

Home	Data Structure	C	C++	C#	Java	SQL	HTML	CSS	JavaScript	Ajax
------	----------------	---	-----	----	------	-----	------	-----	------------	------

# Quick Sort Algorithm

In this article, we will discuss the Quicksort Algorithm. The working procedure of Quicksort is also simple. This article will be very helpful and interesting to students as they might face quicksort as a question in their examinations. So, it is important to discuss the topic.

Sorting is a way of arranging items in a systematic manner. Quicksort is the widely used sorting algorithm that makes  **$n \log n$**  comparisons in average case for sorting an array of  $n$  elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

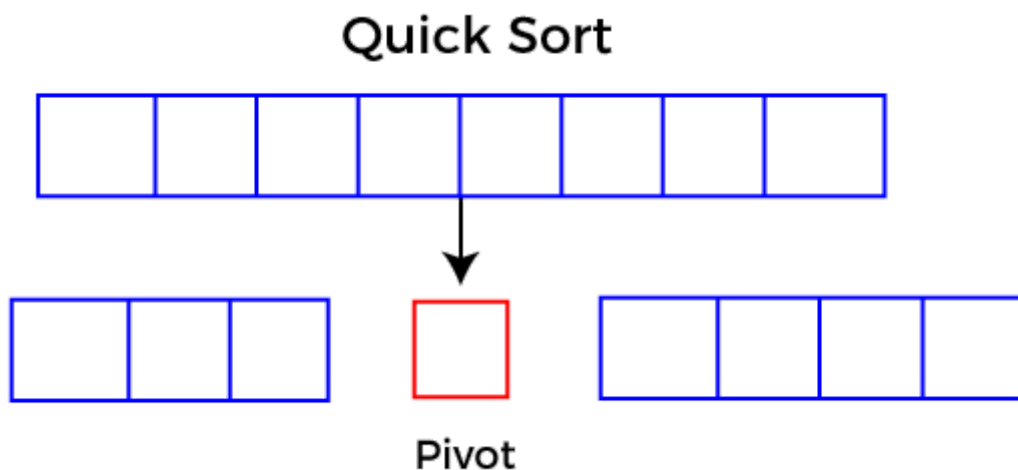
**Divide:** In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

**Conquer:** Recursively, sort two subarrays with Quicksort.

**Combine:** Combine the already sorted array.

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.



## Choosing the pivot

Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows -

- Pivot can be random, i.e. select the random pivot from the given array.
- Pivot can either be the rightmost element or the leftmost element of the given array.
- Select median as the pivot element.

## Algorithm

### Algorithm:

```
QUICKSORT (array A, start, end)
{
  1 if (start < end)
  2 {
  3   p = partition(A, start, end)
  4   QUICKSORT (A, start, p - 1)
  5   QUICKSORT (A, p + 1, end)
  6 }
}
```

### Partition Algorithm:

The partition algorithm rearranges the sub-arrays in a place.

```
PARTITION (array A, start, end)
{
  1 pivot ? A[end]
  2 i ? start-1
  3 for j ? start to end -1 {
  4   do if (A[j] < pivot) {
  5     then i ? i + 1
  6     swap A[i] with A[j]
  7   }}
  8 swap A[i+1] with A[end]
  9 return i+1
}
```

## Working of Quick Sort Algorithm

Now, let's see the working of the Quicksort Algorithm.

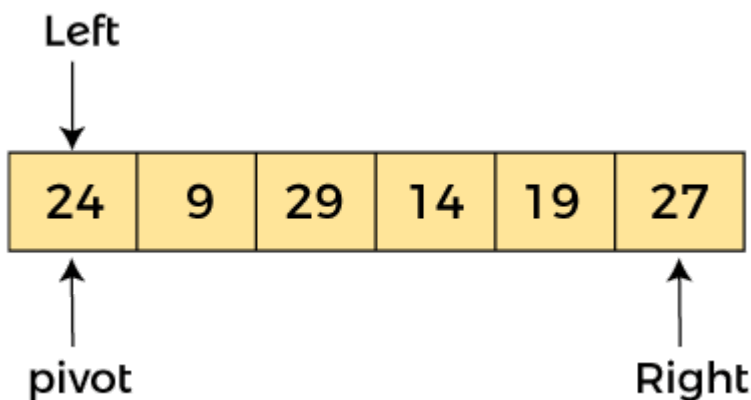
To understand the working of quick sort, let's take an unsorted array. It will make the concept more clear and understandable.

