CS302 - Data Structures

using C++

Topic: Recursion - Examples

Kostas Alexis

CS302 - Data Structures

using C++

Topic: The Binary Search

Kostas Alexis

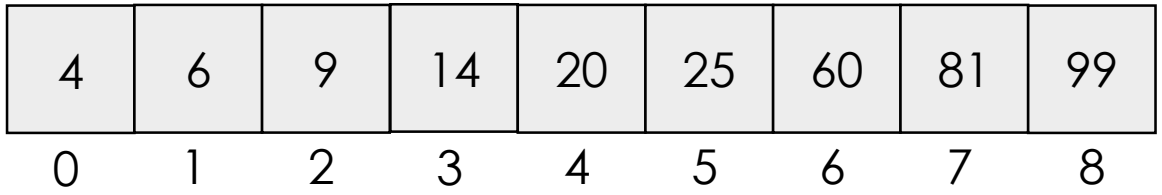
AUTONOMOUS ROBOTS LAB

The Binary Search

- Assumes and exploits that the input array is **sorted**.

The Binary Search

- Assumes and exploits that the input array is **sorted**.



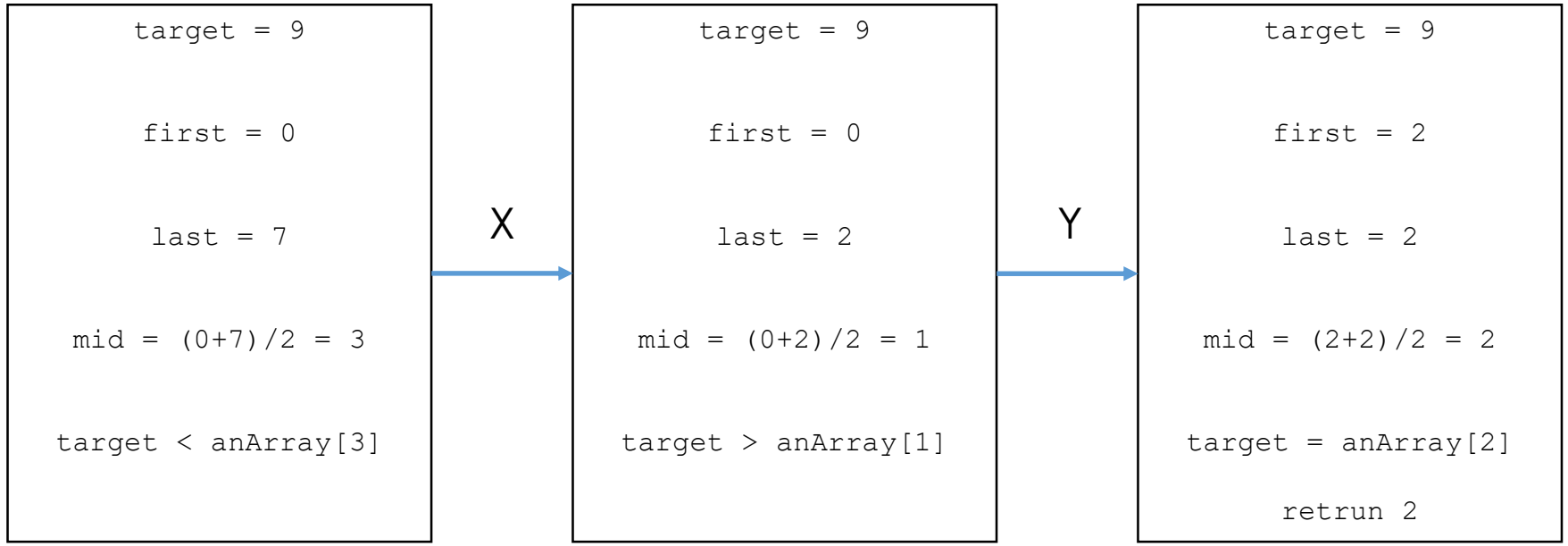
case 1: target == Array[mid] // we are done
case 2: target < A[mid] // we know we have to search on left half
case 3: target > A[mid] // we know we have to search on right half

The Binary Search

- Consider details before implementing the algorithm
  - How to pass half of **`anArray`** to recursive calls of **`binarySearch`**?
  - How to determine which half of array contains target?
  - What should base case(s) be?
  - How will **`binarySearch`** indicate the result of search?

