Wednesday, September 29, 2021

QuickSort Algorithm Example in Java using Recursion - Tutorial

The Quicksort algorithm is one of the very popular sorting algorithms in programming, often used to sort a large array of numbers. Though there is numerous algorithm available to sort a list of objects, including integer, string, and floating-point number, quicksort is best for general purpose. It's a divide and conquers algorithm, where we divide the given array with respect to a particular element, known as **'pivot'** such that the lower partition of the array is less than the pivot and upper partition elements of the array are higher than the pivot. The Quicksort is also one of the best examples of recursion, a key programming technique to solve Algorithmic problems. This algorithm is naturally recursive because it sorts the large list by dividing it into smaller sub-list and then applying the same algorithm to those.

The base case of recursion is when a list contains either one or zero elements, in that case, they are already sorted. Quicksort is well ahead with primitive sorting algorithms like Insertion sort, selection sort, and Bubble sort. The average time complexity of quicksort is `O(N log N)`, while in the worst case its performance is similar to bubble sort, I mean `O(n^2)`.

Apparently, the worst case of quicksort is the best case of insertion sort, where they have to sort an already sorted list. In this article, we will learn *how to implement a quicksort algorithm in Java using recursion*.

We will also learn how quicksort works, and how it sorts a large list of unsorted numbers. In the last section, we will revisit some important things about quicksort.

Btw, if you are new to the Data Structure and Algorithm field and not familiar with essential searching and sorting algorithms like Quicksort, I suggest you take a look at the **Data Structures and Algorithms: Deep Dive Using Java** course on Udemy. One of the better courses to master algorithms and data structure in quick time.

1. How do the QuickSort Algorithm works?

An old saying is, a picture is worth more than a thousand words. This is completely true in the case of understanding *how the sorting algorithm works*.

In the past, I have understood Insertion sort, Bubble sort, and Radix sort much better by following a diagram rather than reading about it.

That's why I am sharing this diagram which explains how the quicksort algorithm works, how it sort a list of integers. It's similar to a flowchart but doesn't use the notation flowchart uses, instead it practically shows how sorting happens.

Once you go through this diagram, read the explanation, it will make more sense.

from an Integer Array in Ja...

Difference between Method and Constructor in Java ...

3 ways to sort a List in Java 8 and 11 - Example T...

5 Free SQL Books For Beginners and Experienced - D...

How Binary Search Algorithm Works? Java Example wi...

How to solve java.lang.NoClassDefFoundError: org/s...

10 Examples to DateTimeFormatter in Java 8 to Pars...

How to declare and Initialize two dimensional Arra...

Top 10 Books Every Programmer Should Read

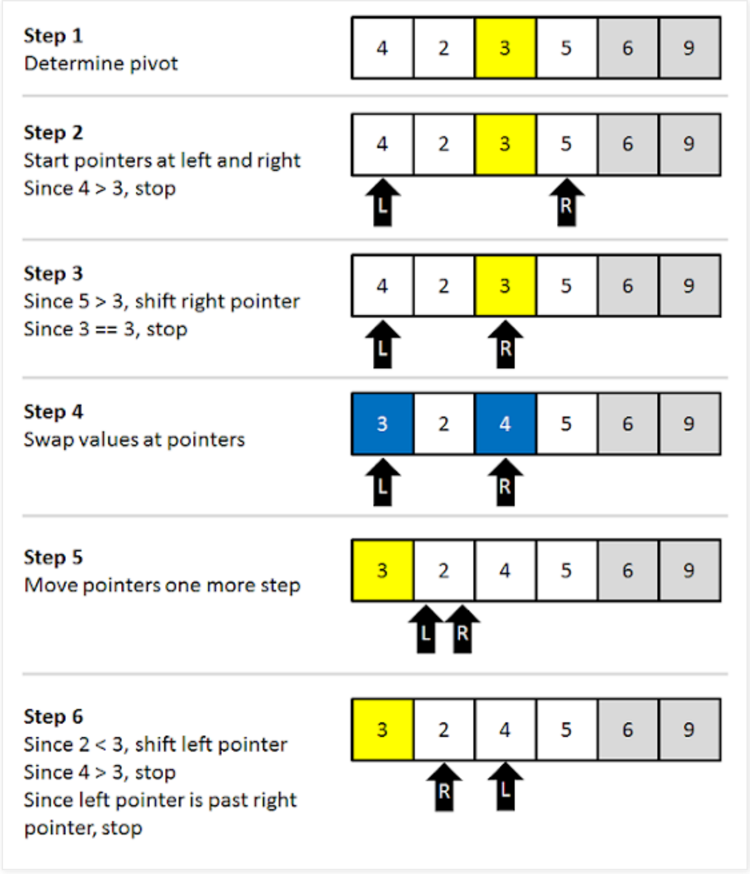
How to use Spread operator and Rest parameter in J...

