<https://www.geeksforgeeks.org/quick-sort/>

# QuickSort

* **Difficulty Level :** [Medium](https://www.geeksforgeeks.org/medium/)
* **Last Updated :** 27 Sep, 2022

 Read

 Discuss

Like [Merge Sort](https://www.geeksforgeeks.org/merge-sort/), **QuickSort** is a [Divide and Conquer algorithm](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/). It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

* Always pick the first element as a pivot.
* Always pick the last element as a pivot (implemented below)
* Pick a random element as a pivot.
* Pick median as the pivot.

The key process in **quickSort** is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.



**Partition Algorithm:**

There can be many ways to do partition, following pseudo-code adopts the method given in the CLRS book. The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.

**Pseudo Code for recursive QuickSort function:**

/\* low  –> Starting index,  high  –> Ending index \*/

quickSort(arr[], low, high) {

    if (low < high) {

        /\* pi is partitioning index, arr[pi] is now at right place \*/

        pi = partition(arr, low, high);

        quickSort(arr, low, pi – 1);  // Before pi

        quickSort(arr, pi + 1, high); // After pi

    }

}

**Pseudo code for partition()**

/\* This function takes last element as pivot, places the pivot element at its correct position in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater elements to right of pivot \*/

partition (arr[], low, high)  
{  
    // pivot (Element to be placed at right position)  
    pivot = arr[high];

    i = (low – 1)  // Index of smaller element and indicates the   
    // right position of pivot found so far

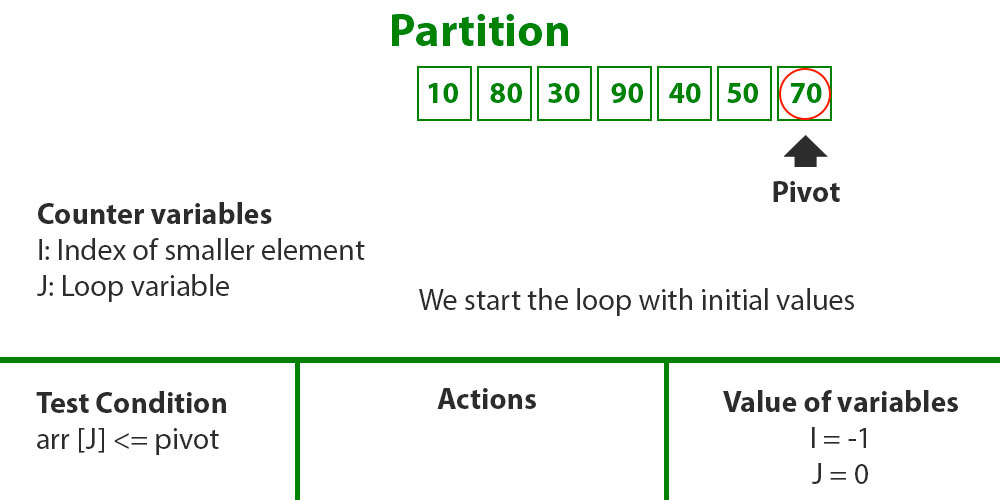
    for (j = low; j <= high- 1; j++){

        // If current element is smaller than the pivot  
        if (arr[j] < pivot){  
            i++;    // increment index of smaller element  
            swap arr[i] and arr[j]  
        }  
    }  
    swap arr[i + 1] and arr[high])  
    return (i + 1)  
}

