sourcemind

Data Structures and Algorithms

Lionel KITIHOUN

Session 3

Sorting Algorithms

sourcemind

Objective

- Understanding the concept of sorting.
- Naive sorting algorithm (bubble sort).
- Efficient sorting algorithm (qsort).

Note

Part of the content of this session comes from the website of the book Algorithms by Robert Sedgewick and Kevin Wayne, https://algs4.cs.princeton.edu.

sourcemind

Problem

Given an array of objects, put them in increasing (or non-increasing) order.

8	1	3	7	2

sourcemind

Definition

Sorting is the process of rearranging a sequence of objects so as to put them in some logical order.

Applications

Sorting plays an important role in computer science and has many applications.

- Determining the winner of an election by number of votes.
- Ranking teams in a championship by number of points.
- Identify the best-performing salespeople based on their sales figures.
- And so many more…

Bubble sort

sourcemind

Bubble sort

It is one of the simplest sorting algorithms:

- Visit each item in the array.
- If it is greater than the item next to it, swap them.
- Repeat this process until no more swap is required.

Bubble sort example

We will demonstrate how bubble sort works with this example array.

8	1	3	7	2

sourcemind

Bubble sort example

8	1	3	7	2

8 is greater than 1, so we swap them.

1	8	3	7	2

sourcemind

Bubble sort example

1	8	3	7	2

8 is greater than 3, so we swap them.

1	3	8	7	2

Bubble sort example

1	3	8	7	2

8 is greater than 7, another swap.

1	3	7	8	2

Bubble sort example

1	3	7	8	2

8 is greater than 2, we swap them.

1	3	7	2	8

sourcemind

Bubble sort example

1	3	7	2	8

1 is less than 3, no swap required.

Bubble sort example

1	3	7	2	8

3 is less than 7, no swap required.

1	3	7	2	8

sourcemind

Bubble sort example

1	3	7	2	8

7 is greater than 2, we have to swap them.

1	3	2	7	8

Bubble sort example

1	3	2	7	8

1 is less than 3, no swap required.

Bubble sort example

1	3	2	7	8

3 is greater than 2, we need to swap them.

1	2	3	7	8

Bubble sort example

1	2	3	7	8

3 is less than 7, no swap.

Bubble sort example

1	2	3	7	8

7 is less than 8, no swap.

Bubble sort example

1	2	3	7	8

At this point, the array is sorted, so we are done.

Question

Can you tell from this example why it is called bubble sort?

sourcemind

Bubble sort pseudocode

```
function bubblesort(a: int[]):
    swapped = true
    while swapped:
        swapped = false
        for i = 0 to a.length - 1:
            if a[i] > a[i+1]:
                swap(a[i], a[i+1])
                swapped = true
```

sourcemind

Proof of bubble sort correctness

When creating an algorithm, it is not enough to show that it works on a **particular input**. You have to give a rigorous proof that it works on **any input**.

🥲

Proof of bubble sort correctness

So, how do we prove that bubble sort algorithm really sorts the input array?

This time, we are somehow lucky.

Proof of bubble sort correctness

The proof is simple. The algorithm stops when for every element of the array, the next element is greater or equal.

That implies the bubble sort algorithm arranges the elements of the array in ascending order, which means it sorts the array.

Complexity of bubble sort

- Short answer: $O(N^2)$
- Why?

Complexity of bubble sort

- Bubble sort processes by rounds.
- Each round corresponds to a run of the outer while loop.
- In each round, the ith greater element is put at its final place in the sorted array.
- So we have at most N rounds.
- In each rounds, we visit the first N - 1 elements of the array that we need to compare to the next element and swap them if needed (4 operations).

```
function bubblesort(a: int[]):
   swapped = true
   while swapped:
      swapped = false
      for i = 0 to a.length - 1:
         if a[i] > a[i+1]:
            swap(a[i], a[i+1])
            swapped = true
```

