Algorithm: Merge Sort

Merge Sort

Key Aspects:

- The algorithm divides the list into two halves: left and right.
- The function is called recursively passing each half as the argument.
- The two sorted halves are merged.
- Efficient to sort small and large lists.
- Not in-place. It requires additional memory because it uses an intermediate list.

The list is sorted in ascending order

Algorithm:

- Check if the list is empty or if it only contains one element.
- If this is the case, return the list intact.
- Else, find the middle index using len(lst) // 2
- Call the merge_sort() function recursively, passing the left half of the list as the argument.
- Call the merge_sort() function recursively, passing the right half of the list as the argument.
- Merge the two halves that were returned by the recursive calls (they are already sorted ascending order).
- Return this sorted list.

Time Complexity:

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

Merge Sort
=
Merge the two halves after sorting them

Merge Sort

Code:

```python
def merge_sort(lst):

    # If the list is empty or it only contains one element
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        # Find the middle index
        middle_index = len(lst)//2

        # Call the function recursively
        # passing the left half and the right half
        # of the list in separate recursive calls
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])

        # Return the merged version of the
        # two halves already sorted in ascending order
        return merge(left, right)
```

Recursive Calls

Merge the Lists

Merge Sort

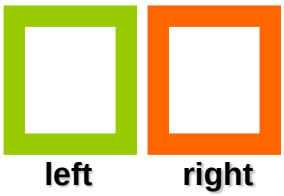
Code:

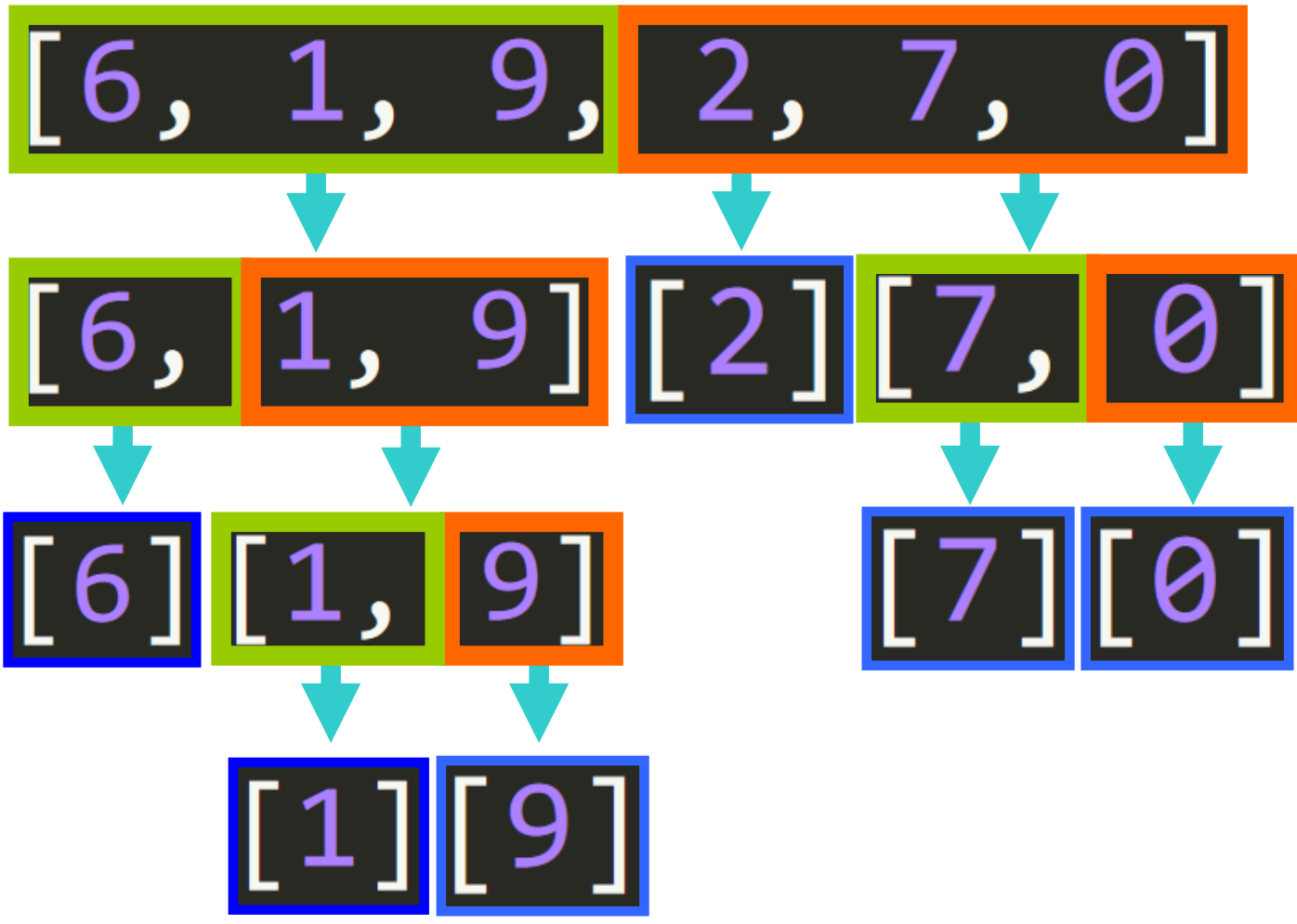
```python
# Merge the two halves
def merge(left_half, right_half):

    if not left_half or not right_half:
        return left_half or right_half

    result = []
    i, j = 0, 0

    while True:
        # If the element in the left half is smaller
        # than the element in the right half,
        # append that element to the result list
        # at the current index.
        if left_half[i]<right_half[j]:
            result.append(left_half[i])
            i += 1
        # Else, if the element in the right half is smaller,
        # add that element to the list at the current index.
        else:
            result.append(right_half[j])
            j += 1

        # To check if there were any elements left
        # because the two halves may differ in length.
        # Insert the remaining elements to the list result.
        if i == len(left_half) or j == len(right_half):
            result.extend(left_half[i:] or right_half[j:])
            break

    return result
```