What is Variable and Function Hoisting in JavaScri...

What is Destructuring in JavaScript? Example Tutorial

Difference between first level and second level ca...

Difference between IN, OUT, and INOUT parameters i...



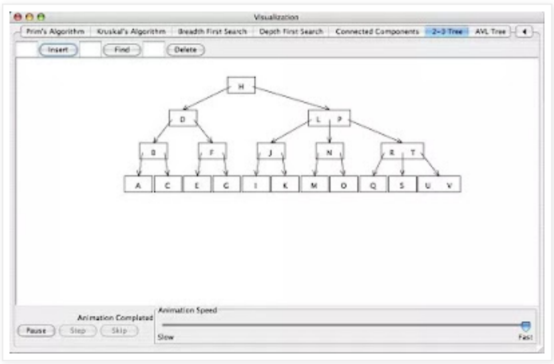
As I told before *QuickSort is a recursive algorithm*, it divides the big list into smaller list around pivot until those lists are individually sorted. The first step of the Quicksort algorithm is to determine pivot, it's general practice to choose the middle element of the array as a pivot, but you are free to choose any index.

Now you have two lists, the next step is to ensure that the left partition only contains numbers less than the pivot and the right partition only contains numbers greater than the pivot.

We start the pointer from the left and right of the pivot, and as soon as the left pointer encounter 4, it stops because 4 is greater than 3. Similarly, the right pointer stops at 3 because all numbers on the right side of the list are greater than 3.

Now it's time to swap, so 3 takes place of 4 and vice-versa. Now, we move the pointer to one more step, since `2 > 3`, the left pointer shifts but since `4 > 3`, it stopped.

Since the left point is also past the right pointer it stopped. Now if we repeat this process one more time, the list will be sorted. If you still don't get the algorithm then I suggest you join the **Visualizing Data Structures and Algorithms in Java** course on Udemy. A special course that will teach you data structures and algorithms in Java through animations and implementations.



- Best Book to Lear
Beginners
- 5 Books to Learn S
and Core Spring
- 2 books to learn H
Java developers
- 12 Advanced Java
Programming Boo
Experienced Progr
- 5 Free JavaScript
download
- 5 Books to Improv
Skill
- Books Every Prog
Should Read
- Top 10 Computer
Books
- 10 Free Java Prog
Books
- 5 Books to Learn J
- Books Every Prog
Should Read
- Top 10 Computer
Books

Followers

Followers (1477) Next

Follow

Privacy

- Privacy Policy
- Terms and Conditio

Subscribe

Get new posts

Enter your e

Subscri

How to create a custom tag in JSP? Steps and Example

Static Class, Static Variable, Keyword, and Static...

How to use lifecycle methods in functional compone...

How to compare two Arrays in Java to check if they...

What is Redux Thunk in React.js? Example tutorial

How to Convert or Print Array to String in Java? E...

How to Format Date to String in Java 8 [Example Tu...

What is Redux and how to use it? Example Tutorial

Can you Overload or Override main method in Java? ...

How to sort HashMap by values in Java 8 [using Lam...

Top 20 Essential Java Interview Questions with Ans...

10 Advanced Programming and Development Books for ...

4 Examples of Stream.collect() method in Java 8

Difference between List and ArrayList

2. The Concept of Pivot and Partition

Though we often select a middle element of the array as a pivot, there is no such rule and pivot can be an element of the given array. You can even consider the first element as the pivot in every partition.

It's experienced that choice of pivot affects the distribution of the elements in partitioning and affects the complexity of the quicksort algorithm.