sourcemind

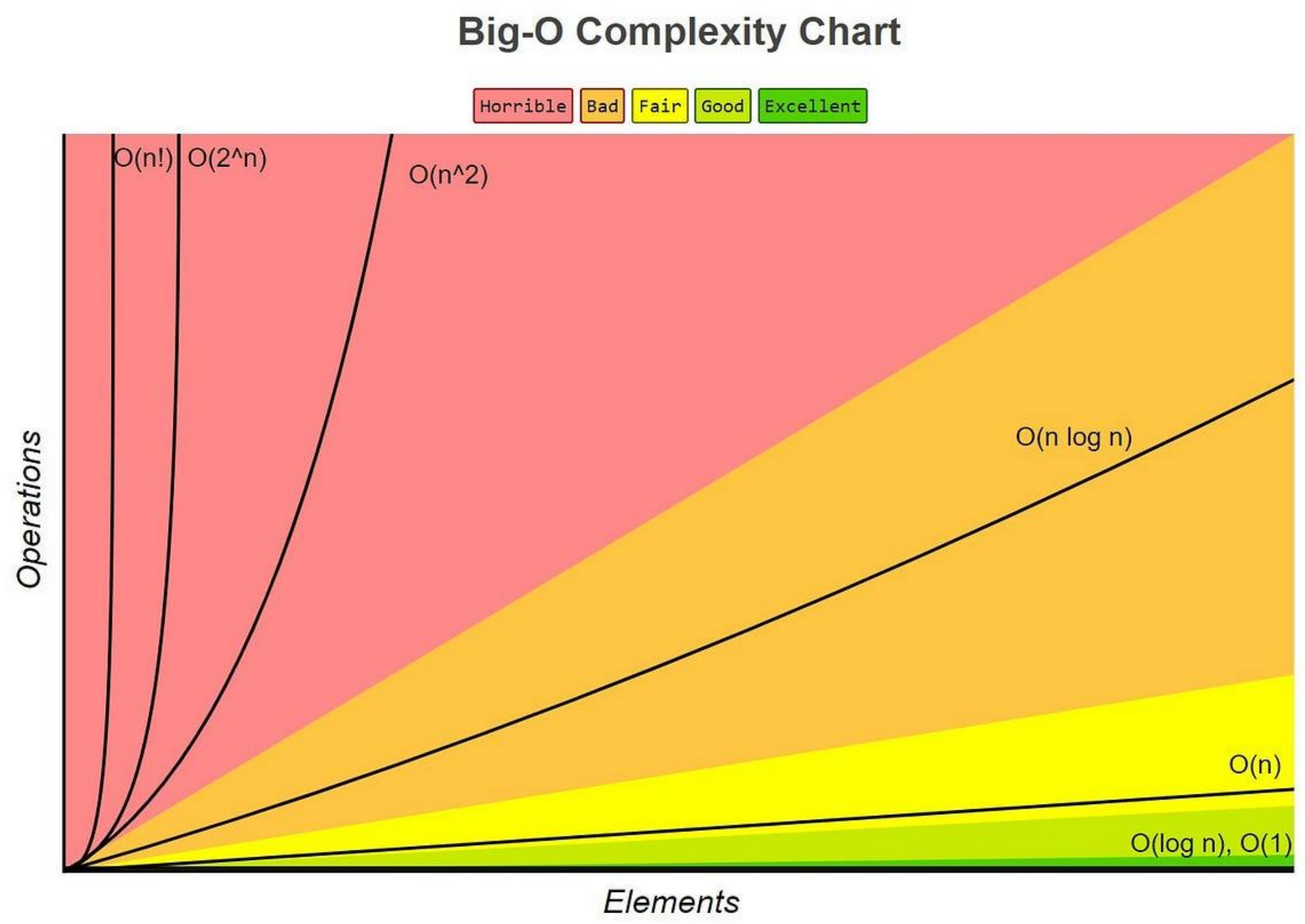
Complexity of bubble sort

- N * 4 (N - 1) operations.
- N (4N - 4).
- $4N^2 - 4N$.
- N^2 dominates N.
- Overall complexity: **$O(N^2)$**.

sourcemind

Performance of bubble sort

- $O(N^2)$ is not good for a sorting algorithm.
- Once the array we want to sort has more than 10K elements, bubble sort is not an option.
- Remember the graph showing function grows?


