Fast sorting algorithms

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

November 1, 2023

In this lecture we consider three fast sorting algorithms. The complexity of them is close to the optimal estimate $O(N \log N)$.

Quicksort algorithm

Quicksort is an efficient, general-purpose sorting algorithm. It is still a very popular and commonly used in different applications algorithm.

We will show that its average complexity is $\mathcal{O}(N \log N)$, and Quicksort can be done in-place, requiring only small additional amounts of memory to perform the sorting.

Quicksort is a divide-and-conquer type algorithm.

A partition produces a division into two consecutive non empty sub-sets, in such a way that no element of the first sub-set is greater than any element of the second sub-set.

Quicksort is a divide-and-conquer type algorithm.

A partition produces a division into two consecutive non empty sub-sets, in such a way that no element of the first sub-set is greater than any element of the second sub-set.

After applying this partition, Quicksort then recursively sorts the sub-sets.

Partitioning step.

We partition a set A into two sub-sets.

Partitioning step.

We partition a set A into two sub-sets.

First, a key element a_j is selected. It is called a division point or pivot.

Partitioning step.

We partition a set A into two sub-sets.

First, a key element a_j is selected. It is called a division point or pivot.

Next we reorder elements of A so that all elements with values less than the pivot come before the division point,

while all elements with values greater than the pivot come after it.

Partitioning step.

We partition a set A into two sub-sets.

First, a key element a_j is selected. It is called a division point or pivot.

Next we reorder elements of A so that all elements with values less than the pivot come before the division point,

while all elements with values greater than the pivot come after it.

Elements that are equal to the pivot can go either way.

Sorting of sub-sets

If the sub-set has fewer than two elements, return.

Otherwise, apply Quicksort to this sub-set (recursion).

Sorting of sub-sets

If the sub-set has fewer than two elements, return.

Otherwise, apply Quicksort to this sub-set (recursion).

A popular modification selects a small number M.

If the sub-set has fewer than M elements, sort it by some simple sorting algorithm, e.g. Insert sort.

Determination of the solution

Since no element of the first sub-set is greater than any element of the second sub-set, thus by sorting sub-sets we finish sorting all elements of A.

No computations are done at this stage.

Quicksort algorithm

QuickSort (l, r)
begin
(1) **if** (l < (r - M)) **then**
(2) Partition (l, r, m);
(3) QuickSort (l, m-1);
(4) QuickSort (m+1, r);
else
(5) **if** (l < r) SelectionSort (l, r);
end if
end QuickSort

Partition (l, r, m)
begin

(1) $v = a_l$;
(2) $i = l$; $j = r$;
(3) **while** $(i < j)$ **do**
(4)   **while** $((a_j \geqslant v) \; \&\& \; (i < j))$ $j = j - 1$;
(5)   **if** $(i \neq j)$ **then**
(6) $a_i = a_j$; i++;
    end if
(7)   **while** $((a_i \leqslant v) \; \&\& \; (i < j))$ $i = i + 1$;
(8)   **if** $(i \neq j)$ **then**
(9) $a_j = a_i$; j−−;
    end if
end do
(10) $a_i = v$; $m = i$;

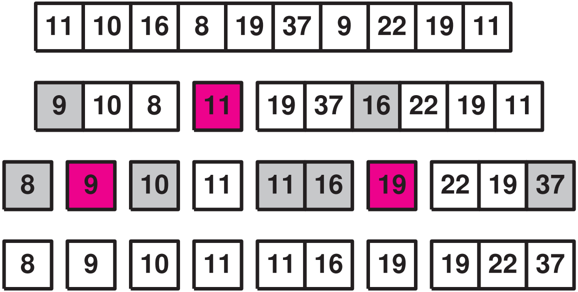
end Partition

Let's sort a list

$$A = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11).$$

Let's sort a list

$$A = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11).$$



The first element of any sub-set is used as a pivot.
Pivots are colored red, grey colored elements are swaped during partition steps.

Complexity of Quicksort algorithm

We are interested to find a number of comparisons L_N required to sort a given set of N elements.

Complexity of Quicksort algorithm

We are interested to find a number of comparisons L_N required to sort a given set of N elements.

During a partition step each element is compared with a pivot.

Thus a total number of comparisons depends only on sizes of produced sub-sets.

Let's consider the worst case, when the smallest element is selected as a pivot.

Then we get the following equation

$$L_B(N) = L_B(N-1) + N - 1\,.$$

If a set contains only one element then it is already sorted:

$$L(1) = 0\,.$$

Let's consider the worst case, when the smallest element is selected as a pivot.