Let the elements of array are -

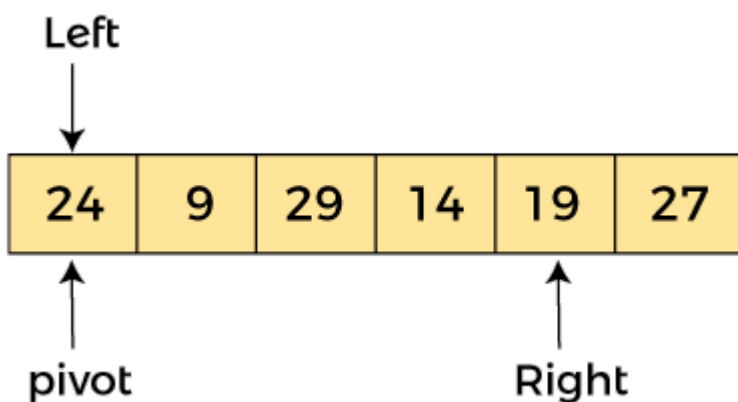
24	9	29	14	19	27
----	---	----	----	----	----

In the given array, we consider the leftmost element as pivot. So, in this case,  $a[\text{left}] = 24$ ,  $a[\text{right}] = 27$  and  $a[\text{pivot}] = 24$ .

Since, pivot is at left, so algorithm starts from right and move towards left.

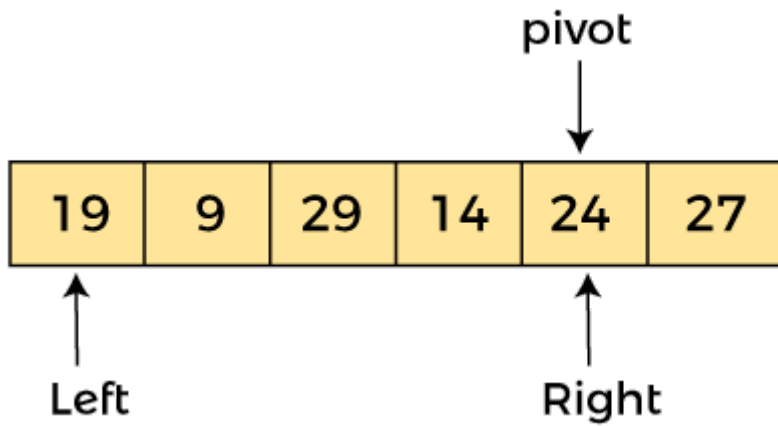


Now,  $a[\text{pivot}] < a[\text{right}]$ , so algorithm moves forward one position towards left, i.e. -



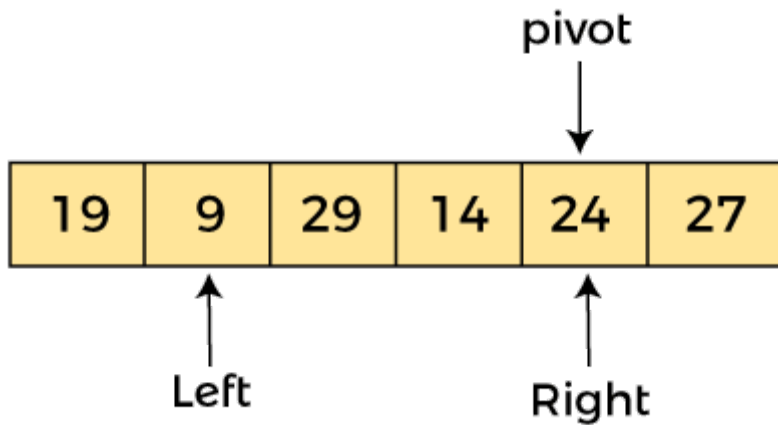
Now,  $a[\text{left}] = 24$ ,  $a[\text{right}] = 19$ , and  $a[\text{pivot}] = 24$ .

