Quicksort

What it is

How it is implemented

Quicksort

Widely-used, fast, in-place sorting algorithm

Devised by Tony Hoare in Moscow (1959) 

Won wager with boss (who advocated shellsort)

Published in ALGOL using recursion (1961)

Note: Quicksort not a stable sort (sorting does not preserve input order of identical keys)

Quicksort complexity (for n data items)

Average case: $\rightarrow O(n \log n)$

Worst-case: $\rightarrow O(n^2)$

Storage: $\Theta(n)$ for items, sort: $O(\log n)$ avg. case, $O(n)$ worst case stack usage $O(\log n)$ with tail call optim.

Quicksort implementations are typically cache-friendly (linear scans suit caches)

Employed in hybrids (e.g. Introsort - falls back to heapsort to avoid $O(n^2)$ worst-case behaviour)

References/Notes/Image credits:

C. A. R. Hoare, "Algorithm 64: quicksort", *Communications of the ACM* (1961)

(Portrait of Hoare) http://curation.cs.manchester.ac.uk/Turing100/www.turing100.manchester.ac.uk/speakers/invited-list/11-speakers/39-hoare.html

(Quicksort history) https://anothercasualcoder.blogspot.com/2015/03/my-quickshort-interview-with-sir-tony.html

(Elliot Brothers Ltd image) https://en.wikipedia.org/wiki/Elliott_Brothers_(computer_company)#/media/File:Elliott_Sector.jpg

Storage complexity of quicksort: https://stackoverflow.com/a/29746572 (technically you can guarantee $O(\log n)$ worst case memory if you have tail call optimisation)