Merge Sort

Example:

```
>>> merge_sort([6, 1, 9, 2, 7, 0])
```





Merge Sort

Example:

```
>>> merge_sort([6, 1, 9, 2, 7, 0])
```

```
======> Calling merge_sort()
List: [6, 1, 9, 2, 7, 0]
List length: 6
Middle index: 3
Left half: [6, 1, 9]
Right half: [2, 7, 0]
Calling merge_sort() for the left half: [6, 1, 9]

======> Calling merge_sort()
List: [6, 1, 9]
List length: 3
Middle index: 1
Left half: [6]
Right half: [1, 9]
Calling merge_sort() for the left half: [6]

======> Calling merge_sort()
List: [6]
List length: 1
The list is empty or it only contains one item
Returning list: [6]
Calling merge_sort() for the right half: [1, 9]

======> Calling merge_sort()
List: [1, 9]
List length: 2
Middle index: 1
Left half: [1]
Right half: [9]
Calling merge_sort() for the left half: [1]

======> Calling merge_sort()
List: [1]
List length: 1
The list is empty or it only contains one item
Returning list: [1]
Calling merge_sort() for the right half: [9]

======> Calling merge_sort()
List: [9]
List length: 1
The list is empty or it only contains one item
Returning list: [9]
Merging the two halves...
```

Merge Sort

Example:

```
---> Inside merge()
Merging...
Left half [1]
Right half [9]

-> While loop
i = 0
j = 0

Is the element left_half[i] (1) < the element right_half[j] (9)
Yes,it is!
Append the element left_half[i] (1) to the list 'result'
List 'result' before: []
List 'result' after: [1]
We've assigned one element of the left half
Incrementing the value of i to: 1

Have we reached the end of either one of the lists? Yes
We reached the end of the left half
Extending the list 'result' with the remaining item(s): [9]
Returning the list 'result': [1, 9]

Back to merge_sort()
Resulting merged list: [1, 9]
Merging the two halves...

---> Inside merge()
Merging...
Left half [6]
Right half [1, 9]

-> While loop
i = 0
j = 0

Is the element left_half[i] (6) < the element right_half[j] (1)
No, it isn't!
Appending right_half[j] (1) to the list 'result'
List 'result' before: []
List 'result' after: [1]
We've assigned one element of the right half
Incrementing the value of j to: 1
```

Merge Sort

Example:

```
Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 0
j = 1

Is the element left_half[i] (6) < the element right_half[j] (9)
Yes,it is!
Append the element left_half[i] (6) to the list 'result'
List 'result' before: [1]
List 'result' after: [1, 6]
We've assigned one element of the left half
Incrementing the value of i to: 1

Have we reached the end of either one of the lists? Yes
We reached the end of the left half
Extending the list 'result' with the remaining item(s): [9]
Returning the list 'result': [1, 6, 9]

Back to merge_sort()
Resulting merged list: [1, 6, 9]
Calling merge_sort() for the right half: [2, 7, 0]

======> Calling merge_sort()
List: [2, 7, 0]
List length: 3
Middle index: 1
Left half: [2]
Right half: [7, 0]
Calling merge_sort() for the left half: [2]

======> Calling merge_sort()
List: [2]
List length: 1
The list is empty or it only contains one item
Returning list: [2]
Calling merge_sort() for the right half: [7, 0]

======> Calling merge_sort()
List: [7, 0]
List length: 2
Middle index: 1
Left half: [7]
Right half: [0]
Calling merge_sort() for the left half: [7]
```