Because,  $a[\text{pivot}] > a[\text{right}]$ , so, algorithm will swap  $a[\text{pivot}]$  with  $a[\text{right}]$ , and pivot moves to right, as -

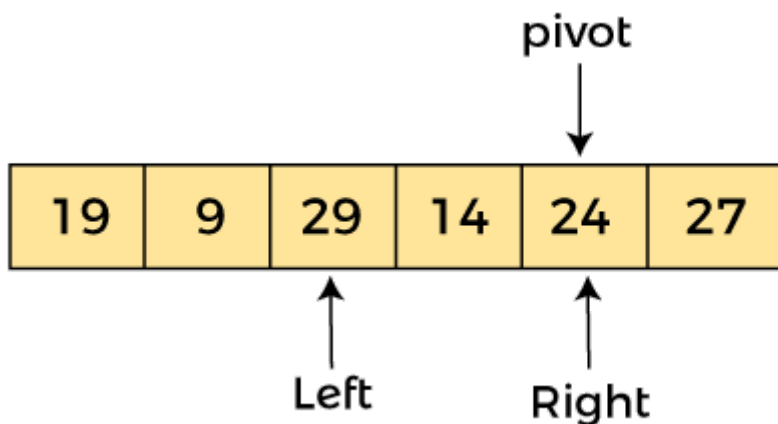


Now,  $a[\text{left}] = 19$ ,  $a[\text{right}] = 24$ , and  $a[\text{pivot}] = 24$ . Since, pivot is at right, so algorithm starts from left and moves to right.

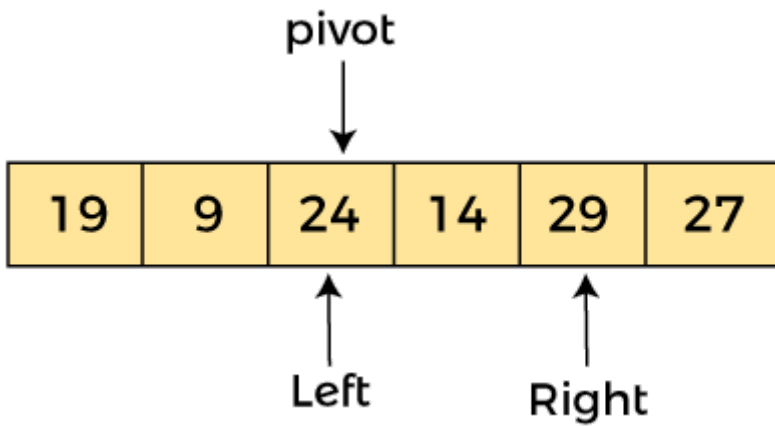
As  $a[\text{pivot}] > a[\text{left}]$ , so algorithm moves one position to right as -



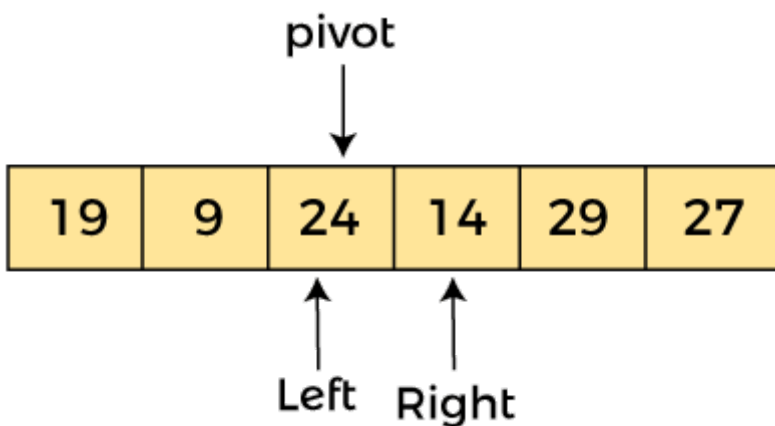
Now,  $a[\text{left}] = 9$ ,  $a[\text{right}] = 24$ , and  $a[\text{pivot}] = 24$ . As  $a[\text{pivot}] > a[\text{left}]$ , so algorithm moves one position to right as -