(Introsort) https://en.wikipedia.org/wiki/Introsort

Quicksort Overview

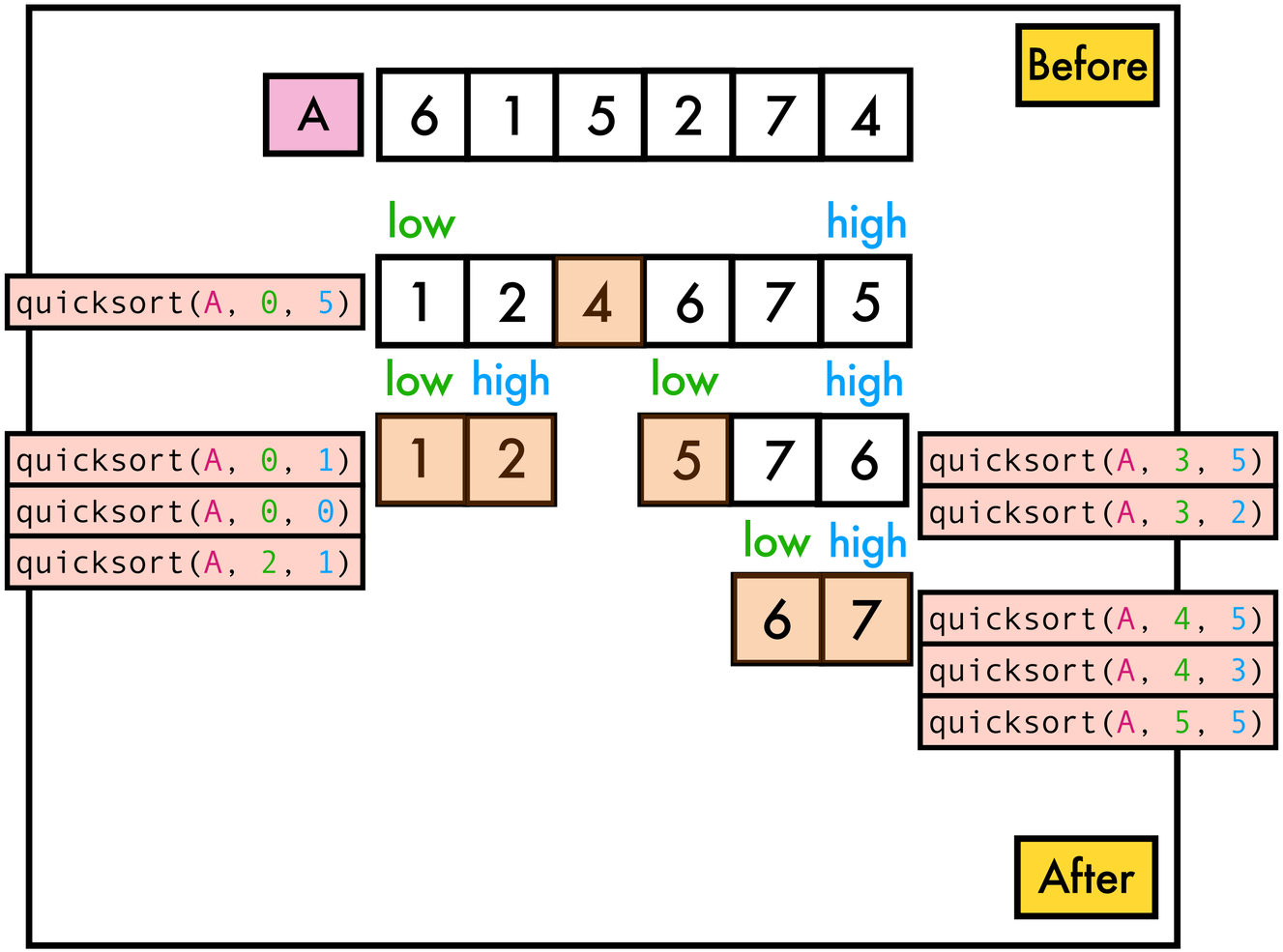
Quicksort

Uses a recursive, divide-and-conquer strategy:

1. Select a pivot from the array
2. Partition array into 3 subarrays arranged as:

3. Recursively quicksort the first and last subarrays


```
def quicksort(A, low, high):
    if low < high:
        pivot = partition(A, low, high)
        quicksort(A, low, pivot-1) # left subarray
        quicksort(A, pivot+1, high) # right subarray
```



References:
J. Erickson, "Algorithms", Chap. 2, http://algorithms.wtf/ (2019)
T. Cormen et al., "Introduction to algorithms", Chap 7.1, MIT press (2022)
L. Xinyu, "Elementary Algorithms", Chap. 13 (2022)

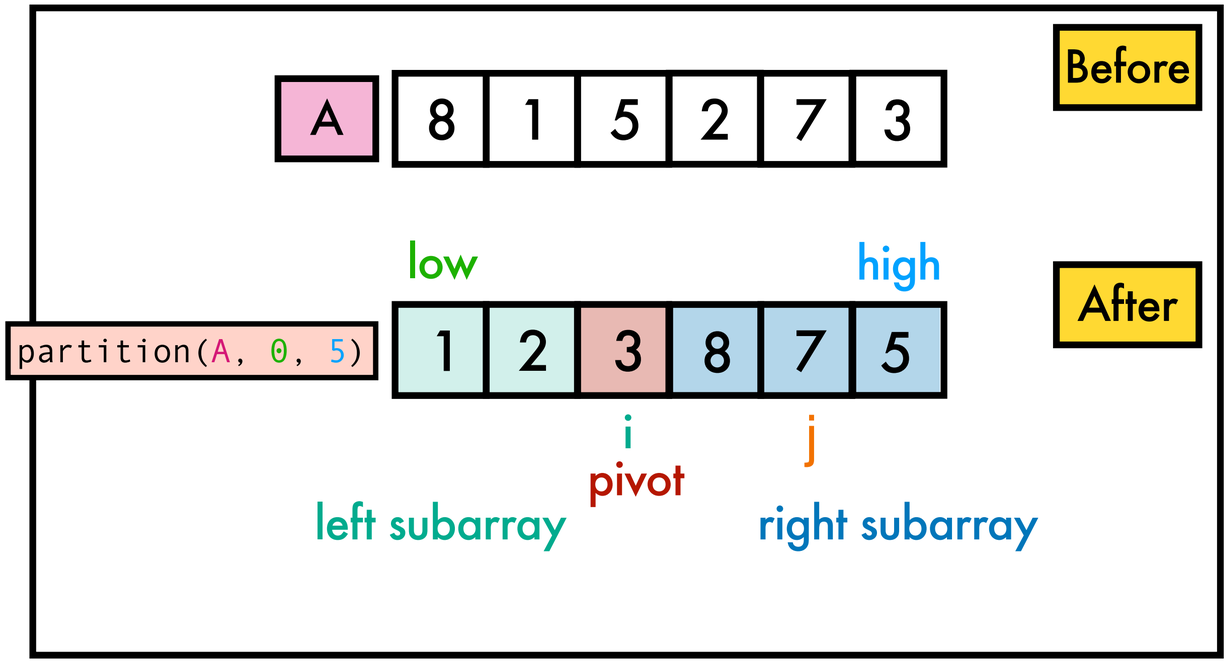
Lomuto Partition

Our first partition function (Lomuto)

partition() does most of the work in quicksort

```
def partition(A, low, high): # lomuto partition
    pivot = high
    pivot_val = A[pivot]
    i = low
    for j in range(low, high):
        if A[j] <= pivot_val:
            A[i], A[j] = A[j], A[i]
            i = i + 1
    A[i], A[pivot] = A[pivot], A[i]
    return i # new pivot
```