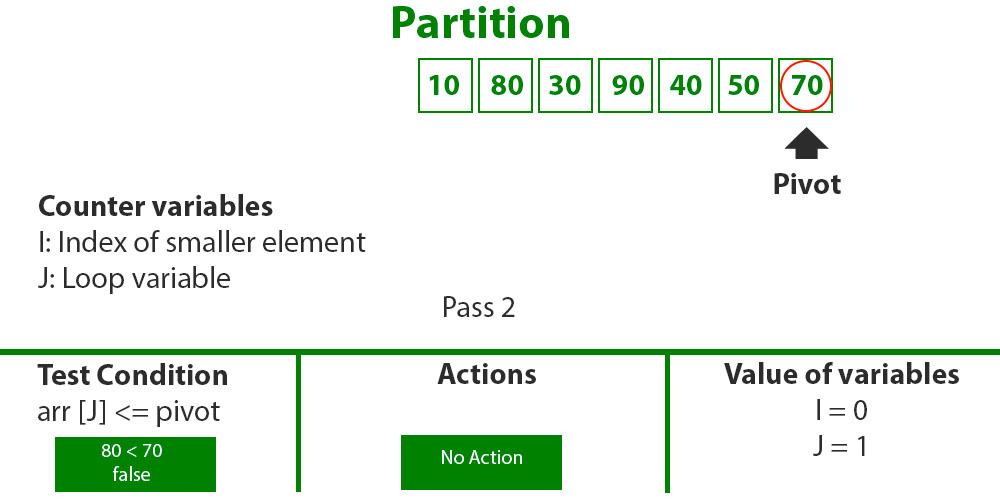
**Illustration of partition() :**

Consider: arr[] = {10, 80, 30, 90, 40, 50, 70}

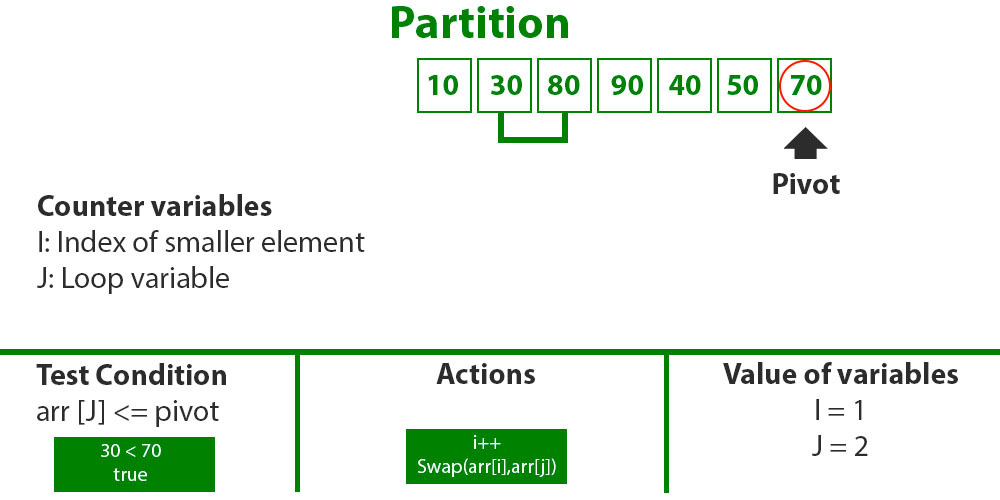
* Indexes:  0   1   2   3   4   5   6
* low = 0, high =  6, pivot = arr[h] = 70
* Initialize index of smaller element, **i = -1**



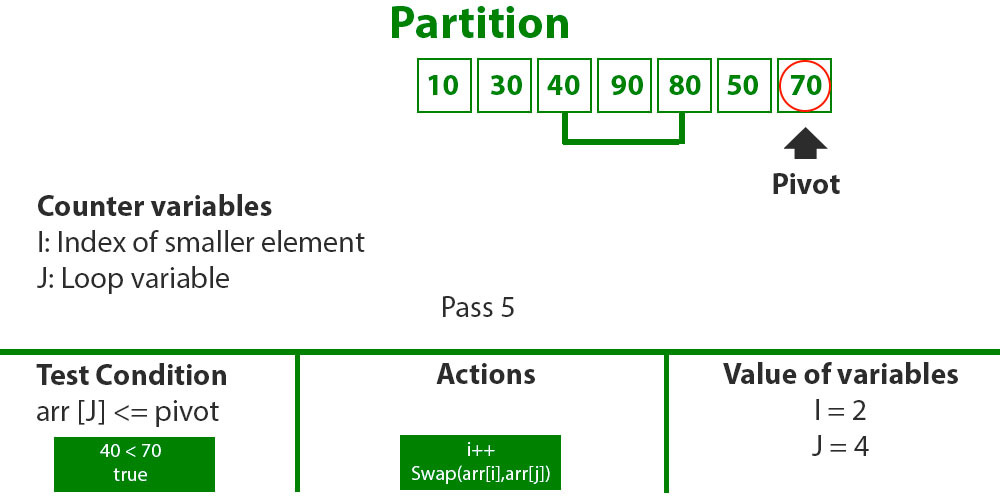
* Traverse elements from j = low to high-1
  + **j = 0**: Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
  + **i = 0**
* arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j are same
* **j = 1**: Since arr[j] > pivot, do nothing