Then we get the following equation

$$L_B(N) = L_B(N-1) + N - 1 \,.$$

If a set contains only one element then it is already sorted:

$$L(1) = 0 \,.$$

By applying this relation $(N-1)$ times, we get

$$L_B(N) = \sum_{i=2}^{N} (i-1) = \sum_{j=1}^{N-1} j = \frac{N^2 - N}{2} \,.$$

Let's consider the worst case, when the smallest element is selected as a pivot.

Then we get the following equation

$$L_B(N) = L_B(N-1) + N - 1\,.$$

If a set contains only one element then it is already sorted:

$$L(1) = 0\,.$$

By applying this relation $(N-1)$ times, we get

$$L_B(N) = \sum_{i=2}^{N}(i-1) = \sum_{j=1}^{N-1} j = \frac{N^2 - N}{2}\,.$$

Thus in the worst case this algorithm is not faster than Insert sort or Select sort algorithms.

The most un-expected conclusion is that such a result follows for already sorted sets (when the first element is selected as a pivot).

Let's consider the best case, when at each partition step we select the pivot element which divides a set into two sub-sets of equal sizes.

Let's consider the best case, when at each partition step we select the pivot element which divides a set into two sub-sets of equal sizes.

Take $N = (2^m - 1)$. Then the number of comparisons satisfy the relation:

$$L_G(2^m - 1) = \begin{cases} 2L_G(2^{m-1} - 1) + 2^m - 2, & \text{when} \quad m > 1, \\ 0, & \text{when} \quad m = 1\,. \end{cases}$$

Applying it $(m-2)$ times we get

$$\begin{aligned} L_G(N) &= 2^m - 2 + 2 \cdot (2^{m-1} - 2) + 2^2 \cdot (2^{m-2} - 2) + \dots \\ &\quad + 2^{m-2} \cdot (2^2 - 2) \\ &= (m-1)2^m + 2^m - 2 \\ &= (N+1)\log(N+1) - 2. \end{aligned}$$

Applying it $(m-2)$ times we get

$$\begin{aligned} L_G(N) &= 2^m - 2 + 2 \cdot (2^{m-1} - 2) + 2^2 \cdot (2^{m-2} - 2) + \dots \\ &+ 2^{m-2} \cdot (2^2 - 2) \\ &= (m-1)2^m + 2^m - 2 \\ &= (N+1)\log(N+1) - 2. \end{aligned}$$

We note that for the Insert sort algorithm the complexity of the best case is even better N.

Applying it $(m-2)$ times we get

$$\begin{aligned} L_G(N) &= 2^m - 2 + 2 \cdot (2^{m-1} - 2) + 2^2 \cdot (2^{m-2} - 2) + \dots \\ &\quad + 2^{m-2} \cdot (2^2 - 2) \\ &= (m-1)2^m + 2^m - 2 \\ &= (N+1)\log(N+1) - 2. \end{aligned}$$

We note that for the Insert sort algorithm the complexity of the best case is even better N.

But only for the best case.

Quicksort algorithm is so popular since in the average case its complexity is also very close to the best case

$$L_V(N) = 1,386 N \log N + \mathcal{O}(N)\,.$$

Quicksort algorithm is so popular since in the average case its complexity is also very close to the best case

$$L_V(N) = 1,386 N \log N + \mathcal{O}(N) \,.$$

Sorting is done in-place.

It was explained above that a complexity of Quicksort algorithm is only $\mathcal{O}(N^2)$, when

a set of elements is almost sorted,

and the first element of a sub-set is selected as a pivot.

It was explained above that a complexity of Quicksort algorithm is only $\mathcal{O}(N^2)$, when

a set of elements is almost sorted,

and the first element of a sub-set is selected as a pivot.

Thus the following two modfications of the base algorithm are recommended:

It was explained above that a complexity of Quicksort algorithm is only $\mathcal{O}(N^2)$, when

a set of elements is almost sorted,

and the first element of a sub-set is selected as a pivot.

Thus the following two modfications of the base algorithm are recommended:

1. At each recursion stage three elements of A are selected in random a_k, a_l and a_m and they are sorted.

Then a mid element is taken as a pivot.

It was explained above that a complexity of Quicksort algorithm is only $\mathcal{O}(N^2)$, when

a set of elements is almost sorted,

and the first element of a sub-set is selected as a pivot.

Thus the following two modfications of the base algorithm are recommended:

1. At each recursion stage three elements of A are selected in random a_k, a_l and a_m and they are sorted.

Then a mid element is taken as a pivot.

2. Before starting the Quicksort algorithm we swap all elements of A in random.

There is a big probability that sorting costs of such perturbed set will be close to the average complexity of Quicksort.