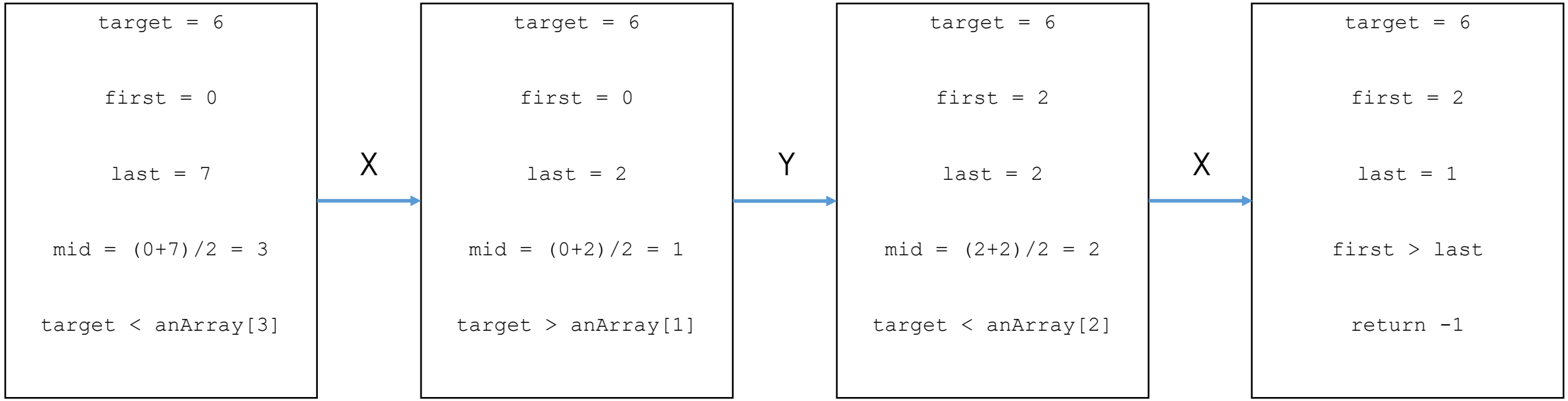
The Binary Search

- Box traces of **`binarySearch`** with **`anArray`** = <1, 5, 9, 12, 15, 21, 29, 31>: (a) a successful search for 9



The Binary Search

- Box traces of **`binarySearch`** with **`anArray`** = <1, 5, 9, 12, 15, 21, 29, 31>: (b) an unsuccessful search for 6



The Binary Search

```c
#include <stdio.h>

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid+1, r, x);
   }

   // We reach here when element is not
   // present in array
   return -1;
}
```

```c
int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d",
                                                    result);
   return 0;
}
```