As per the rule of partition, numbers in the lower partition should be less than the pivot and upper partition numbers should be higher than the pivot. The running time of partition logic is linear.

3. The complexity of Quicksort Algorithm Explained

On average Quicksort Algorithm has the complexity of `O(NlogN)` and in the worst case, it has $O(n^2)$ when the elements of the input array are already sorted in ascending or descending order.

The good thing about Quicksort is that it's an *in-place algorithm*, which means it does not take any additional space, except those used by method stack.

By the way, there are **some tricks to improve the performance of quicksort**, even in the worst case. As suggested in one of the best algorithm design books, **The Algorithm Design Manual,** by Steven Skiena, you can apply the following recommendation to improve your quicksort algorithm implementation.

3.1. Randomization

You can avoid the worst-case performance of $O(n^2)$ when sorting nearly-sorted data by random permutation of keys. Though it incurs some cost of permutation still gives better performance than $O(n^2)$

3.2. Leave small sub-arrays for Insertion sort

finish Quicksort recursion and switch to insertion sort when fewer than 20 elements:

There is a drawback of using recursion to implement a quicksort algorithm, It will not scale because JVM has no tail call optimization, it will simply grow the method call stack to something proportional to the array to sort, and it will fail for the very large array.

Btw, if you have trouble understanding how we calculate the time and space complexity of an algorithm or solution, I suggest you check out **Algorithms and Data Structures - Part 1 and 2**, a two-part series in Pluralsight to learn how to calculate time complexity. It's an excellent course for beginners.

Reference Va...

How to Increasing Heap Size of Java application i...

The Ultimate Guide of Remote Debugging in Java usi...

How to Convert java.util.Date to LocalDate in Java...

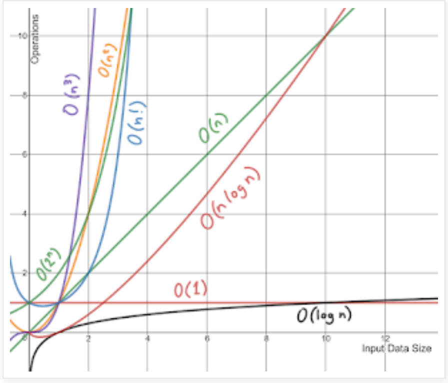
Can you take training after passing OCMJEA 6 Exam ...

How to parse String to LocalDate in Java 8 - Date...

▶ August (65)

▶ July (219)

▶ June (41)



4. Java Program to Sort Integer Array using QuickSort Algorithm

Here is our recursive implementation of the QuickSort sorting algorithm. We have used it to sort an array of randomly distributed integers. We have two sets of inputs, one which doesn't contain any repeated numbers and the other which contains duplicates.

The Logic of quicksort is encapsulated in method `recursiveQuickSort(int[] array, int startIdx, int endIdx)` and `partition(int[] array, int left, int right)`, which implements partitioning logic.

