

# Quick Sort

Mithila Bongu

December 13, 2013

## Abstract

Quicksort is a sorting algorithm and also known as a Partion-Exchange Sort in which partition is carried out dynamically. Quicksort is a comparision sort and is popular because it is not difficult to implement, works well for a variety of different kinds of input data, and is substantially faster than any other sorting method in typical applications.

In this paper we study about how quicksort is implemented, its algorithm and examples. We even look at the partitioning in detail with comparisons to the other sorts and its complexity.

## 1 Introduction

One of the most widely studied practical problems in computer science is sorting: the use of a computer to put files in order. A person wishing to use a computer to sort is faced with the problem of determining which of the many available algorithms is best suited for his purpose. This task is becoming less difficult than it once was for three reasons. First, sorting is an area in which the mathematical analysis of algorithms has been particu larly successful: we can predict the performance of many sorting methods and compare them intelligently. Second, we have a great deal of experience using sorting algorithm, and we can learn from that experience to separate good algorithms from bad ones. Third, if the tile fits into the memory of the computer, there is one algorithm, called Quicksort, which has been shown to perform well in a variety of situations. Not only is this algorithm simpler than many other sorting algorithms, but empir ical and analytic studies show that Quicksort can be expected to be up to twice as fast as its nearest competitors. The method is simple enough to be learned by programmers who have no previous experi ence with sorting, and those who do know other sorting methods should also find it profitable to learn about quicksort.

The three steps of quicksort are

**Divide :** Rearrange the elements and split the array into two subarrays such that each element on the left subarray is less than or equal to the middle element and each element in the right subarray is greater than the middle element.

**Conquer :** Recursively sort the two subarrays.

**Combine :** None.

## 2 Example

Pick an element, say P(the pivot).

Re-arrange the elements into 3 sub-blocks,

1. Those less than or equal to  $\leq P$  (the left-block)
2. P (the only element in the middle-block)
3. Those greater then or equal to  $\geq P$  (the right-block)

Repeat the process recursively for the left and right sub-blocks.(That is the results of left-block, folowed by P, followed by the results of right-block).

**Step 1:** Choose a pivot



**Step 2:** Lesser values go to the left, equal or greater values go to the right



**Step 3:** Repeat from step 1 with the two sub lists



## 3 History

The quicksort algorithm was developed in 1960 by Sir Charles A. R. Hoare commonly known as Tony Hoare while in the Soviet Union. He won the 1980 ACM Turing Award. He is a British computer scientist and Studied statistics as a graduate student and made major contributions to developing computer languages. One of the quotes of Hoare is

"I conclude that there are two ways of constructing a software design:

One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies."

While studying at Moscow State University, Tony Hoare received an offer of employment from the National Physical Laboratory (NPL) to work on a new project for machine translation from russian to english. However, because dictionaries were stored on magnetic tape, he would have needed to sort the words of a sentence into alphabetical order before translation.

Hoare thought of two methods to solve this problem. The first method would have taken an amount of time proportional to the square of the length of the sentence. The second method would

later manifest as quicksort. At that time, he only knew one language, Mercury Autocode. Unfortunately, he was not able to successfully code quicksort using Mercury Autocode.

In 1961, Hoare attended an Algol 60 class in Brighton. Algol 60 allowed for recursion (the ability of a procedure to call itself). During this course, Hoare programmed an ultra-fast sorting algorithm now known as quicksort. His first paper on quicksort was also published in 1961.

## 4 Advantages and Applications

### Advantages:

One of the fastest algorithms on average. Does not need additional memory (the sorting takes place in the array - this is called in-place processing).

**Disadvantages:** The worst-case complexity is  $O(N^2)$ .

**Applications:** Commercial applications use Quicksort - generally it runs fast, no additional memory, this compensates for the rare occasions when it runs with  $O(N^2)$ .

Never use in applications which require guaranteed response time: Life-critical (medical monitoring, life support in aircraft and space craft) Mission-critical (monitoring and control in industrial and research plants handling dangerous materials, control for aircraft, defense, etc) unless you assume the worst-case response time.

