Arrays: Quicksort

Agenda

- Explore the Quick sort algorithm

- Understand the following aspects
  - Algorithm mechanism and pseudocode
  - Algorithm iterations on varying input
  - Algorithm time and space complexity

Quick sort

- A “randomized” sort algorithm:
    - Addresses performance issues with Merge sort.
    - Performs well in most scenarios, badly very rarely.

- Algorithm scheme:
    i. Randomly shuffle the given array A[0...N-1].
    ii. Partition the array into 2 pieces, as follows:
        - For some index j, A[j] is in-place.
        - No element to the left of j is larger than A[j].
        - No element to the right of j is smaller than A[j].
    iii. Sort each piece recursively.

Quick sort

- Characteristics:
  - Quick sort is a comparison-based sort.
  - Quick sort can sort an array in-place.
  - Auxiliary array not required.
  - Starts by randomly shuffling the order of elements.

- Comparison operation:
    - Defined as required for the data type
      - Numbers
      - Strings
      - Objects: by attributes

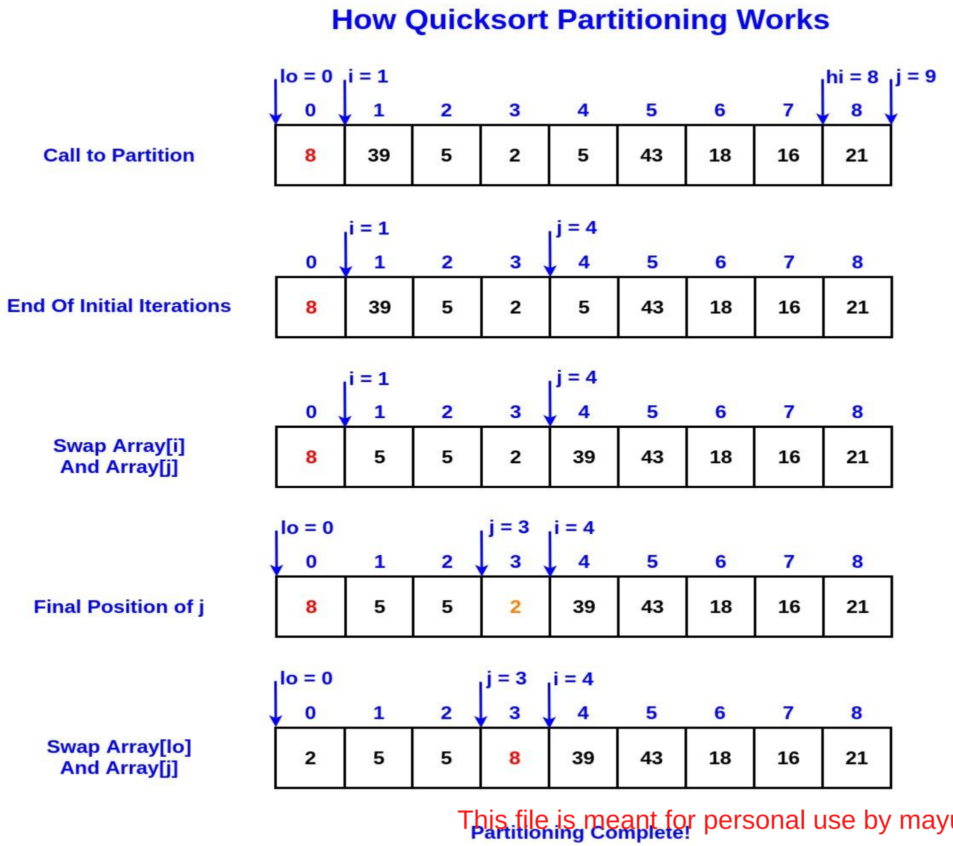
Quick sort

1. Shuffle the array.

2. Partition this array, so that, for some j:
    a. Array[j] is in place.
    b. All entries to the left of j are smaller.
    c. All entries to the right of j are larger.

