

## QuickSort

January 7, 2014

Like [Merge Sort](#), QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- 1) Always pick first element as pivot.
- 2) Always pick last element as pivot (implemented below)
- 3) Pick a random element as pivot.
- 4) Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in 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 code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

### Implementation:

Following is C++ implementation of QuickSort.

```
/* A typical recursive implementation of quick sort */
#include<stdio.h>
```



GeeksQuiz



4,126 people like [GeeksQuiz](#).



Facebook social plugin

## HP Chromebook 11

[google.com/chromebook](http://google.com/chromebook)

Everything you need in one laptop.  
Made with Google. [Learn more.](#)



```
// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places the pivot element
correct position in sorted array, and places all smaller (smaller
to left of pivot and all greater elements to right of pivot */
int partition (int arr[], int l, int h)
{
    int x = arr[h];    // pivot
    int i = (l - 1);  // Index of smaller element

    for (int j = l; j <= h- 1; j++)
    {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= x)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]); // Swap current element with its
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* arr[] --> Array to be sorted, l --> Starting index, h --> Ending index */
void quickSort(int arr[], int l, int h)
{
    if (l < h)
    {
        int p = partition(arr, l, h); /* Partitioning index */
        quickSort(arr, l, p - 1);
        quickSort(arr, p + 1, h);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

## Categories

Articles (28)

C (4)

C++ (2)

Data Structures (13)

DBMS (1)

Operating Systems (1)

Searching and Sorting (7)

Programs (7)

Quizzes (1,392)

Aptitude (1)

Computer Science Quizzes (1,391)

Algorithms (146)

C (203)

C++ (123)

Data Structures (131)

GATE (709)

Java (51)

Operating Systems (28)

```

}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

```

Output:

Sorted array:  
1 5 7 8 9 10

### Analysis of QuickSort

Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

The first two terms are for two recursive calls, the last term is for the partition process.  $k$  is the number of elements which are smaller than pivot.

The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

**Worst Case:** The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

which is equivalent to

$$T(n) = T(n-1) + \Theta(n)$$

The solution of above recurrence is  $\Theta(n^2)$ .

**Best Case:** The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

Custom market  
research at scale.

Get \$75 off

 Google consumer surveys

## Recent Discussions

**Sumit Khatri** this is the sorting technique which can work...

Insertion Sort · 7 hours ago

**Sumit Khatri** no, quick sort requires more swaps than...

Selection Sort · 7 hours ago

**Sumit Khatri** yes, it is the only sorting technique which...

Selection Sort · 7 hours ago

$$T(n) = 2T(n/2) + \Theta(n)$$

The solution of above recurrence is  $\Theta(n \log n)$ . It can be solved using case 2 of [Master Theorem](#).

#### Average Case:

To do average case analysis, we need to **consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy**.

We can get an idea of average case by considering the case when partition puts  $O(n/9)$  elements in one set and  $O(9n/10)$  elements in other set. Following is recurrence for this case.

$$T(n) = T(n/9) + T(9n/10) + \Theta(n)$$

Solution of above recurrence is also  $O(n \log n)$

Although the worst case time complexity of QuickSort is  $O(n^2)$  which is more than many other sorting algorithms like [Merge Sort](#) and [Heap Sort](#), QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

#### References:

<http://en.wikipedia.org/wiki/Quicksort>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**Sudhakar Mishra** I think it should be  $2n + 1$


Data Structures | Binary Trees | Question 12 · 8 hours ago

**Sudhakar Mishra**  $(2n)!/((n+1)!*n!)$

Data Structures | Binary Trees | Question 6 · 1 day ago

**Sudhakar Mishra** Always Y will be more than one because after...


Data Structures | Stack | Question 7 · 1 day ago

AdChoices 

[▶ Java Sort](#)

[▶ C++ Sort List](#)


[▶ Quick Quiz](#)

AdChoices 

[▶ N Sort](#)

[▶ Quick Solution](#)

[▶ Quick Pick](#)

AdChoices 

[▶ Quick Start](#)

[▶ Quick Test](#)

[▶ QuickSort](#)

# ITT Tech - Official Site

itt-tech.edu

Tech-Oriented Degree Programs.  
Education for the Future.



## Related Questions:

- [Bubble Sort](#)
- [Selection Sort](#)
- [Binary Search](#)
- [Heap Sort](#)
- [Merge Sort](#)
- [Insertion Sort](#)



9



0



1

Sort by Best ▼



Join the discussion...

**Gopal Shankar** • 3 months ago

I see that if input is a sorted array, the above program almost swaps every the partition function in answer section of <http://stackoverflow.com/quest...>

^ | ▼ •

**Kartik** ➔ Gopal Shankar • 3 months ago

Yes, it seems to be doing. This is standard algorithm from CLRS b

^ | ▼ •



Subscribe



Add Disqus to your site