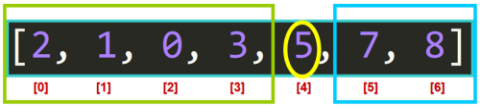
Algorithm: Quicksort

Quicksort

Key Aspects:

- Quicksort is an in-place comparison sorting algorithm, so no extra space is required.
- The implementation uses two functions: quicksort() and partition().
- It is efficient to sort very large lists.
- It works recursively by selecting a "pivot" that divides the list into two portions.
- The pivot chosen determines the exact efficiency of the algorithm for a specific input.
- You can choose the pivot either randomly, by selecting the median of three numbers in the list, or by using the first or last element in the list.

The list is sorted in ascending order



Algorithm:

- The function quicksort() has three parameters: the list that will be sorted in memory, the lower bound, and the upper bound that describe the interval (indices) that you wish to sort.
- If the lower bound < upper bound, call the partition() function.
- Partition() selects a pivot and creates a partition of the list with all the elements that are smaller than the pivot to the left of the pivot and all the elements that are greater than the pivot to the right of the pivot.
- Partition() returns the final index of the pivot once it has been placed in the correct position in the list.
- Using this pivot index, quicksort() calls itself recursively twice, the first call sorts the left portion of the list and the second call sorts the right portion of the list.

Time Complexity:

- Worst-Case Time Complexity: Quadratic O(n^2)
- Average-Case Time Complexity: Log-Linear O(nlog(n))
- Best-Case Time Complexity: Log-Linear O(nlog(n))

Quicksort

Code:

Calling partition()

```
def quicksort(lst, low, high):
    # If the interval [low, high] is valid.
    if low < high:
        # Partition the list and get
        # the final index of the pivot element.
        pivot_index = partition(lst, low, high)

        # Sort the left and right
        # portions of the list (determined
        # by the final position of the pivot).
        quicksort(lst, low, pivot_index-1)
        quicksort(lst, pivot_index+1, high)
```