Complexity: $\Theta(n)$ where $n = \text{high} - \text{low} + 1$



References:
J. Erickson, "Algorithms", Chap. 2, http://algorithms.wtf/ (2019)
T. Cormen et al., "Introduction to algorithms", Chap 7.1, MIT press (2022)
L. Xinyu, "Elementary Algorithms", Chap. 13 (2022)

Hoare Partition

Our second function (Hoare)

```
def partition(A, low, high): # CLRS-style hoare partition
    pivot_val = A[low]
    i = low - 1
    j = high + 1
    while True:
        while True:
            i += 1
            if A[i] >= pivot_val:
                break
        while True:
            j -= 1
            if A[j] <= pivot_val:
                break
        if i >= j:
            return j # new pivot
        A[i], A[j] = A[j], A[i]
```

Modify quicksort() to use this partition():

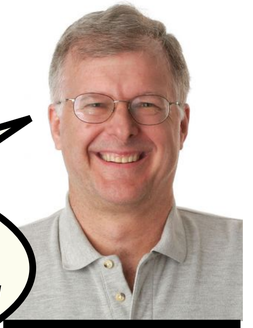
```
# lomuto left subarray recursion
quicksort(A, low, pivot-1)
# hoare left subarray recursion
quicksort(A, low, pivot)
```

Complexity: $\Theta(n)$ where $n = \text{high} - \text{low} + 1$

Fewer swaps than lomuto partition

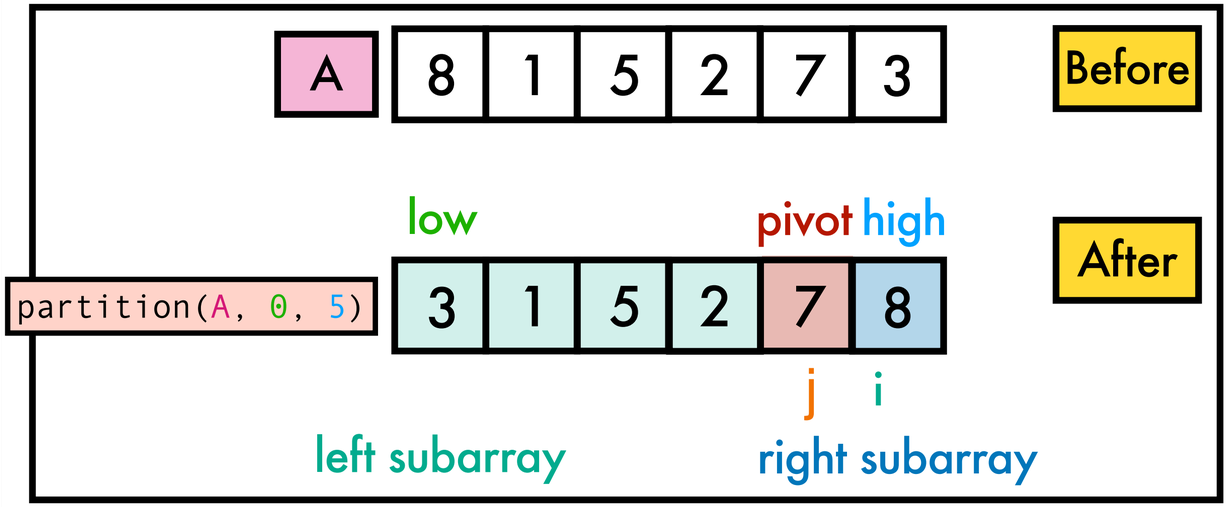
"one of the places off-by-one errors go to die"

Jeff Erickson



"I once spent the better part of two days chasing down a bug..."