CS302 - Data Structures

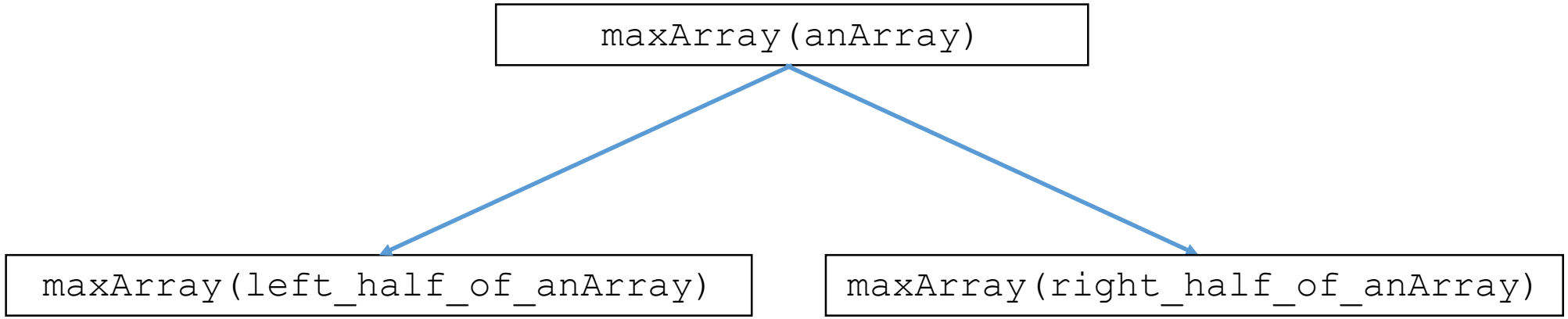
using C++

Topic: Finding Largest Value in an Array

Kostas Alexis

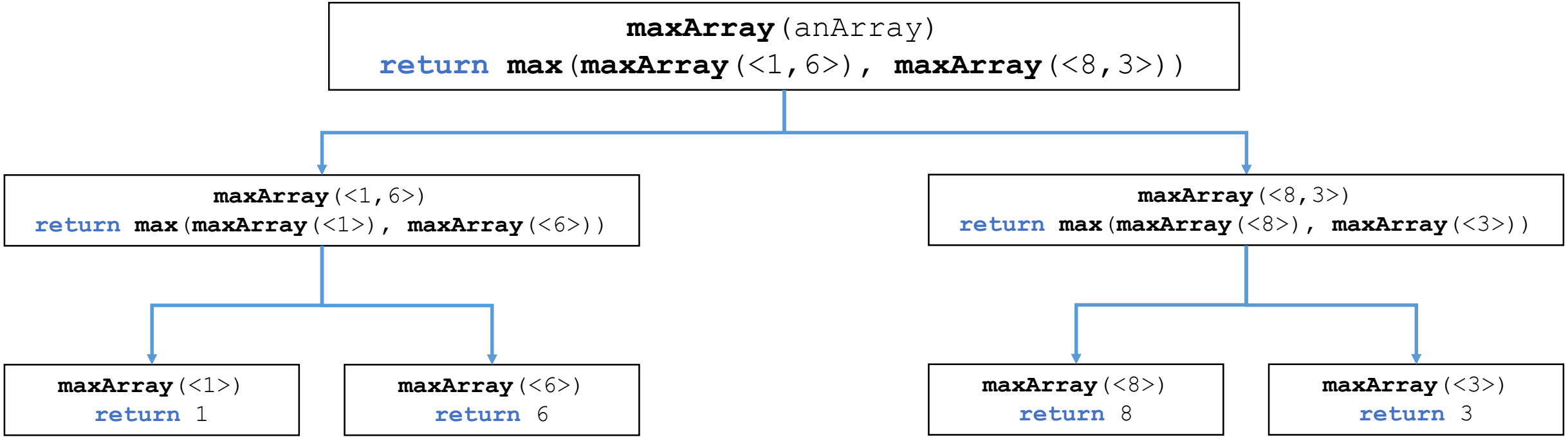
Finding the Largest Value in an Array

- Recursive solution to the largest-value problem



Finding the Largest Value in an Array

- The recursive calls that **maxArray**(<1,6,8,3>) generates



CS302 - Data Structures

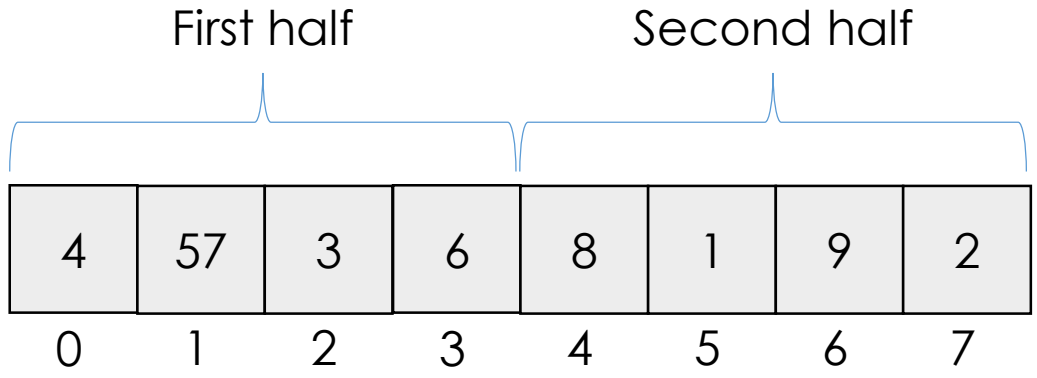
using C++

Topic: Finding k-th Smallest Value

Kostas Alexis

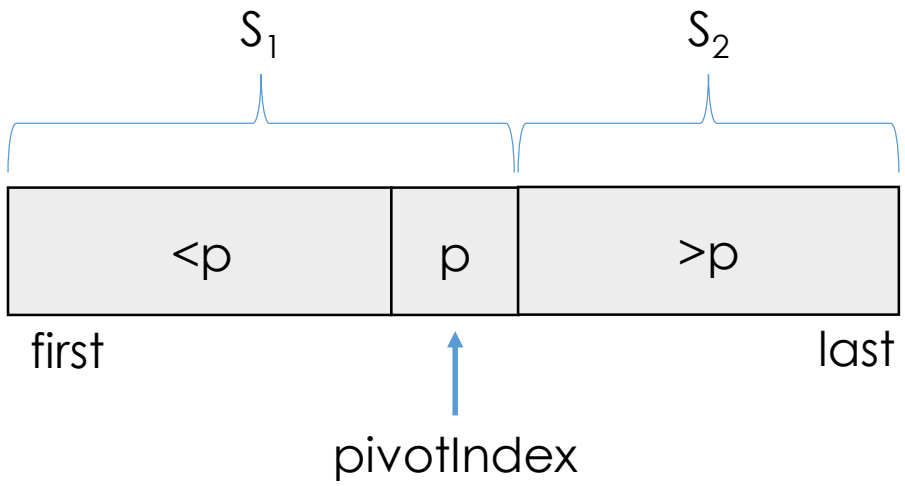
Finding k-th Smallest Value of Array