In order to hide implementation details, we have only exposed a convenient static utility method called `quickSort(int[] array)`, which takes an integer array and sorts that in place.

```java
package test;

import java.util.Arrays;


/**
* Java program to Sort integer array using QuickSort algorithm using recursion.
* Recursive QuickSort algorithm, partitioned list into two parts by a pivot,
* and then recursively sorts each list.
* @author Javin Paul
*/
public class QuickSort{

    public static void main(String args[]) {

        int[] input = { 23, 31, 1, 21, 36, 72};
        System.out.println("Before sorting : " + Arrays.toString(input));
        quickSort(input); // sort the integer array using quick sort algorithm
        System.out.println("After sorting : " + Arrays.toString(input));

        // input with duplicates
        int[] withDuplicates = { 11, 14, 16, 12, 11, 15};
        System.out.println("Before sorting : "
                              + Arrays.toString(withDuplicates));
        quickSort(withDuplicates); // sort the array using quick sort algorithm
        System.out.println("After sorting : "
                              + Arrays.toString(withDuplicates));
    }

    /**
     * public method exposed to client, sorts given array using QuickSort
     * Algorithm in Java
     * @param array
     */
    public static void quickSort(int[] array) {
        recursiveQuickSort(array, 0, array.length - 1);
    }

    /**
     * Recursive quicksort logic
     *
     * @param array input array
     * @param startIdx start index of the array
     * @param endIdx end index of the array
     */
    public static void recursiveQuickSort(int[] array, int startIdx,
                                                      int endIdx) {

        int idx = partition(array, startIdx, endIdx);

        // Recursively call quicksort with left part of the partitioned array
        if (startIdx < idx - 1) {
            recursiveQuickSort(array, startIdx, idx - 1);
        }

        // Recursively call quick sort with right part of the partitioned array
        if (endIdx > idx) {
            recursiveQuickSort(array, idx, endIdx);
        }
    }

    /**
     * Divides array from pivot, left side contains elements less than
     * Pivot while right side contains elements greater than pivot.
     *
     * @param array array to partitioned
     * @param left lower bound of the array
     * @param right upper bound of the array
     * @return the partition index
     */
    public static int partition(int[] array, int left, int right) {
        int pivot = array[left]; // taking first element as pivot

        while (left <= right) {
            //searching number which is greater than pivot, bottom up
            while (array[left] < pivot) {
                left++;
            }
            //searching number which is less than pivot, top down
            while (array[right] > pivot) {
                right--;
            }

            // swap the values
            if (left <= right) {
                int tmp = array[left];
                array[left] = array[right];
                array[right] = tmp;

                //increment left index and decrement right index
                left++;
                right--;
            }
        }
        return left;
    }
}

Output:
Before sorting : [23, 31, 1, 21, 36, 72]
```

```
After sorting : [1, 21, 23, 31, 36, 72]
Before sorting : [11, 14, 16, 12, 11, 15]
After sorting : [11, 11, 12, 14, 15, 16]
```

5. Things to know about QuickSort Algorithm in Java

As I said, QuickSort is one of the most popular sorting algorithms between programmers, maybe just next to Bubble sort, which is ironically the worst algorithm to sort a large list of numbers. But one thing is common between QuickSort and Bubble Sort, do you know what? In the worst case, both have the complexity of `O(n^2)`.

5.1 QuickSort is a divide and conquers algorithm, which means it sort a large array of numbers by dividing them into a smaller array and then individually sorting them (conquer).

5.2 Average case complexity of Quicksort is `O(n log(n))` and the worst-case complexity of Quicksort is `O(n²)`.

5.3 Quicksort is a comparison sort and, inefficient implementations, it's not a stable sort, which means equal numbers may not retain their original position after sorting.

5.4 Quicksort algorithm can be implemented in-place, which means no additional space will be required. This makes it suitable to sort a large array of numbers.

5.5 The `Arrays.sort()` method in Java uses quicksort to sort an array of primitives like an array of integers or float and uses `Mergesort` to sot objects like an array of String.

That's all about **how to implement a QuickSort algorithm in Java**. QuickSort is one of the fast and efficient sorting algorithm, perfect for sorting large arrays, but some programmer finds it extremely hard to understand. One reason for this could be that because quicksort is an **in-place algorithm** due to which programmers find it a bit confusing, but it's very efficient. Otherwise, if you choose simplicity you can always implement it in other ways.

Other **Data Structure and Algorithms** You may like

- My favorite free courses to learn data Structure in-depth (FreeCodeCamp)
- 50+ Data Structure and Algorithms Problems from Interviews (list)
- 5 Books to Learn Data Structure and Algorithms in-depth (books)
- How to reverse an array in Java? (solution)
- 75+ Coding Interview Questions for Programmers (questions)
- How to remove duplicate elements from the array in Java? (solution)
- How to implement a recursive preorder algorithm in Java? (solution)
- How to implement a binary search tree in Java? (solution)
- Post-order binary tree traversal without recursion (solution)
- How to print leaf nodes of a binary tree without recursion? (solution)
- Recursive Post Order traversal Algorithm (solution)
- Iterative PreOrder traversal in a binary tree (solution)
- How to count the number of leaf nodes in a given binary tree in Java? (solution)
- Recursive InOrder traversal Algorithm (solution)
- 10 Free Data Structure and Algorithms Tutorials (free courses)
- 100+ Data Structure Coding Problems from Interviews (questions)

Thanks for reading this article so far. If you like this Java Array tutorial then please share it with your friends and colleagues. If you have any questions or feedback then please drop a