

# Introduction to Algorithms and Data Structures

## Lecture 13: QuickSort

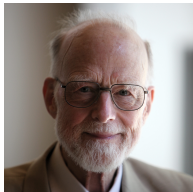
Mary Cryan

School of Informatics  
University of Edinburgh

5th Nov 2024

# QuickSort

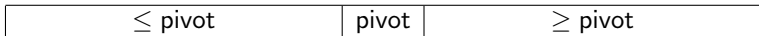
Invented by British computer scientist Tony Hoare in 1960 while studying in Moscow, published in 1961.



Divide-and-Conquer algorithm:

1. If the input array has  $<$  two elements, do nothing.

Otherwise, call Partition: Pick a pivot key and use it to divide the array into two:



2. Sort the two subarrays recursively.

# QuickSort

**Algorithm** QuickSort( $A, p, r$ )

1. **if**  $p < r$  **then**
2.      $split \leftarrow$  Partition( $A, p, r$ )
3.     QuickSort( $A, p, split - 1$ )
4.     QuickSort( $A, split + 1, r$ )

# Partition

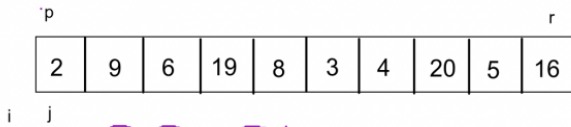
**Algorithm** Partition( $A, p, r$ )

1.  $pivot \leftarrow A[r].key$
2.  $i \leftarrow p - 1$
3. **for**  $j \leftarrow p$  **to**  $r - 1$  **do**
4.     **if**  $A[j].key \leq pivot$
5.          $i \leftarrow i + 1$
6.         exchange  $A[i]$  and  $A[j]$
7. exchange  $A[i + 1]$  and  $A[r]$
8. **return**  $i + 1$

**Invariant:**  $i$  is 1 less than the leftmost  $> pivot$  value in the range  $p \dots j$  (or is  $j - 1$  if no  $> pivot$  is there).

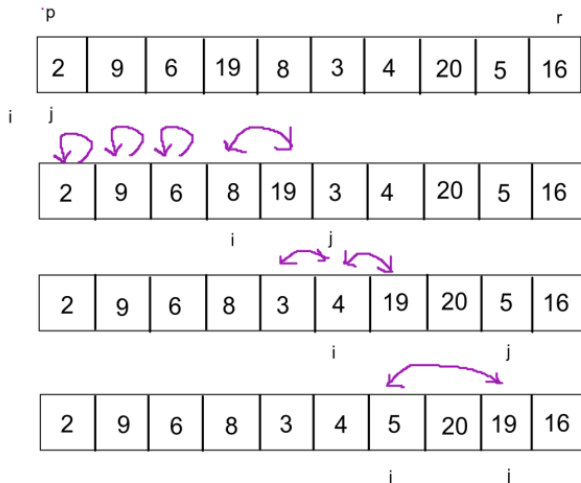
## Partition discussion ( $i$ and $j$ )

## Partition example



# Partition example (done on video)

pivot ← 16

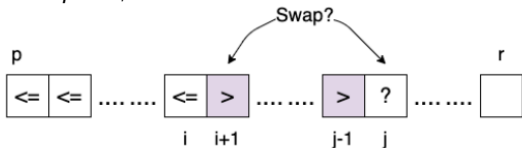


Finally we will swap  $A[i+1]$  (20) with  $A[r]$  (16) and return  $(i+1)$

## Correctness of Partition

Assume we are part-way through Partition and *for this j* (and this *i*), we have the Invariant (all  $A[i + 1 \dots j - 1]$  are  $> pivot$ ). Induction step:

- ▶ If  $A[j].key > pivot$ , the algorithm changes nothing.  
And the range of  $> pivot$  cells expands by 1 (as  $j$  gets incremented).
- ▶ If  $A[j].key \leq pivot$ , we will swap  $A[j]$  with  $A[i + 1]$ .  $A[i + 1].key$  either ...
  - ▶ was  $\leq pivot$  (and  $i = j - 1$ ), in which case  $i' = i + 1$  is  $j$  and we swap  $A[j]$  with itself, and then have the same pattern after  $j$  increments, or
  - ▶ was  $> pivot$ , in which case ...



- ▶ After loop, Invariant implies that the first  $> pivot$  (if any) is in  $A[i + 1]$ .  
Hence swapping  $A[r]$  and  $A[i + 1]$  gives a “partition” of the desired form.