Merge Sort

Example:

```
======> Calling merge_sort()
List: [7]
List length: 1
The list is empty or it only contains one item
Returning list: [7]
Calling merge_sort() for the right half: [0]

======> Calling merge_sort()
List: [0]
List length: 1
The list is empty or it only contains one item
Returning list: [0]
Merging the two halves...

---> Inside merge()
Merging...
Left half [7]
Right half [0]

-> While loop
i = 0
j = 0

Is the element left_half[i] (7) < the element right_half[j] (0)
No, it isn't!
Appending right_half[j] (0) to the list 'result'
List 'result' before: []
List 'result' after: [0]
We've assigned one element of the right half
Incrementing the value of j to: 1

Have we reached the end of either one of the lists? Yes
We reached the end of the right half
Extending the list 'result' with the remaining item(s): [7]
Returning the list 'result': [0, 7]

Back to merge_sort()
Resulting merged list: [0, 7]
Merging the two halves...
```

Merge Sort

Example:

```
---> Inside merge()
Merging...
Left half [2]
Right half [0, 7]

-> While loop
i = 0
j = 0

Is the element left_half[i] (2) < the element right_half[j] (0)
No, it isn't!
Appending right_half[j] (0) to the list 'result'
List 'result' before: []
List 'result' after: [0]
We've assigned one element of the right half
Incrementing the value of j to: 1

Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 0
j = 1

Is the element left_half[i] (2) < the element right_half[j] (7)
Yes,it is!
Append the element left_half[i] (2) to the list 'result'
List 'result' before: [0]
List 'result' after: [0, 2]
We've assigned one element of the left half
Incrementing the value of i to: 1

Have we reached the end of either one of the lists? Yes
We reached the end of the left half
Extending the list 'result' with the remaining item(s): [7]
Returning the list 'result': [0, 2, 7]

Back to merge_sort()
Resulting merged list: [0, 2, 7]
Merging the two halves...
```

Merge Sort

Example:

```
---> Inside merge()
Merging...
Left half [1, 6, 9]
Right half [0, 2, 7]

-> While loop
i = 0
j = 0

Is the element left_half[i] (1) < the element right_half[j] (0)
No, it isn't!
Appending right_half[j] (0) to the list 'result'
List 'result' before: []
List 'result' after: [0]
We've assigned one element of the right half
Incrementing the value of j to: 1

Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 0
j = 1

Is the element left_half[i] (1) < the element right_half[j] (2)
Yes,it is!
Append the element left_half[i] (1) to the list 'result'
List 'result' before: [0]
List 'result' after: [0, 1]
We've assigned one element of the left half
Incrementing the value of i to: 1

Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 1
j = 1

Is the element left_half[i] (6) < the element right_half[j] (2)
No, it isn't!
Appending right_half[j] (2) to the list 'result'
List 'result' before: [0, 1]
List 'result' after: [0, 1, 2]
We've assigned one element of the right half
Incrementing the value of j to: 2
```

Merge Sort

Example:

```
Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 1
j = 2

Is the element left_half[i] (6) < the element right_half[j] (7)
Yes,it is!
Append the element left_half[i] (6) to the list 'result'
List 'result' before: [0, 1, 2]
List 'result' after: [0, 1, 2, 6]
We've assigned one element of the left half
Incrementing the value of i to: 2

Have we reached the end of either one of the lists? No
Let's continue the while loop
i = 2
j = 2

Is the element left_half[i] (9) < the element right_half[j] (7)
No, it isn't!
Appending right_half[j] (7) to the list 'result'
List 'result' before: [0, 1, 2, 6]
List 'result' after: [0, 1, 2, 6, 7]
We've assigned one element of the right half
Incrementing the value of j to: 3

Have we reached the end of either one of the lists? Yes
We reached the end of the right half
Extending the list 'result' with the remaining item(s): [9]
Returning the list 'result': [0, 1, 2, 6, 7, 9]

Back to merge_sort()
Resulting merged list: [0, 1, 2, 6, 7, 9]
[0, 1, 2, 6, 7, 9]
```