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

sourcemind

Quicksort

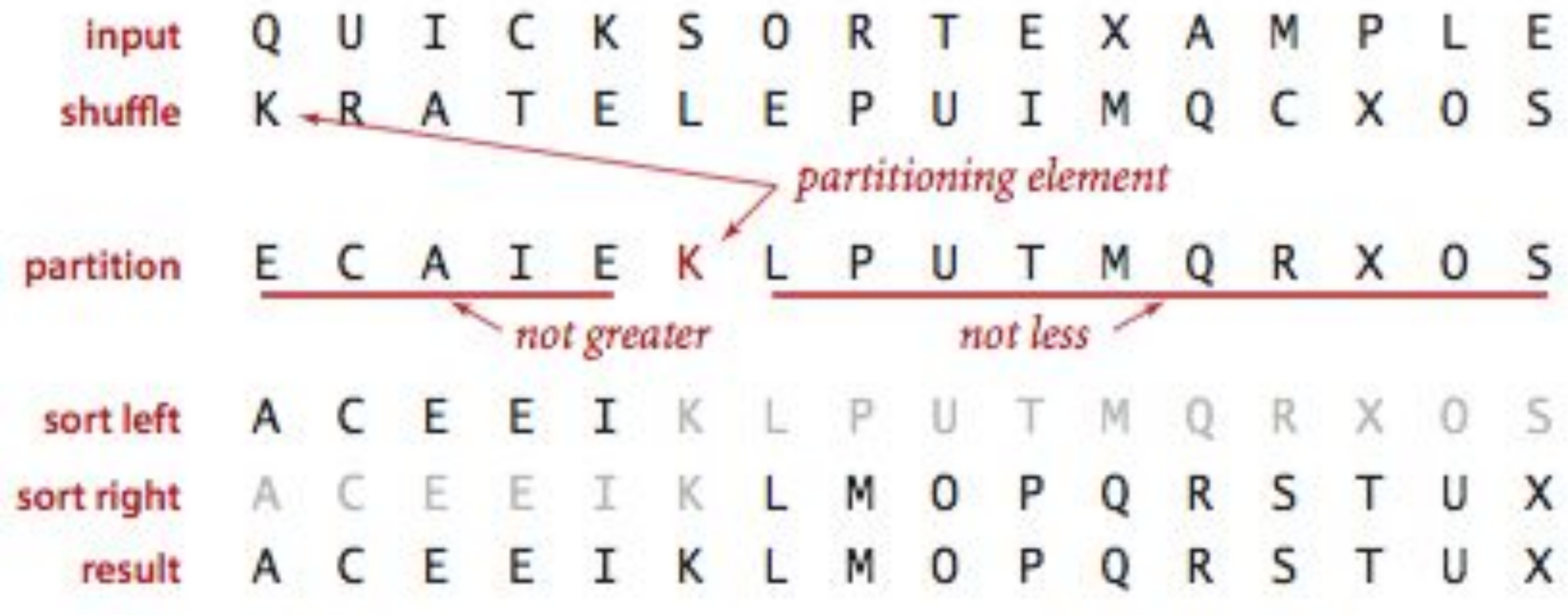
sourcemind

Quicksort

Quicksort is an efficient and popular sorting algorithm created by Tony Hoare in 1959. It is usually performs better than many other sorting algorithms (**even better when the input array is randomized**) and requires time proportional to N log N on the average to sort N items,

The basic algorithm

Quicksort is a divide-and-conquer method for sorting. It works by **partitioning** an array into two parts, then sorting the parts independently.


input Q U I C K S O R T E X A M P L E
shuffle K R A T E L E P U I M Q C X O S
partitioning element
partition E C A I E K L P U T M Q R X O S
not greater
not less
sort left A C E E I K L P U T M Q R X O S
sort right A C E E I K L M O P Q R S T U X
result A C E E I K L M O P Q R S T U X

Quicksort overview

Partitioning

The crux of the method is the partitioning process, which rearranges the array to make the following three conditions hold:

- The entry `a[j]` is in its final place in the array, for some `j`.
- No entry in `a[lo]` through `a[j-1]` is greater than `a[j]`.
- No entry in `a[j+1]` through `a[hi]` is less than `a[j]`.

We achieve a complete sort by partitioning, then recursively applying the method to the subarrays. It is a randomized algorithm, because it randomly shuffles the array before sorting it.

sourcemind

Partitioning explanation

There is a very nice video that explains the partition process.

Quicksort: Partitioning an array - YouTube

Partition pseudocode

```
function partition(a: int[], lo: int, hi: int):
     pivot = a[hi]
     i = lo - 1
     for j = lo to hi - 1
          if a[j] < pivot:
                 i += 1
                 swap(a[i], a[j])
     swap(arr[i+1], arr[hi])
     return i + 1
```

sourcemind

Quicksort pseudocode

```
function qsort(a: int[], lo: int, hi: int):
    if lo < hi:
        p = partition(a, lo, hi)
        qsort(a, lo, p - 1)
        qsort(a, p + 1, hi)
```

sourcemind

Qsort complexity

- Let T the function that gives the time needed by qsort to sort an array of size N.
- Partition time is proportional to N.
- Then sorting the two halves is 2 * T(N / 2).
- T(N) = 2 * T(N / 2) + N
- **T(N) = N $\log_2$ N.** **Master theorem**

Master theorem

In the analysis of algorithms, the master theorem for divide-and-conquer recurrences provides an asymptotic analysis (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms[Wikipedia].