## Results from Partition

- ▶ We might get a fairly balanced partition, with “pivot” lying near the middle (“*split*” roughly halfway between  $p$  and  $r$  ).
  - ▶ When this happens, the Divide-and-Conquer balance is like MergeSort (and also, we have  $\Theta(n)$  work at the “top level”)
- ▶ Alternatively, we could get a very unbalanced partition, with one side empty or very small
  - ▶ Then we are doing linear work to reduce the size of the problem to be solved only a tiny bit. More like BubbleSort.
- ▶ Or anything in between, depends on the original arrangement of keys (and what “rank” *pivot* has among the keys in the array).

# Running Time of QuickSort

## Partition

$$T_{\text{partition}}(n) = \Theta(n)$$

## QuickSort

$$\begin{aligned} T_{\text{quickSort}}(n) &\in \max_{1 \leq s \leq n-1} (T_{\text{quickSort}}(s) + T_{\text{quickSort}}(n-s-1)) \\ &\quad + T_{\text{partition}}(n) + \Theta(1) \\ &= \max_{1 \leq s \leq n-1} (T_{\text{quickSort}}(s) + T_{\text{quickSort}}(n-s-1)) + \Theta(n). \end{aligned}$$

## Implies

$$T_{\text{quickSort}}(n) \in \Theta(n^2)$$

To show  $\Omega(n^2)$  you need a specific structured input (not too hard).

# QuickSort

*The average-case running-time of a sorting algorithm is the average number of computational steps (comparisons) carried out on a uniform random permutation of the keys  $\{1, \dots, n\}$ .*

- ▶ We don't say "amortized" as we have a single computation, and are comparing input of exactly the same size.
- ▶ For sorting-algorithms, typically the running-time can be captured by the number of "comparisons" (these measures tend to be asymptotically equivalent).
- ▶ *Uniform random permutation* means all permutations arise with same probability.

The average-case running time of QuickSort is  $\Theta(n \lg(n))$ .

# QuickSort

- ▶ QuickSort can be very fast in practice.
- ▶ But performs badly —  $\Theta(n^2)$  — on sorted and almost sorted arrays.

## Practical Improvements

- ▶ Different choice of pivot (key of middle item, random)
- ▶ Refined partitioning routine
- ▶ Use InsertionSort for small arrays (similar to “TimSort”)

## RandomQuickSort

- ▶ The  $\Theta(n \lg(n))$  result for average-case can be shown to carry over characterize (expected) running-time for RandomQuickSort (choose the pivot randomly).

# Sorting in Python



The default sorting algorithm in python used to be “Timsort”, an optimized version of MergeSort developed by Tim Peters, a major contributor to the development of CPython.

- ▶ It was Python’s sort between versions 2.3 and 3.10 inclusive
- ▶ Does a pre-processing step looking for “runs” of **strictly decreasing** or **(non-strict) increasing** items.
- ▶ We understand how to handle “runs” (without sorting).
- ▶ Then merge the sorted runs, using InsertSort for short subarrays, and MergeSort for bigger subarrays.

# Sorting in Python since 3.11



In 2021 (version 3.11), Python switched to Munro and Wild's PowerSort.

- ▶ A “run-adaptive” MergeSort which exploits *runs* of sorted items to identify natural sub-arrays for merging.
- ▶ Extra structure of a binary tree with *runs* as leaves to encode the most efficient pairing structure of runs at lower levels for merging operations.
- ▶ Wild is running an online contest where programmers submit lists where Powersort and TimSort most differ:

<http://powersort-competition.github.io/PowersortCompetitionWebsite>

# Reading Material

## Personal Reading:

- ▶ QuickSort and its analysis, sections 7.1, 7.2 and 7.4 of [CLRS]
- ▶ QuickSort can also be found in “Algorithms Illuminated”, Sections 5.1 and 5.3, however the version of Partition (still  $\Theta(n)$  time, and still partitioning wrt a pivot) is different to ours.
- ▶ “QuickShort” interview with Tony Hoare can be viewed at <http://anothercasualcoder.blogspot.com/2015/03/my-quickshort-interview-with-sir-tony.html>
- ▶ Can read about PowerSort in `listsort.txt` file of `Objects` directory in the Python documentation (need to skip past lots of TimSort chat): <https://github.com/python/cpython/blob/main/Objects/listsort.txt>