Recursive Calls

Quicksort

Code:

```
def partition(lst, low, high):
    # Select the last element as the pivot
    pivot = lst[high]

    i = low - 1

    for j in range(low, high):
        # If the current element is
        # smaller than or equal to pivot
        if lst[j] <= pivot:
            i += 1
            lst[i], lst[j] = lst[j], lst[i]

    # Swap the pivot with
    # the element at index i+1
    lst[i+1], lst[high] = lst[high], lst[i+1]
    return i+1
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
=======> Starting quicksort()
List: [4, 1, 8, 2, 6, 3]
Low (index): 0
High (index): 5
Is the interval [0, 5] valid (has more than 1 element)? Yes

---> Entering partition()
The pivot is the element: 3
Located at index: 5
i is initialized to: -1

-> Starting the for loop

Value of j: 0
Is the current element lst[0] (4) <= to the pivot (3)?
No, it isn't! We don't need to make any changes

Value of j: 1
Is the current element lst[1] (1) <= to the pivot (3)?
Yes, it is!

So we need to increment the value of i
Old value of i: -1
New value of i: 0

And we need to swap the element at index i (index 0): 4
with the element at index j (index 1): 1

Swapping...
Old list: [4, 1, 8, 2, 6, 3]
New list: [1, 4, 8, 2, 6, 3]
Swap completed

Value of j: 2
Is the current element lst[2] (8) <= to the pivot (3)?
No, it isn't! We don't need to make any changes

Value of j: 3
Is the current element lst[3] (2) <= to the pivot (3)?
Yes, it is!
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
So we need to increment the value of i
Old value of i: 0
New value of i: 1

And we need to swap the element at index i (index 1): 4
with the element at index j (index 3): 2

Swapping...
Old list: [1, 4, 8, 2, 6, 3]
New list: [1, 2, 8, 4, 6, 3]
Swap completed

Value of j: 4
Is the current element lst[4] (6) <= to the pivot (3)?
No, it isn't! We don't need to make any changes

--> Out of the for loop

Now we need to move the pivot to index i+1 (index 2)

Swapping...
Swapping the pivot (3) with the element at index 2 (8)

Old list: [1, 2, 8, 4, 6, 3]
New list: [1, 2, 3, 4, 6, 8]
Swap completed!

Returning the index of the pivot element after the swap: 2

This generates a partition of the list:
List: [1, 2, 3, 4, 6, 8]
Left sublist: [1, 2] where all the elements are smaller than the pivot.
Right sublist: [4, 6, 8] where all the elements are greater than the pivot.
The pivot element is in the middle.

Partition Completed!
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
-> Back to quicksort()

The pivot index is: 2
Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 0
High: 1

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 0
High (index): 1
Is the interval [0, 1] valid (has more than 1 element)? Yes

---> Entering partition()
The pivot is the element: 2
Located at index: 1
i is initialized to: -1

-> Starting the for loop

Value of j: 0
Is the current element lst[0] (1) <= to the pivot (2)?
Yes, it is!

So we need to increment the value of i
Old value of i: -1
New value of i: 0

And we need to swap the element at index i (index 0): 1
with the element at index j (index 0): 1

Swapping...
Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed

--> Out of the for loop
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
Now we need to move the pivot to index i+1 (index 1)

Swapping...
Swapping the pivot (2) with the element at index 1 (2)

Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed!

Returning the index of the pivot element after the swap: 1

This generates a partition of the list:
List: [1, 2, 3, 4, 6, 8]
Left sublist: [1] where all the elements are smaller than the pivot.
Right sublist: [] where all the elements are greater than the pivot.
The pivot element is in the middle.

Partition Completed!

-> Back to quicksort()

The pivot index is: 1
Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 0
High: 0

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 0
High (index): 0
Is the interval [0, 0] valid (has more than 1 element)? No
This part of the recursive process stops.

-> Back to quicksort()
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 2
High: 1

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 2
High (index): 1
Is the interval [2, 1] valid (has more than 1 element)? No
This part of the recursive process stops.

-> Back to quicksort()

Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 3
High: 5

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 3
High (index): 5
Is the interval [3, 5] valid (has more than 1 element)? Yes

---> Entering partition()
The pivot is the element: 8
Located at index: 5
i is initialized to: 2

-> Starting the for loop

Value of j: 3
Is the current element lst[3] (4) <= to the pivot (8)?
Yes, it is!

So we need to increment the value of i
Old value of i: 2
New value of i: 3
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
And we need to swap the element at index i (index 3): 4
with the element at index j (index 3): 4

Swapping...
Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed

Value of j: 4
Is the current element lst[4] (6) <= to the pivot (8)?
Yes, it is!

So we need to increment the value of i
Old value of i: 3
New value of i: 4

And we need to swap the element at index i (index 4): 6
with the element at index j (index 4): 6

Swapping...
Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed

--> Out of the for loop

Now we need to move the pivot to index i+1 (index 5)

Swapping...
Swapping the pivot (8) with the element at index 5 (8)

Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed!

Returning the index of the pivot element after the swap: 5

This generates a partition of the list:
List: [1, 2, 3, 4, 6, 8]
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
Left sublist: [4, 6] where all the elements are smaller than the pivot.
Right sublist: [] where all the elements are greater than the pivot.
The pivot element is in the middle.

Partition Completed!

-> Back to quicksort()

The pivot index is: 5
Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 3
High: 4

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 3
High (index): 4
Is the interval [3, 4] valid (has more than 1 element)? Yes

---> Entering partition()
The pivot is the element: 6
Located at index: 4
i is initialized to: 2

-> Starting the for loop

Value of j: 3
Is the current element lst[3] (4) <= to the pivot (6)?
Yes, it is!

So we need to increment the value of i
Old value of i: 2
New value of i: 3

And we need to swap the element at index i (index 3): 4
with the element at index j (index 3): 4
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
Swapping...
Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed

--> Out of the for loop

Now we need to move the pivot to index i+1 (index 4)

Swapping...
Swapping the pivot (6) with the element at index 4 (6)

Old list: [1, 2, 3, 4, 6, 8]
New list: [1, 2, 3, 4, 6, 8]
Swap completed!

Returning the index of the pivot element after the swap: 4

This generates a partition of the list:
List: [1, 2, 3, 4, 6, 8]
Left sublist: [4] where all the elements are smaller than the pivot.
Right sublist: [] where all the elements are greater than the pivot.
The pivot element is in the middle.

Partition Completed!

-> Back to quicksort()

The pivot index is: 4
Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 3
High: 3
```

Quicksort

Example:

```
>>> a = [4, 1, 8, 2, 6, 3]
>>> quicksort(a, 0, len(a)-1)
```

```
=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 3
High (index): 3
Is the interval [3, 3] valid (has more than 1 element)? No
This part of the recursive process stops.

-> Back to quicksort()

Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 5
High: 4

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 5
High (index): 4
Is the interval [5, 4] valid (has more than 1 element)? No
This part of the recursive process stops.

-> Back to quicksort()

Calling quicksort() recursively passing:
List: [1, 2, 3, 4, 6, 8]
Low: 6
High: 5

=======> Starting quicksort()
List: [1, 2, 3, 4, 6, 8]
Low (index): 6
High (index): 5
Is the interval [6, 5] valid (has more than 1 element)? No
This part of the recursive process stops.
```