## 5 Algorithm

```

Quicksort( $A, n$ )
Quicksort( $A, 1, n$ )
Quicksort( $A, p, r$ )
if  $p \geq r$  then
  return
end if
 $q = \text{Partition}(A, p, r)$ 
Quicksort( $A, p, q - 1$ )
Quicksort( $A, q + 1, r$ )
Partition( $A, p, r$ )
 $x = A[r]$ 
 $i \leftarrow p - 1$ 
for  $j \leftarrow p$  to  $r - 1$  do
  if  $A[j] \leq x$  then
     $i \leftarrow i + 1$ 
    Exchange  $A[i]$  and  $A[j]$ 
    Exchange  $A[i + 1]$  and  $A[r]$ 
  end if
end for
return  $i + 1$ 

```

## 6 Implementation

### STEP 1: Choosing the pivot

Choosing the pivot is an essential step. Depending on the pivot the algorithm may run very fast, or in quadric time.

Some fixed element: e.g. the first, the last, the one in the middle. This is a bad choice - the pivot may turn to be the smallest or the largest element, then one of the partitions will be empty.

Randomly chosen (by random generator ) - still a bad choice.

The median of the array (if the array has N numbers, the median is the  $[N/2]$  largest number.

This is difficult to compute - increases the complexity. The median-of-three choice: take the first, the last and the middle element. Choose the median of these three elements.

Example:

4,2,6,5,3,9

The first element is 4, the middle is 5 , the last is 9 .

The median of [4,5,9] is 5

### STEP 2: Partitioning

Partitioning is illustrated on the below example.

1. We want larger elements to go to the right and smaller elements to go to the left.

Two pointers L,R are used to scan the elements from left to right and from right to left:

[4,2,6,5,3,9]

L = first element 4 R = last element 9

While L is to the left of R, we move L right, skipping all the elements less than the pivot. If an element is found greater then the pivot, L stops. While R is to the right of L, we move R left, skipping all the elements greater than the pivot. If an element is found less then the pivot, R stops. When both L and R have stopped, the elements are swapped. When L and R have crossed, no swap is performed, scanning stops, and the element pointed to by L is swapped with the pivot. In the example the first swapping will be between 3 and 6.

2. Restore the pivot.

After restoring the pivot we obtain the following partitioning into three groups:

[4,2,3] [ 5 ] [6,9]

**STEP 3:** Recursively quicksort the left and the right parts.



**Step 1**  
Determine pivot

4	2	6	5	3	9
---	---	---	---	---	---

---

**Step 2**  
Start pointers at left and right



4	2	6	5	3	9
---	---	---	---	---	---

---

**Step 3**  
Since  $4 < 5$ , shift left pointer



4	2	6	5	3	9
---	---	---	---	---	---

---

**Step 4**  
Since  $2 < 5$ , shift left pointer  
Since  $6 > 5$ , stop



4	2	6	5	3	9
---	---	---	---	---	---

---

**Step 5**  
Since  $9 > 5$ , shift right pointer  
Since  $3 < 5$ , stop



4	2	6	5	3	9
---	---	---	---	---	---

---

**Step 6**  
Swap values at pointers



4	2	3	5	6	9
---	---	---	---	---	---

---

**Step 7**  
Move pointers one more step



4	2	3	5	6	9
---	---	---	---	---	---

---

**Step 8**  
Since  $5 == 5$ ,  
move pointers one more step  
Stop

4	2	3	5	6	9
---	---	---	---	---	---

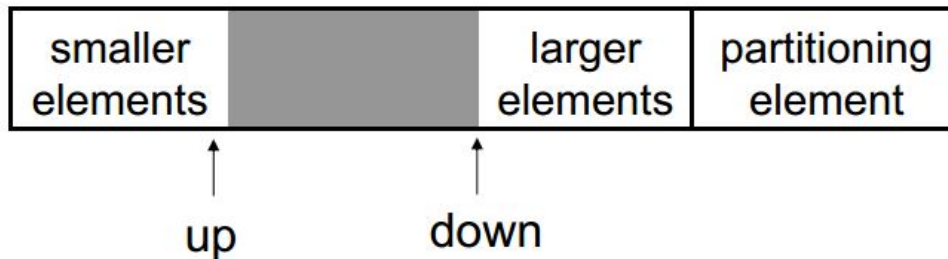
 

## 7 Partitioning

### The Basic Partition:

Choose an arbitrary element. Place the all smaller values to the left. Place the all larger values to the right.