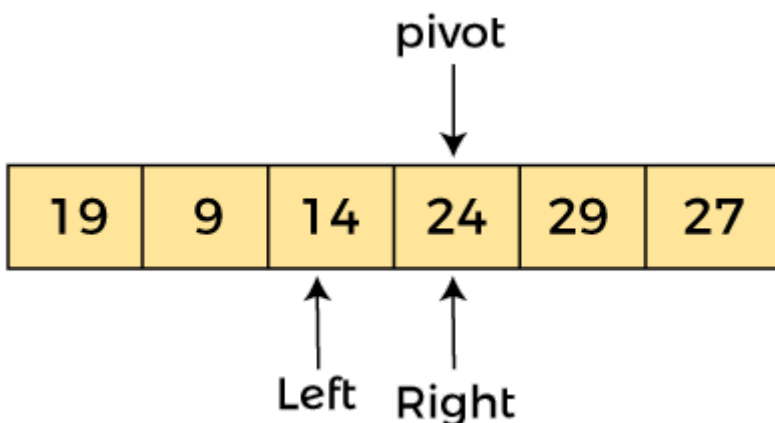
Now,  $a[\text{left}] = 29$ ,  $a[\text{right}] = 24$ , and  $a[\text{pivot}] = 24$ . As  $a[\text{pivot}] < a[\text{left}]$ , so, swap  $a[\text{pivot}]$  and  $a[\text{left}]$ , now pivot is at left, i.e. -



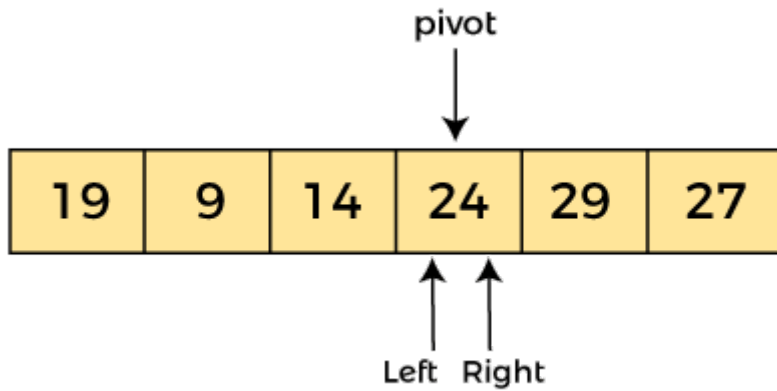
Since, pivot is at left, so algorithm starts from right, and move to left. Now,  $a[\text{left}] = 24$ ,  $a[\text{right}] = 29$ , and  $a[\text{pivot}] = 24$ . As  $a[\text{pivot}] < a[\text{right}]$ , so algorithm moves one position to left, as -



Now,  $a[\text{pivot}] = 24$ ,  $a[\text{left}] = 24$ , and  $a[\text{right}] = 14$ . As  $a[\text{pivot}] > a[\text{right}]$ , so, swap  $a[\text{pivot}]$  and  $a[\text{right}]$ , now pivot is at right, i.e. -



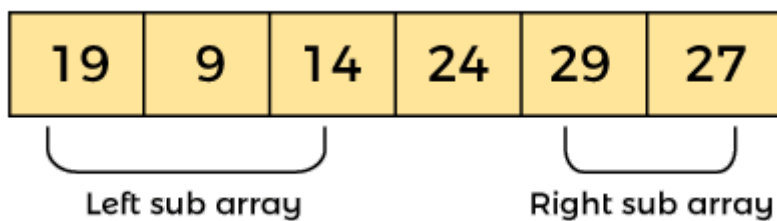
Now,  $a[\text{pivot}] = 24$ ,  $a[\text{left}] = 14$ , and  $a[\text{right}] = 24$ . Pivot is at right, so the algorithm starts from left and move to right.



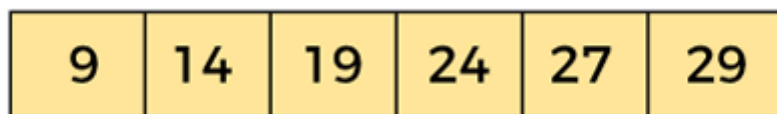
Now,  $a[\text{pivot}] = 24$ ,  $a[\text{left}] = 24$ , and  $a[\text{right}] = 24$ . So, pivot, left and right are pointing the same element. It represents the termination of procedure.

Element 24, which is the pivot element is placed at its exact position.

Elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.