Jon Bentley



References/image credit:
https://en.wikipedia.org/wiki/Quicksort#Hoare_partition_scheme
(CLRS) T. Cormen et al., "Introduction to algorithms", Chap 7, MIT press (2022)
(sketch of Jeff Erickson by Damien Erickson) https://jeffe.cs.illinois.edu/
(Quote about off-by-one-errors) J. Erickson, "Algorithms", Chap. 2, http://algorithms.wtf/ (2019)
J. Bentley, "Programming pearls: how to sort", *Communications of the* ACM (1984)
(Image of Jon Bentley) https://engineering.lehigh.edu/dac/jon-bentley

Worst-Case Analysis

Unbalanced partitions and quadratic complexity

How can things go badly for quicksort?

We have seen that `partition()` is $\Theta(n)$

Since quicksort is recursive, can write complexity:

$$T(n) = \underbrace{T(r-1)}_{\text{left subarray}} + \underbrace{T(n-r)}_{\text{right subarray}} + \underbrace{\Theta(n)}_{\text{partition}}$$

where r is the rank of the pivot

If we choose the pivot as smallest value ($r = 1$)

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

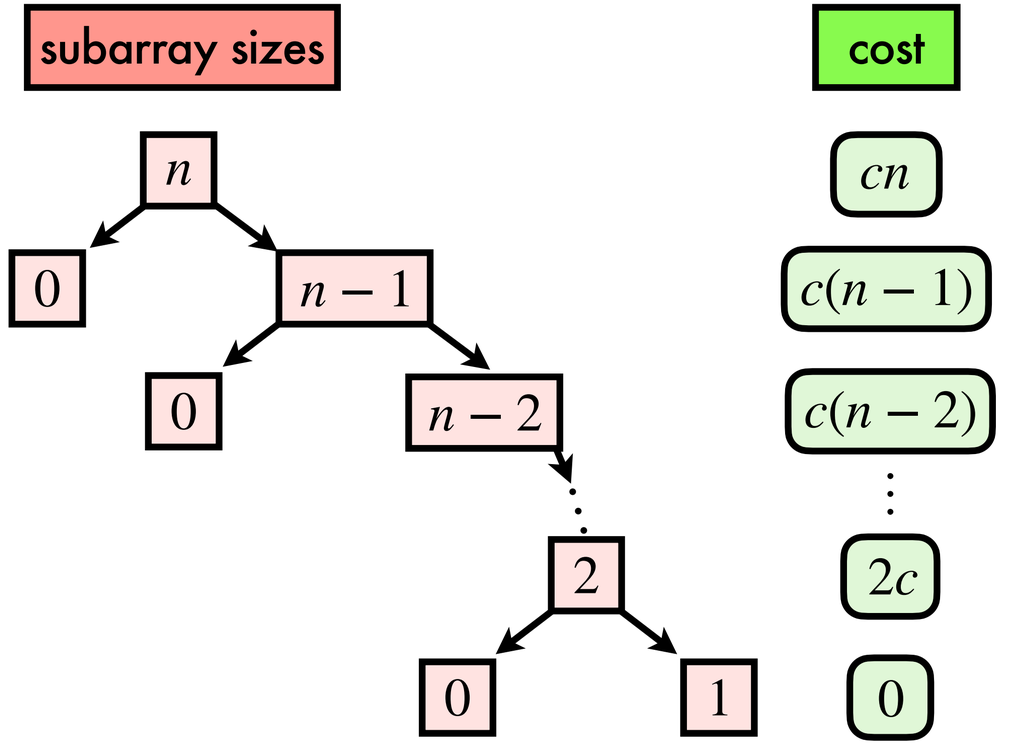
$T(0) = O(1)$ (empty subarray - no comparisons)