3. Recursively sort each piece.

Quick sort: Partition



Quick sort pseudocode

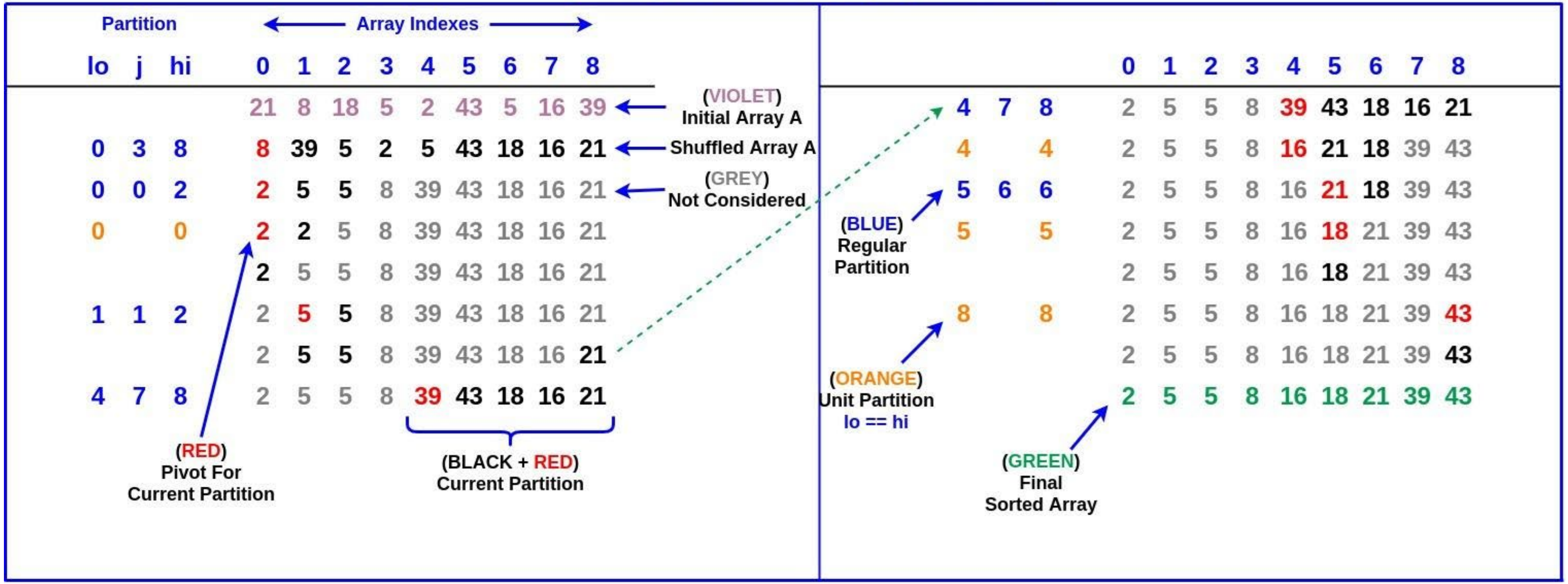
Code For Quick Sort

```
import random

def swap(array, i, j):
    temp = array[i]
    array[i] = array[j]
    array[j] = temp

def partition(array, lo, hi):
    i = lo
    j = hi+1

    while True:
        i += 1
        while array[i] < array[lo]:
            if i == hi:
                break

            i += 1

        j -= 1
        while array[lo] < array[j]:
            if j == lo:
                break

            j -= 1

        if i >= j:

            break

        swap(array, i, j)

    swap(array, lo, j)
    return j
```

```
def quick_sort(array, lo, hi):
    if hi <= lo:
        return

    j = partition(array, lo, hi)

    quick_sort(array, lo, j-1)
    quick_sort(array, j+1, hi)


def wrapper_sort(array):
    random.shuffle(array)
    quick_sort(array, 0, len(array)-1)
```

Quick sort iterations

Quick Sort Iterations On Array A
(Always Selecting A[lo] As Pivot)



Quick sort iterations

Quick Sort Iterations On Pre-Sorted Array A (Always Selecting A[lo] As Pivot)

Partition			← Array Indexes →									
lo	j	hi	0	1	2	3	4	5	6	7	8	
			2	5	5	8	16	18	21	39	43	← (VIOLET) Initial Array A
0	6	8	21	18	5	5	39	2	43	8	16	← Shuffled Array A
0	3	5	8	18	5	5	16	2	21	43	39	
0	1	2	5	2	5	8	16	18	21	43	39	
0	1	1	2	5	5	8	16	18	21	43	39	
			2	5	5	8	16	18	21	43	39	
4	4	5	2	5	5	8	16	18	21	43	39	
			2	5	5	8	16	18	21	43	39	
7	8	8	2	5	5	8	16	18	21	43	39	
			2	5	5	8	16	18	21	39	43	
(GREEN) Final Sorted Array →			2	5	5	8	16	18	21	39	43	

Quick sort iterations

Quick Sort Iterations On Reverse-Sorted Array A
(Always Selecting A[lo] As Pivot)

Partition			← Array Indexes →									
lo	j	hi	0	1	2	3	4	5	6	7	8	
			43	39	21	18	16	8	5	5	2	← (VIOLET) Initial Array A
0	4	8	16	18	2	8	39	21	43	5	5	← Shuffled Array A
0	3	5	5	5	2	8	16	21	43	39	18	
0	0	3	2	5	5	8	16	21	43	39	18	
0		0	2	5	5	8	16	21	43	39	18	
			2	5	5	8	16	21	43	39	18	
2	2	3	2	5	5	8	16	21	43	39	18	
			2	5	5	8	16	21	43	39	18	
5	6	8	2	5	5	8	16	21	43	39	18	
5		5	2	5	5	8	16	18	21	39	43	
			2	5	5	8	16	18	21	39	43	
7	7	8	2	5	5	8	16	18	21	39	43	
			2	5	5	8	16	18	21	39	43	
(GREEN) Final Sorted Array →			2	5	5	8	16	18	21	39	43	

Quick sort: Complexity

- Analyzing time complexity:
  - Note that the input is always randomly shuffled.
  - This does not eliminate worst behaviour of Quicksort, but greatly reduces its probability.

- Worst case: The total number of compares and entry swaps (during partitioning) is:

$C(N) = N + (N-1) + (N-2) + \ldots + 1$: proportional to N^2 steps

- Average case: Proportional to $N\log_2 N$ steps

Quick sort: Complexity

- Analyzing space complexity:
  - Quicksort is an in-place sorting algorithm.
    - Extra space is not used during the sorting.

  - Best case: Constant space
  - Worst case: Constant space
  - Average case: Constant space

Summary

We explored the Quick sort algorithm as follows:

- Algorithm mechanism and pseudocode
- Algorithm iterations on varying input
- Time and space complexity

Thank You