- Recursive solution proceeds by
  1. Selecting pivot value in array
  2. Cleverly arranging/partitioning values in array about pivot value
  3. Recursively applying strategy to one of partitions
- A sample array



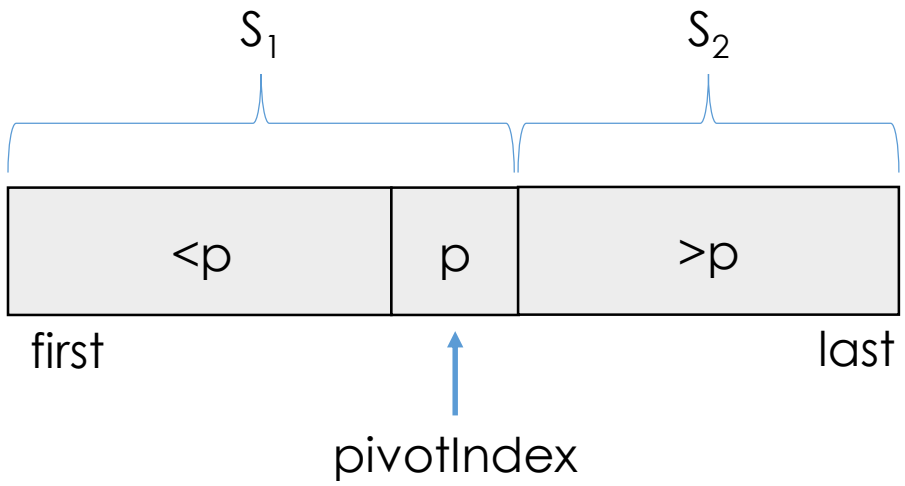
Finding k-th Smallest Value of Array

- Recursive solution proceeds by
  1. Choose a **pivot** value at random: generate a random number in the range of [firstIndex,lastIndex]
  2. Rearrange the list in a way that all elements less than the pivot are on left side of pivot and others on the right.
  3. Return the index of the pivot element