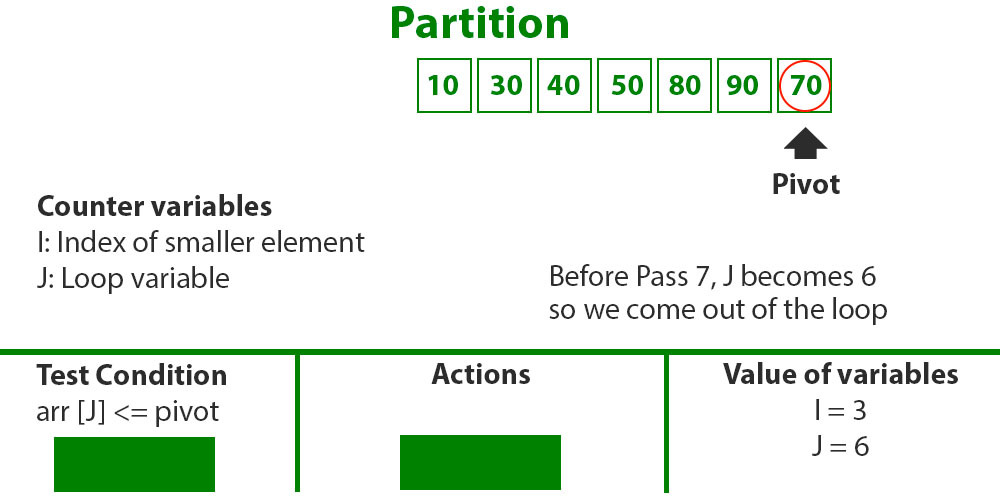
* **j = 2** : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
* **i = 1**
* arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30



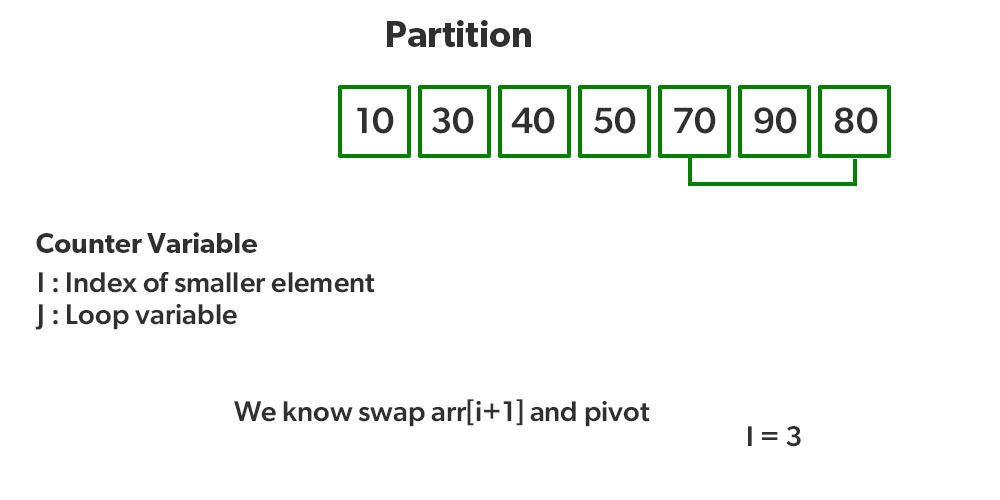
* **j = 3** : Since arr[j] > pivot, do nothing // No change in i and arr[]
* **j = 4** : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
* **i = 2**
* arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped



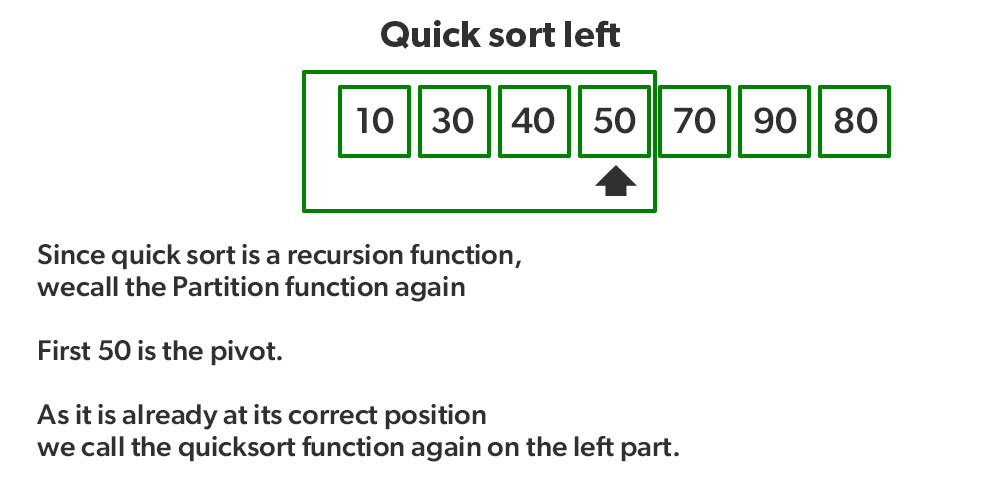
* **j = 5** : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
* **i = 3**
* arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped



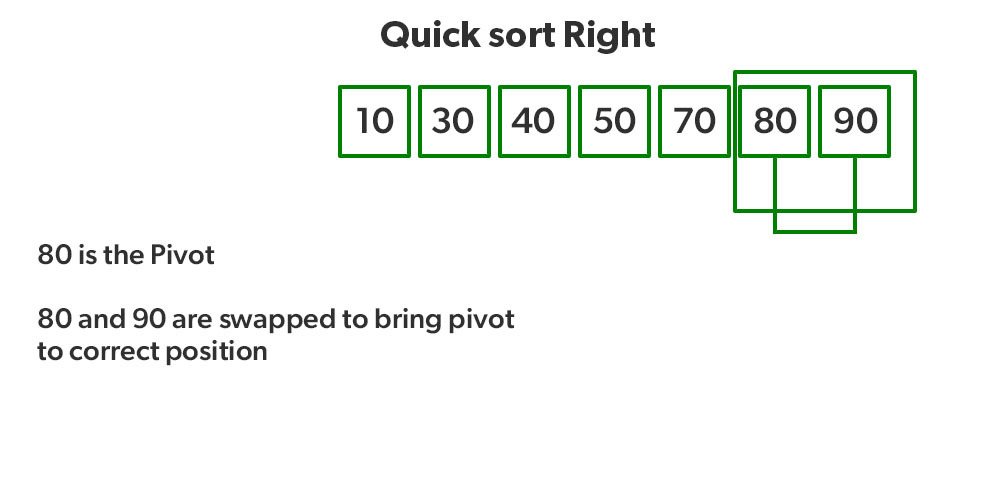
* We come out of loop because j is now equal to high-1.
* **Finally we place pivot at correct position by swapping arr[i+1] and arr[high] (or pivot)**
* arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped



* Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.
* Since quick sort is a recursive function, we call the partition function again at left and right partitions



* Again call function at right part and swap 80 and 90



Recommended Problem

Quick Sort

[Divide and Conquer](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Divide%20and%20Conquer&sortBy=submissions)

[Sorting](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Sorting&sortBy=submissions)

+1 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+11 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/quick-sort/1" \o "Permalink to Quick Sort)

Submission count: 1.1L

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Implementation:**   
Following are the implementations of QuickSort:

|  |
| --- |
| // Java implementation of QuickSort  import java.io.\*;    class GFG {        // A utility function to swap two elements      static void swap(int[] arr, int i, int j)      {          int temp = arr[i];          arr[i] = arr[j];          arr[j] = temp;      }        /\* This function takes last element as pivot, places         the pivot element at its correct position in sorted         array, and places all smaller (smaller than pivot)         to left of pivot and all greater elements to right         of pivot \*/      static int partition(int[] arr, int low, int high)      {            // pivot          int pivot = arr[high];            // Index of smaller element and          // indicates the right position          // of pivot found so far          int i = (low - 1);            for (int j = low; j <= high - 1; j++) {                // If current element is smaller              // than the pivot              if (arr[j] < pivot) {                    // Increment index of                  // smaller element                  i++;                  swap(arr, i, j);              }          }          swap(arr, i + 1, high);          return (i + 1);      }        /\* The main function that implements QuickSort                arr[] --> Array to be sorted,                low --> Starting index,                high --> Ending index       \*/      static void quickSort(int[] arr, int low, int high)      {          if (low < high) {                // pi is partitioning index, arr[p]              // is now at right place              int pi = partition(arr, low, high);                // Separately sort elements before              // partition and after partition              quickSort(arr, low, pi - 1);              quickSort(arr, pi + 1, high);          }      }        // Function to print an array      static void printArray(int[] arr, int size)      {          for (int i = 0; i < size; i++)              System.out.print(arr[i] + " ");            System.out.println();      }        // Driver Code      public static void main(String[] args)      {          int[] arr = { 10, 7, 8, 9, 1, 5 };          int n = arr.length;            quickSort(arr, 0, n - 1);          System.out.println("Sorted array: ");          printArray(arr, n);      }  }    // This code is contributed by Ayush Choudhary |

**Output**

Sorted array:

1 5 7 8 9 10