sourcemind

RECURRENCE	$T(n)$	EXAMPLE
$T(n) = T(n/2) + 1$	$\sim \log_2 n$	*binary search*
$T(n) = 2T(n/2) + n$	$\sim n \log_2 n$	*mergesort*
$T(n) = T(n-1) + n$	$\sim \frac{1}{2}n^2$	*insertion sort*
$T(n) = 2T(n/2) + 1$	$\sim n$	*tree traversal*
$T(n) = 2T(n-1) + 1$	$\sim 2^n$	*towers of Hanoi*
$T(n) = 3T(n/2) + \Theta(n)$	$\Theta(n^{\log_2 3}) = \Theta(n^{1.58...})$	*Karatsuba multiplication*
$T(n) = 7T(n/2) + \Theta(n^2)$	$\Theta(n^{\log_2 7}) = \Theta(n^{2.81...})$	*Strassen multiplication*
$T(n) = 2T(n/2) + \Theta(n \log n)$	$\Theta(n \log^2 n)$	*closest pair*

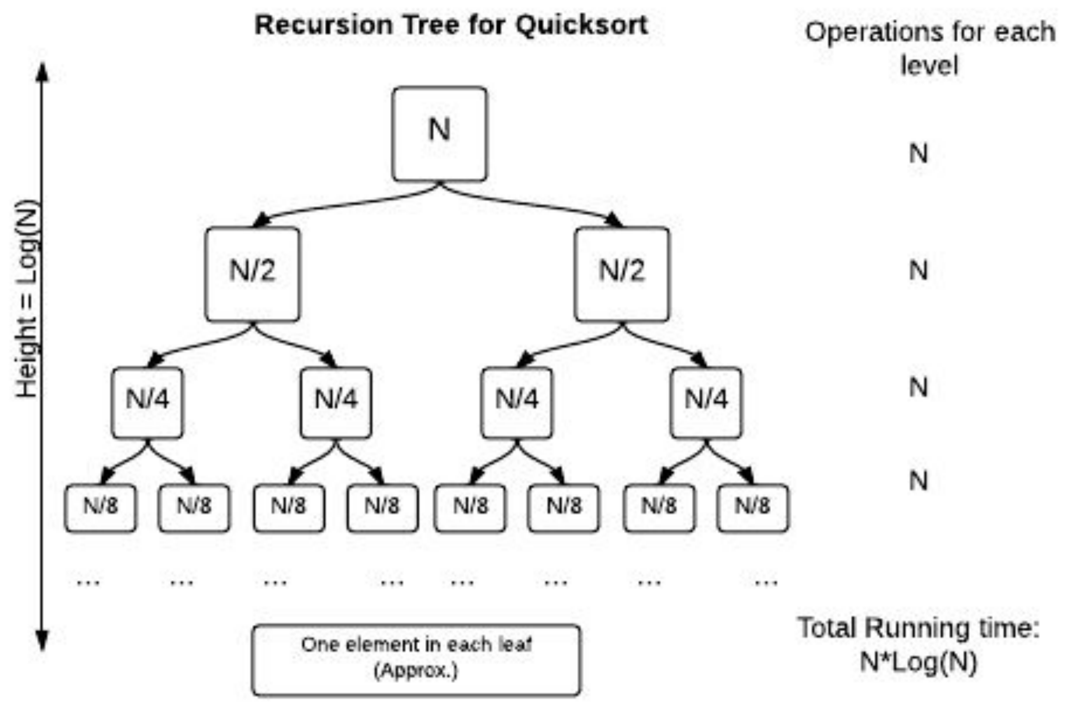
Source: https://algs4.cs.princeton.edu/23quicksort/

sourcemind

Recursion tree method

Recursion tree method is used to solve recurrence relations.

Generally, these recurrence relations follow the divide and conquer approach to solve a problem. Like $T(N) = 2 * T(N / 2) + N$.



Recursion Tree for Quicksort
Operations for each level
Height = Log(N)
N
N/2
N/2
N/4
N/4
N/4
N/4
N/8
N/8
N/8
N/8
N/8
N/8
N/8
N/8
...
One element in each leaf (Approx.)
N
N
N
N
Total Running time: N*Log(N)

Source: https://www.hackerrank.com/challenges/quicksort4/problem

Qsort pitfalls

- Worst case complexity is $O(N^2)$.
- It happens when the pivot is the smallest element or the array is already sorted.
- Hence the need to shuffle the array.

Other sorting algorithms

- Mergesort
- Heapsort
- Timsort (used by Python)

Stable and unstable sorts

An important property of a sorting algorithm is **stability**.

A sort is stable if for two elements with the same value, their original order is preserved in the sorted array.

Stable vs unstable sorts

Stable	Unstable
Bubble sort	Quicksort
Mergesort	Heapsort
Timsort	