Now, in a similar manner, quick sort algorithm is separately applied to the left and right sub-arrays. After sorting gets done, the array will be -



## Quicksort complexity

Now, let's see the time complexity of quicksort in best case, average case, and in worst case. We will also see the space complexity of quicksort.

### 1. Time Complexity

Case	Time Complexity
Best Case	$O(n \cdot \log n)$

<b>Average Case</b>	$O(n \cdot \log n)$
<b>Worst Case</b>	$O(n^2)$

- **Best Case Complexity** - In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksort is  **$O(n \cdot \log n)$** .
- **Average Case Complexity** - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of quicksort is  **$O(n \cdot \log n)$** .
- **Worst Case Complexity** - In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is  **$O(n^2)$** .

Though the worst-case complexity of quicksort is more than other sorting algorithms such as **Merge sort** and **Heap sort**, still it is faster in practice. Worst case in quick sort rarely occurs because by changing the choice of pivot, it can be implemented in different ways. Worst case in quicksort can be avoided by choosing the right pivot element.

## 2. Space Complexity

<b>Space Complexity</b>	$O(n \cdot \log n)$
<b>Stable</b>	NO

- The space complexity of quicksort is  $O(n \cdot \log n)$ .

## Implementation of quicksort

Now, let's see the programs of quicksort in different programming languages.

**Program:** Write a program to implement quicksort in C language.

```
#include <stdio.h>

/* function that consider last element as pivot,
place the pivot at its exact position, and place
smaller elements to left of pivot and greater
elements to right of pivot. */
```



```
int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {
        // If current element is smaller than the pivot
        if (a[j] < pivot)
        {
            i++; // increment index of smaller element
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

/* function to implement quick sort */
void quick(int a[], int start, int end) /* a[] = array to be sorted, start = Starting index, end = End index */
{
    if (start < end)
    {
        int p = partition(a, start, end); //p is the partitioning index
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

/* function to print an array */
void printArr(int a[], int n)
{

```

```
int i;
for (i = 0; i < n; i++)
    printf("%d ", a[i]);
}

int main()
{
    int a[] = { 24, 9, 29, 14, 19, 27 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    quick(a, 0, n - 1);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

### Output:

```
Before sorting array elements are -
24 9 29 14 19 27
After sorting array elements are -
9 14 19 24 27 29
```

**Program:** Write a program to implement quick sort in C++ language.

```
#include <iostream>

using namespace std;

/* function that consider last element as pivot,
place the pivot at its exact position, and place
smaller elements to left of pivot and greater
elements to right of pivot. */
int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element
    int i = (start - 1);
```

```
for (int j = start; j <= end - 1; j++)
{
    // If current element is smaller than the pivot
    if (a[j] < pivot)
    {
        i++; // increment index of smaller element
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}
int t = a[i+1];
a[i+1] = a[end];
a[end] = t;
return (i + 1);
}

/* function to implement quick sort */
void quick(int a[], int start, int end) /* a[] = array to be sorted, start = Starting index, end = End index */
{
    if (start < end)
    {
        int p = partition(a, start, end); //p is the partitioning index
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

/* function to print an array */
void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout<<a[i]<< " ";
}
```

```
int main()
{
    int a[] = { 23, 8, 28, 13, 18, 26 };
    int n = sizeof(a) / sizeof(a[0]);
    cout<<"Before sorting array elements are - \n";
    printArr(a, n);
    quick(a, 0, n - 1);
    cout<<"\nAfter sorting array elements are - \n";
    printArr(a, n);

    return 0;
}
```

### Output:

```
Before sorting array elements are -
23 8 28 13 18 26
After sorting array elements are -
8 13 18 23 26 28
```

**Program:** Write a program to implement quicksort in python.