Figure 2: Partitioning



### Improved Partitioning:

How do we avoid picking smallest or largest element for partitioning?

We could take a random element.

We could take a sample.

Median-of-Three Partitioning - Take the sample of three elements. Usually, first, last and middle element. Sort these three elements. It is very unlikely that worst case would occur.

### Different ways to select a good pivot:

First element.

Last element.

Median-of-three elements - Pick three elements, and find the median  $x$  of these elements. Use that median as the pivot.

Random element - Randomly pick a element as a pivot.

## 8 Improvements on Quicksort

Since its development in 1961 by Hoare, the quicksort algorithm has experienced a series of modifications aimed at Improving the  $O(N^2)$  worst case behavior. The improvements can be divided into four categories:

Improvements on the choice of pivot.

Algorithms that use another sorting algorithm for sorting sublists of certain smaller sizes.

Different ways of partitioning lists and sublists.

Adaptive sorting that tries to improve on the  $O(N^2)$  behavior of the quicksort algorithm when used for sorting lists that are sorted or nearly sorted.

## 9 Complexity of Quicksort

**Worst-case:**  $O(N^2)$

This happens when the pivot is the smallest (or the largest) element. Then one of the partitions is empty, and we repeat recursively the procedure for  $N-1$  elements.

**Best-case-**  $O(N\log N)$  is when the pivot is the median of the array, and then the left and the right part will have same size. There are  $\log N$  partitions, and to obtain each partitions we do  $N$  comparisons (and not more than  $N/2$  swaps). Hence the complexity is  $O(N\log N)$ .

**Average-case** -  $O(N\log N)$ .

## 10 Comparison

**Comparison with heapsort:**

Both algorithms have  $O(N\log N)$  complexity.  
Quicksort runs faster, (does not support a heap tree).  
The speed of quick sort is not guaranteed.

**Comparison with mergesort:**

Mergesort guarantees  $O(N\log N)$  time, however it requires additional memory with size  $N$ .  
Quicksort does not require additional memory, however the speed is not guaranteed.  
Usually mergesort is not used for main memory sorting, only for external memory sorting.

Figure 3: Comparisons

<u>Sorting Algorithm</u>	<u>Worst-case time</u>	<u>Average-case time</u>	<u>Space overhead</u>
<b>Bubble Sort</b>	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$
<b>Insertion Sort</b>	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$
<b>Merge Sort</b>	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
<b>Quick Sort</b>	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(1)$

## 11 Conclusion

Quicksort turns out to be the fastest sorting algorithm in practice. It has a time complexity of  $(N \log(N))$  on the average. However, in the (very rare) worst case quicksort is as slow as Bubblesort, namely in  $O(N^2)$ . There are sorting algorithms with a time complexity of  $O(N \log(N))$  even in the worst case, e.g. Heapsort and Mergesort. But on the average, these algorithms are by a constant factor slower than quicksort. It is possible to obtain a worst case complexity of  $O(N \log(N))$  with a variant of quicksort (by choosing the median as comparison element).

## References

- [1] <http://en.wikipedia.org/wiki/Quicksort>
- [2] [http://en.wikipedia.org/wiki/Tony\\_Hoare](http://en.wikipedia.org/wiki/Tony_Hoare)
- [3] Robert Sedgewick "Implementing Quicksort Programs" -1978
- [4] David R.Musser "Introspective Sorting and Selection Algorithms" -1997