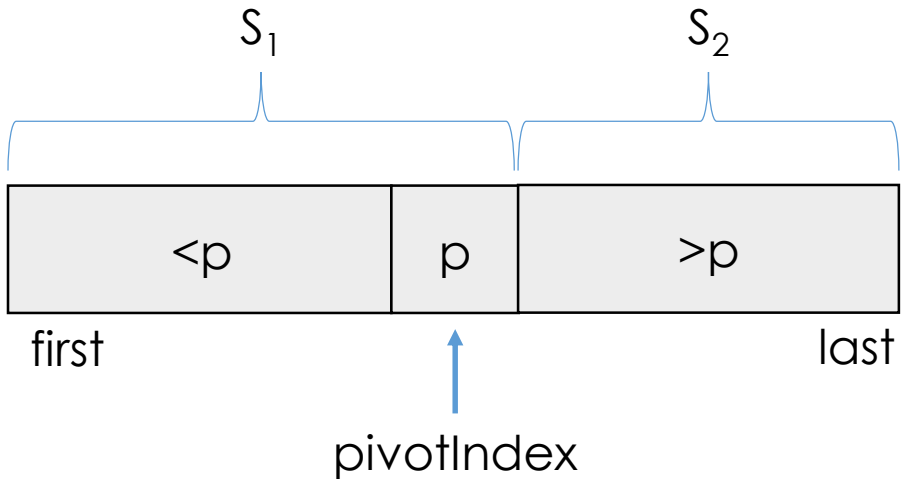
Finding k-th Smallest Value of Array

- Recursive solution proceeds by
  1. Choose a **pivot** value at random: generate a random number in the range of [firstIndex,lastIndex]
  2. Rearrange the list in a way that all elements less than the pivot are on left side of pivot and others on the right.
  3. Return the index of the pivot element



Finding k-th Smallest Value of Array

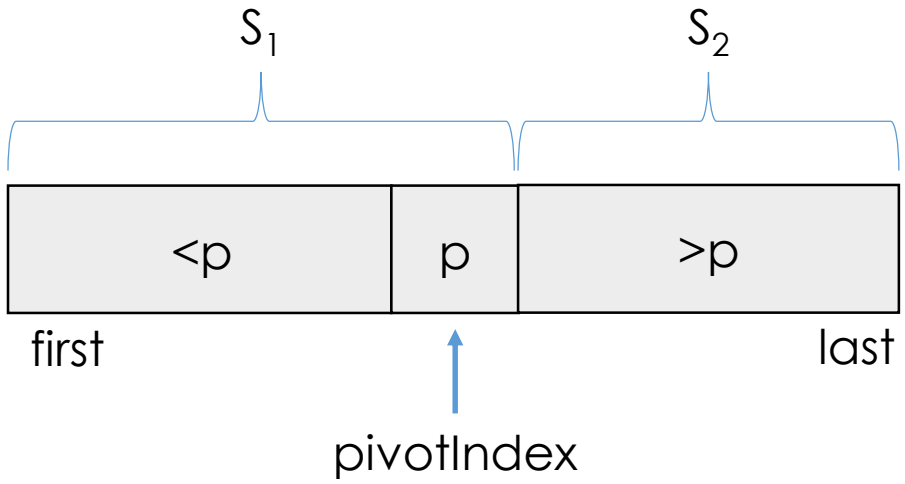
- After partitioning there are 3 cases
    1. **case 1:** k == pivot // we found the k-th smallest item
    2. **case 2:** k < pivot // the k-th smallest item is on the left side of the pivot, that's is why we can discard the other subarray
    3. **case 3:** k > pivot // k-th smallest item is on the right side of the pivot



Finding k-th Smallest Value of Array

- After partitioning there are 3 cases
    1. **case 1:** k == pivot // we found the k-th smallest item
    2. **case 2:** k < pivot // the k-th smallest item is on the left side of the pivot, that's is why we can discard the other subarray
    3. **case 3:** k > pivot // k-th smallest item is on the right side of the pivot

  Base case is that k == pivot



Thank you