```
#function that consider last element as pivot,
#place the pivot at its exact position, and place
#smaller elements to left of pivot and greater
#elements to right of pivot.

def partition (a, start, end):
    i = (start - 1)
    pivot = a[end] # pivot element

    for j in range(start, end):
        # If current element is smaller than or equal to the pivot
        if (a[j] <= pivot):
            i = i + 1
            a[i], a[j] = a[j], a[i]

    a[i+1], a[end] = a[end], a[i+1]
```

```
return (i + 1)
```

```
# function to implement quick sort
```

```
def quick(a, start, end): # a[] = array to be sorted, start = Starting index, end = Ending index
```

```
    if (start < end):
```

```
        p = partition(a, start, end) # p is partitioning index
```

```
        quick(a, start, p - 1)
```

```
        quick(a, p + 1, end)
```

```
def printArr(a): # function to print the array
```

```
    for i in range(len(a)):
```

```
        print (a[i], end = " ")
```

```
a = [68, 13, 1, 49, 58]
```

```
print("Before sorting array elements are - ")
```

```
printArr(a)
```

```
quick(a, 0, len(a)-1)
```

```
print("\nAfter sorting array elements are - ")
```

```
printArr(a)
```

### Output:

```
Before sorting array elements are -  
68 13 1 49 58  
After sorting array elements are -  
1 13 49 58 68
```

**Program:** Write a program to implement quicksort in Java.

```
public class Quick
```

```
{
```

```
    /* function that consider last element as pivot,
```

```
    place the pivot at its exact position, and place
```

```
    smaller elements to left of pivot and greater
```

```
    elements to right of pivot. */
```

```
int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element
    int i = (start - 1);

    for (int j = start; j <= end - 1; j++)
    {
        // If current element is smaller than the pivot
        if (a[j] < pivot)
        {
            i++; // increment index of smaller element
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }
    int t = a[i+1];
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}

/* function to implement quick sort */
void quick(int a[], int start, int end) /* a[] = array to be sorted, start = Starting index, end = End index */
{
    if (start < end)
    {
        int p = partition(a, start, end); //p is partitioning index
        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

/* function to print an array */
void printArr(int a[], int n)
{

```

```
int i;
for (i = 0; i < n; i++)
    System.out.print(a[i] + " ");
}

public static void main(String[] args) {
    int a[] = { 13, 18, 27, 2, 19, 25 };
    int n = a.length;
    System.out.println("\nBefore sorting array elements are - ");
    Quick q1 = new Quick();
    q1.printArr(a, n);
    q1.quick(a, 0, n - 1);
    System.out.println("\nAfter sorting array elements are - ");
    q1.printArr(a, n);
    System.out.println();
}
}
```

## Output

After the execution of above code, the output will be -

```
D:\JTP>javac Quick.java
D:\JTP>java Quick
Before sorting array elements are -
13 18 27 2 19 25
After sorting array elements are -
2 13 18 19 25 27
```

**Program:** Write a program to implement quick sort in php.

```
<?php
```

```
/* function that consider last element as pivot,
place the pivot at its exact position, and place
smaller elements to left of pivot and greater
elements to right of pivot. */
function partition (&$a, $start, $end)
{
    $pivot = $a[$end]; // pivot element
    $i = ($start - 1);
```

```
for ($j = $start; $j <= $end - 1; $j++)
{
    // If current element is smaller than the pivot
    if ($a[$j] < $pivot)
    {
        $i++; // increment index of smaller element
        $t = $a[$i];
        $a[$i] = $a[$j];
        $a[$j] = $t;
    }
}
$t = $a[$i+1];
$a[$i+1] = $a[$end];
$a[$end] = $t;
return ($i + 1);
}

/* function to implement quick sort */
function quick(&$a, $start, $end) /* a[] = array to be sorted, start = Starting index, end = End index */
{
    if ($start < $end)
    {
        $p = partition($a, $start, $end); //p is partitioning index
        quick($a, $start, $p - 1);
        quick($a, $p + 1, $end);
    }
}

function printArray($a, $n)
{
    for($i = 0; $i < $n; $i++)
    {
        print_r($a[$i]);
        echo " ";
    }
}
```

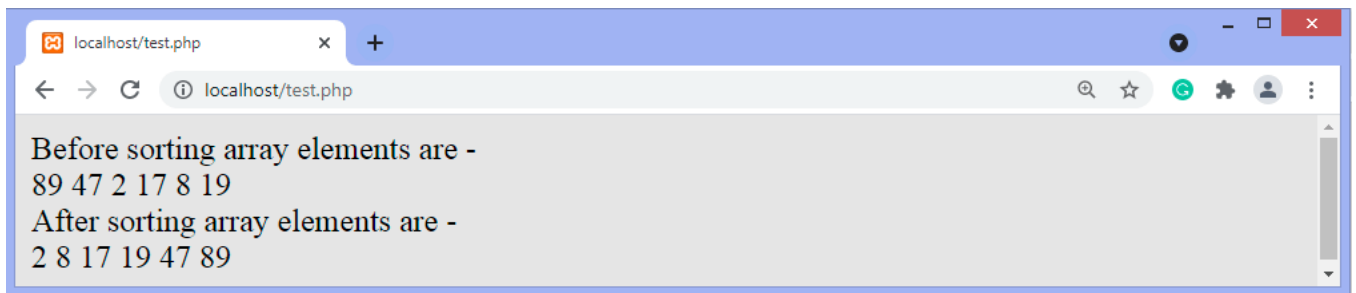


