A detailed explanation of the code has been included in the code itself using comments.

What algorithms did I use?
I used a custom-made algorithm that utilizes shifting swapping and searching functionalities. The shifting process is called recursively with a new index. Elements are moved recursively at the beginning of an array if they are smaller than or equal to the key. And since before starting this process I always move the key at the beginning of the array first, the array will always be sorted in a way that to the left of the key, all the smaller numbers reside, and to the right of the key we will find all the bigger numbers than the key.

Why this algorithm?
I chose this algorithm since it allows the key to be in any place of the array. Either way, I will always be swapping positions so that the key is at the beginning of the array before performing the shifting process.

What other algorithms could I have used here?
Another option that could have been used here is using the key as a partition. As the name suggests it would have used the key to split the array into two halves. On one side the smaller elements would be placed and on the other side the larger numbers.

What are the differences?
The difference here is that I do not use the key as a partition in my approach. I instead place the key at the beginning of the array so that I can then add all the smaller or equal elements right behind the key. Which would end up with all the smaller elements being left off the key while all the bigger elements remain on the right side of the key.

Complexity analysis?
The complexity of this program is O(n^2) due to the recursive call that shifts all elements of the array. However, when refactoring the code to be iterative instead, the complexity will probably remain the same. Since shifting the elements in the array would most likely give us a situation where could find a nested loop unless we change the approach.

Difference between iterative and recursive
Like explained earlier the recursive approach here shifts the entire array to the right and inserts the elements smaller than the key at the beginning of the array. This can cause the code to run slower than an iterative approach. An iterative approach here would most likely include a nested loop to swap the elements to the beginning of the array. A recursive approach usually consumes more memory but are usually result in a cleaner code.

Code:

```
#include <iostream>

using namespace std;


// Defining a maximum size for the array so that we can later shift the elements inside of the array.
```

```
const int MAX_SIZE = 100;


// This function prints out the array
void printArray(int array[], int size) {
   // A for loop that loops through each item of the array
   for (int i = 0; i < size; i++) {
      // Printing out every element of the array
      cout << array[i] << " ";
   }
   cout << endl;
}



// This function is to find the index  of the key provided so that I can later on swap the key with the element at the beginning of the array. This way I can then add all elements smaller than the key at the beginning of the array right behind the key.
int findingIndexofKey(int key, int array[], int size) {
   int KeyIndex = 0;
   // We loop through each item of the array
   for (int i = 0; i < size; i++) {
      // When we find the key in the array we then store it's index in a variable
      if (array[i] == key) {
         KeyIndex = i;
         return i;
      }
   }
   // Checking if the key is present in the array
   return -1;
}


// This function shifts the array to the right whenever we find an element that is smaller than the key and add it to the beginning of the array
```

```
void task4(int arr[], int key, int& size, int index = 0) {

   // This is the base case which is when the index is out of range
   if (index >= size) return;

   // An if condition that stores the current element in a temporarily variable if it is less than or equal to the key
   if (arr[index] <= key) {
      int temp = arr[index];

      // This loop shifts all elements to the right to make space at the beginning of the array for our element
      for (int j = index; j > 0; j--) {
         arr[j] = arr[j - 1];
      }
      // We now place the element previously stored at the beginning of the array
      arr[0] = temp;

      // Now we recursively call our function again with our next index
      task4(arr, key, size, index + 1);
   }
   else {
      // Same call here but inside of an else statement which means that the element was bigger than the key so we do nothing but just call the function again with the next index
      task4(arr, key, size, index + 1);
   }
}

int main() {
   // Defining and initializing our variables and array
   int temp;
   bool swapped;
```

```
int KeyIndex{};
int key = 12;
int array[MAX_SIZE] = { 6, 4, 25, 12, 106, 7, 13, 46, 1, 96 };
int size = 10;

// Printing out the original array
cout << "Array before applying changes: ";
printArray(array, size);

// Calling the function to find our index of the key and storing it in a variable
KeyIndex = findingIndexofKey(key, array, size);

// If the key is not found in the array the program does not execute the swap function
if (KeyIndex == -1) {
   cout << "Key was not found in the array." << endl;
   return 0;
}

// if the key is found in the array we proceed normally
// Here we swap the element at the beginning of the array with the key
swap(array[0], array[KeyIndex]);
swapped = true;

// Now we call the function to perform the rearrangement of the array
task4(array, key, size);

// Printing out the end result
cout << "Array after applying changes: ";
printArray(array, size);

return 0;
```

}