

# Quicksort

Quicksort is a general sorting algorithm of type "Divide and Conquer". It is based on splitting the array that is to be sorted into two parts and sorting the two parts independently. The algorithm has the following form:

```
void quicksort (int A[], int l, int r)
{
    if(l >= r) return;
    int i = l;
    int j = r+1;
    int v = A[l];
    for (;;)
    {
        while(A[++i] < v && i < r);
        while(A[--j] > v);
        if(i >= j) {
            swap(A, l, j);
            break;
        }
        swap(A, i, j);
    }
    quicksort(A, l, j-1);
    quicksort(A, j+1, r);
}
```

The parameter `l` and `r` bound the part of the array inside the original array that is to be sorted; the call `quicksort(A, 0, A.size()-1)` sorts the whole array.

The crucial part of the method is the code inside the infinite loop `for(;;)`, which rearranges the array in such a way that the following conditions are satisfied:

- i. For an arbitrary `j` the element `A[j]` is in its final position in the array.
- ii. All elements `A[l]`, `...`, `A[j-1]` are smaller than or equal to `A[j]`.
- iii. All elements `A[j+1]`, `...`, `A[r]` are greater than or equal `A[j]`.

In the first step the splitting element `v = A[l]` is chosen, which is to be placed in its final position. Now the search begins from the left until an element is found that is greater than `v`. Then the search begins from the right until an element is found that is smaller than `v`. If the pointers `i` and `j` have not yet met, the elements `A[i]` and `A[j]` are exchanged, otherwise another exchange operation ensures condition i. and the infinite loop is left by a `break`.

In the following the function is called recursively for the left part and afterwards for the right part of the array. Finally the array is in non-decreasing order.

## Correctness of the Algorithm

*Proof.*

It has to be shown that the following holds in every iteration step:

$$A[1], \dots, A[i] \leq A[j], \dots, A[r].$$

*Induction Base:*

Before the first iteration we have  $i = 1$  and  $j = r+1$ .

*Induction Step:*

The transition from  $i_{old} \rightarrow i_{new}$  and  $j_{old} \rightarrow j_{new}$ .

By Induction Hypothesis we have:

$$A[1], \dots, A[i_{old}] \leq A[j_{old}], \dots, A[r]$$

By construction we have:

$$A[i_{old}+1], \dots, A[i_{new}-1] \leq A[j_{new}+1], \dots, A[j_{old}+1]$$

It follows that

$$A[1], \dots, A[i_{new}-1] \leq A[j_{new}+1], \dots, A[r]$$

It is also true that

$$A[i_{new}] \geq x \text{ und } A[j_{new}] \leq x$$

After exchanging  $A[i_{new}]$  and  $A[j_{new}]$  we have:

$$A[1], \dots, A[i_{new}] \leq A[j_{new}], \dots, A[r]$$

□

## Running Time (worst-case)

We assume that we have the worst case if the array is already in non-decreasing order. In this case in every recursive call only one element is removed from the still to be sorted part of the array (see animation). The resulting running time  $T(n)$  has the complexity.

$$\begin{aligned} T(n) &= \mathcal{O} \left( \sum_{i=2}^n i + n \cdot c \right) \\ &= \mathcal{O} (n^2) \end{aligned}$$

*Proof.*

In general we have:

$$T(n) = \max_{1 \leq q \leq n-1} \{T(q) + T(n-q) + c_1 \cdot n\}$$

It is shown now that  $T(n) \leq (c \cdot n^2)$  for  $c \geq \frac{c_1}{2} \cdot \frac{n_0}{n-1}$  for  $n_0$  big enough holds.

$$\begin{aligned} T(n) &\leq \max_{1 \leq q \leq n-1} \{T(q) + T(n-q) + c_1 \cdot n\} \\ &\stackrel{\text{i.H.}}{\leq} \max_{1 \leq q \leq n-1} \{c \cdot q^2 + c \cdot (n-q)^2 + c_1 \cdot n\} \\ &= c \cdot \max_{1 \leq q \leq n-1} \{q + (n-q)^2 - 2q \cdot (n-q)\} + c_1 \cdot n \\ &= c \cdot n^2 - 2c \cdot \min_{1 \leq q \leq n-1} \{q \cdot (n-q)\} + c_1 \cdot n \\ &= c \cdot n^2 - 2c \cdot 1(n-1) + c_1 \cdot n \end{aligned}$$

For  $n$  big enough it follows that

$$\begin{aligned} T(n) &= c \cdot n - 2 \cdot \frac{c_1}{2} \cdot \frac{n}{n-1} \cdot (n-1) + c_1 \cdot n \\ &= c \cdot n^2 \end{aligned}$$

□

In practice the algorithm is used very often. One reason, among others, is the average running time, which is examined in the following.

**Theorem 1.** *Quicksort needs on average  $2 \log n + \Theta(1)$  many comparisons. By randomized quicksort (randomly exchanging two elements in the still to be sorted part of the array) the worst case is avoided with high probability.*

*Proof.*

$$QSA(n) = n + 2 + \frac{1}{n-1} \cdot \sum_{1 \leq q \leq n-1} (QSA(q) + QSA(n-q))$$

with

$$\sum_{1 \leq q \leq n-1} QSA(q) = \sum_{1 \leq q \leq n-1} QSA(n-q)$$

follows

$$QSA(n) = n + 2 + \frac{2}{n-1} \cdot \sum_{1 \leq q \leq n-1} QSA(q)$$

$$(n-1) \cdot QSA(n) = (n+2) \cdot (n-1) + 2 \cdot \sum_{1 \leq q \leq n-1} QSA(q) \quad | -$$

$$(n-2) \cdot QSA(n) = (n+1) \cdot (n-2) + 2 \cdot \sum_{1 \leq q \leq n-1} QSA(q)$$

$$(n-1) \cdot QSA(n) - (n-2) \cdot QSA(n-1) = \underbrace{(n+2) \cdot (n-1)}_{n^2-n+2} - \underbrace{(n+1) \cdot (n-2)}_{n^2+n-2} + 2 \cdot QSA(n-1)$$

$$(n-1) \cdot QSA(n) = 2 \cdot n + n \cdot QSA(n-1) \quad | : n \cdot (n-1)$$

$$\begin{aligned} \frac{QSA(n)}{n} &= \frac{QSA(n-1)}{n-1} + \frac{2}{n-1} \\ &= \frac{QSA(n-2)}{n-2} + \frac{2}{n-2} + \frac{2}{n-1} \\ &= \frac{QSA(n-2)}{n-3} + \frac{2}{n-3} + \frac{2}{n-2} + \frac{2}{n-1} \\ &\dots \end{aligned}$$

$QSA(k) = c$  for a fixed  $k$  follows

$$\begin{aligned} \frac{QSA(n)}{n} &= \Theta \left( c + 2 \cdot \sum_{i=1}^{n-1} \frac{1}{i} \right) \\ &= 2 \log n + \Theta(1) \\ &= \Theta(\log n) \end{aligned}$$

$$QSA(n) = \Theta(n \log n)$$

□