$$T(n) = T(n-1) + \Theta(n) \implies T(n) = \Theta(n^2)$$

$T(n) = \Theta(n^2)$ for fixed r not dependant on n

Illustrate runtime complexity with $r = 1$

Partition cost $\Theta(n)$ as cn for some constant c



Total cost $= c(n + (n-1) + (n-2) + \ldots + 2 + 0)$

$$= c\sum_{i=2}^{n} i \;\; = c(n(n+1)/2 - 1) \implies \Theta(n^2)$$

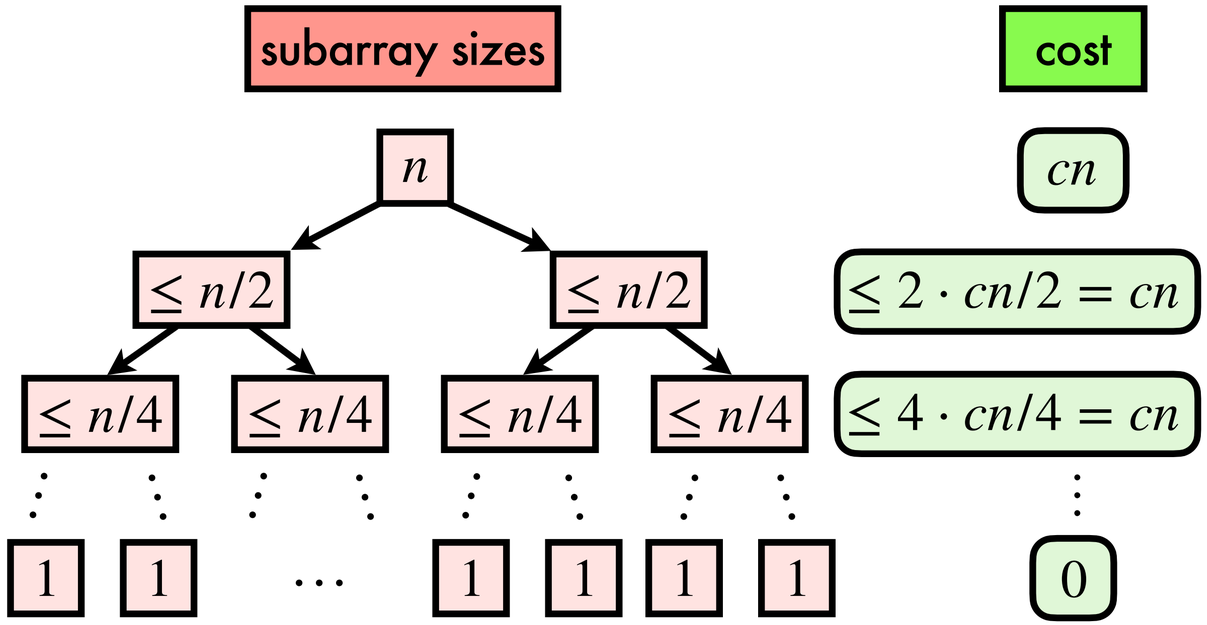
References:
J. Erickson, "Algorithms", Chap. 2, http://algorithms.wtf/
T. Cormen et al., "Introduction to algorithms", Chap 7, MIT press (2022)
https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Balanced Partition Analysis

Quicksort is happy with $r = K \cdot n$ for $0 < K < 1$

By happy, we mean $O(n \log n)$

Consider the simple case when $K = 0.5$:



Tree height is $\log n$ and cost at each level $\leq cn$

Total cost $\leq cn \cdot \log n \implies O(n \log n)$

For complexity, what matters is tree depth growth

Suppose $K = 0.995$ (seems unbalanced!)

Tree depth is $\log_{\frac{1000}{995}} n$

So the complexity is still $O(n \log n)$

Some intuition for why "most" partitions will produce $O(n \log n)$ behaviour

References:
Fig. based on https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort

Engineering Improvements

Behaviour on concerningly common inputs

If array already sorted, both Lomuto and Hoare

partitions have worst case behaviour $O(n^2)$

If all elements the same, Lomuto is also $O(n^2)$

"Median-of-three"

A simple heuristic to get a better pivot:

pick median of first, middle and last elements

Useful on sorted input arrays (and other inputs)

Does not rule out worst case behaviour

Randomised quicksort

Pick pivot uniformly at random and swap it with:

element at index 0 (if using Lomuto partition)

element at index $n-1$ (if using Hoare partition)

Expected runtime $O(n \log n)$

The full proof is quite involved (see refs)

Worst case becomes very unlikely (widely used)

Provoking $O(n^2)$ behaviour

"Killer adversary for quicksort" (McIlroy, 1999)

Decide ordering of elements lazily during sorting

Achieving $O(n \log n)$ worst case

Can ensure $O(n \log n)$ worst case quicksort via

$O(n)$ median-of-medians algorithm

In practice, this is too slow, so not widely used

References:

(Median-of-3) R. Sedgewick, "Algorithms in C: Fundamentals, Data Structures, Sorting, Searching" (1998)

(Proof for randomised quicksort) https://www.cs.cmu.edu/~avrim/451f11/lectures/lect0906.pdf

T. Cormen et al., "Introduction to algorithms", Chap 7, MIT press (2022)

D. M. McIlroy, "A killer adversary for quicksort", Software: Practice and Experience (1999)

(quickselect) C. A. R. Hoare, "Algorithm 65: Find", *Communications of the ACM* (1961)