```
}  
$a = array( 89, 47, 2, 17, 8, 19 );  
$n = count($a);  
echo "Before sorting array elements are - <br>";  
printArray($a, $n);  
quick($a, 0, $n - 1);  
echo "<br> After sorting array elements are - <br>";  
printArray($a, $n);
```

?>

## Output

After the execution of above code, the output will be -



So, that's all about the article. Hope the article will be helpful and informative to you.

This article was not only limited to the algorithm. Along with the algorithm, we have also discussed the quick sort complexity, working, and implementation in different programming languages.

← Prev

Next →

 Youtube **For Videos Join Our Youtube Channel: [Join Now](#)**
















## Feedback

- Send your Feedback to [feedback@javatpoint.com](mailto:feedback@javatpoint.com)






## Help Others, Please Share



### Learn Latest Tutorials

 Splunk tutorial Splunk	 SPSS tutorial SPSS	 Swagger tutorial Swagger	 T-SQL tutorial Transact-SQL
 Tumblr tutorial Tumblr	 React tutorial ReactJS	 Regex tutorial Regex	 Reinforcement learning tutorial Reinforcement Learning
 R Programming tutorial R Programming	 RxJS tutorial RxJS	 React Native tutorial React Native	 Python Design Patterns Python Design Patterns
 Python Pillow tutorial Python Pillow	 Python Turtle tutorial Python Turtle	 Keras tutorial Keras	

### Preparation

 Aptitude Aptitude	 Logical Reasoning Reasoning	 Verbal Ability Verbal Ability	 Interview Questions Interview Questions
 Company Interview Questions Questions			

## Company Questions

## Trending Technologies



Artificial  
Intelligence  
Tutorial

Artificial  
Intelligence



AWS Tutorial  
AWS



Selenium  
tutorial  
Selenium



Cloud  
Computing  
tutorial  
Cloud Computing



Hadoop tutorial  
Hadoop



ReactJS  
Tutorial  
ReactJS



Data Science  
Tutorial  
Data Science



Angular 7  
Tutorial  
Angular 7



Blockchain  
Tutorial  
Blockchain



Git Tutorial  
Git



Machine  
Learning Tutorial  
Machine Learning



DevOps  
Tutorial  
DevOps

## B.Tech / MCA



DBMS tutorial  
DBMS



Data Structures  
tutorial  
Data Structures



DAA tutorial  
DAA



Operating  
System tutorial  
Operating System



Computer  
Network tutorial  
Computer Network



Compiler  
Design tutorial  
Compiler Design



Computer  
Organization and  
Architecture  
Computer  
Organization



Discrete  
Mathematics  
Tutorial  
Discrete  
Mathematics



Ethical Hacking  
Tutorial  
Ethical Hacking














Computer  
Graphics Tutorial  
Computer Graphics



Software  
Engineering  
Tutorial  
Software  
Engineering



html tutorial  
Web Technology

 <div>Cyber Security tutorial</div> <div>Cyber Security</div>	 <div>Automata Tutorial</div> <div>Automata</div>	 <div>C Language tutorial</div> <div>C Programming</div>	 <div>C++ tutorial</div> <div>C++</div>
 <div>Java tutorial</div> <div>Java</div>	 <div>.Net Framework tutorial</div> <div>.Net</div>	 <div>Python tutorial</div> <div>Python</div>	 <div>List of Programs</div> <div>Programs</div>
 <div>Control Systems tutorial</div> <div>Control System</div>	 <div>Data Mining Tutorial</div> <div>Data Mining</div>	 <div>Data Warehouse Tutorial</div> <div>Data Warehouse</div>	