)

A search is to be done to find the first name that comes last in alphabetical order. How should this be done?

- | | |
|--|-----------------------------------|
| A. From the root, go left as long as possible | C. Do a search of the entire tree |
| B. From the root, go right as long as possible | D. Something else |

Suppose a tree is based on the following TreeNode definition

```
String firstName;  
String lastName;  
TreeNode left;  
TreeNode right;
```

Nodes are ordered by last name
(if last names are the same, first name is used)

A search is to be done to find the first name that comes last in alphabetical order.

Let's write the code together.