Algorithm

Merge Sort Code Walkthrough Part 1



Merge Sort

```
def merge sort(Lst):
    if len(lst) == 0 or len(lst) == 1:
        return 1st
    else:
        middle index = len(1st)//2
        left = merge_sort(lst[:middle index])
        right = merge sort(lst[middle index:])
        return merge(left, right)
```



Merge Sort





```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
```



```
def merge_sort(Lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
```

```
def merge_sort(Lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])

[5, 1, 4, 7, 3])

merge_sort([5, 1])

[5, 1]
```

```
def merge_sort(Lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])

[5, 1, 4, 7, 3])

merge_sort([5, 1])

[5, 1]
```

```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
 merge_sort([5, 1])
merge_sort([5])
```

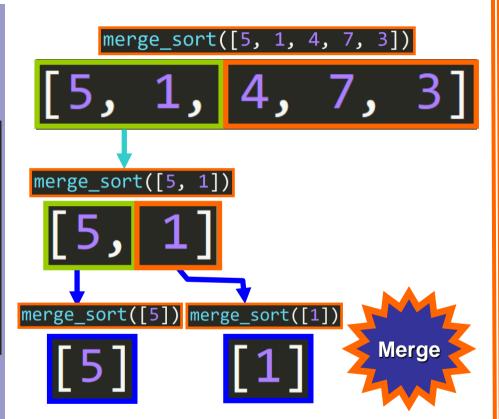


```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
 merge_sort([5, 1])
merge_sort([5]) merge_sort([1])
```



```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])

[5, 1, 4, 7, 3])

merge_sort([5, 1])

[1, 5]
```



```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])

[5, 1, 4, 7, 3]

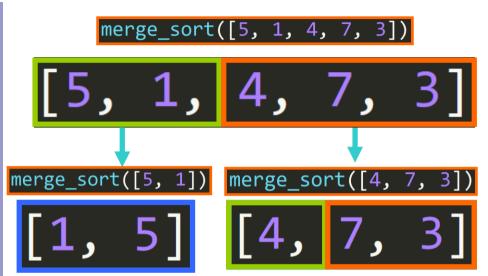
merge_sort([5, 1])

merge_sort([4, 7, 3])

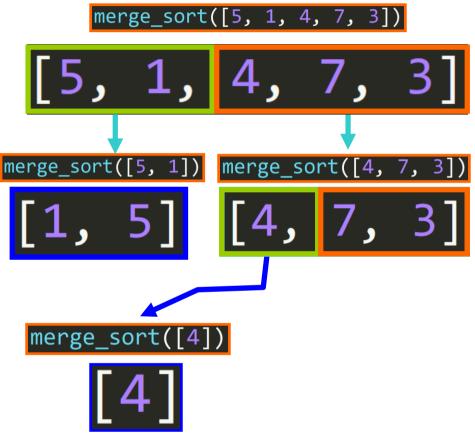
[1, 5]

[4, 7, 3]
```

```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

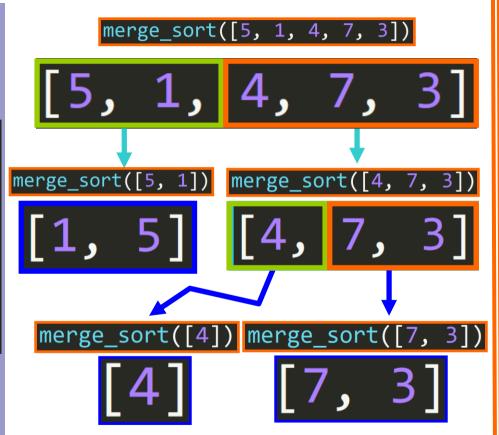


```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```



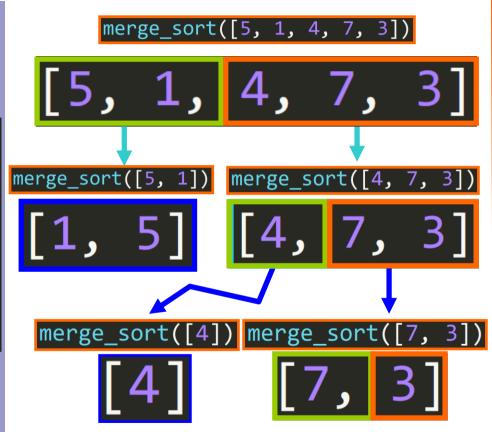


```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

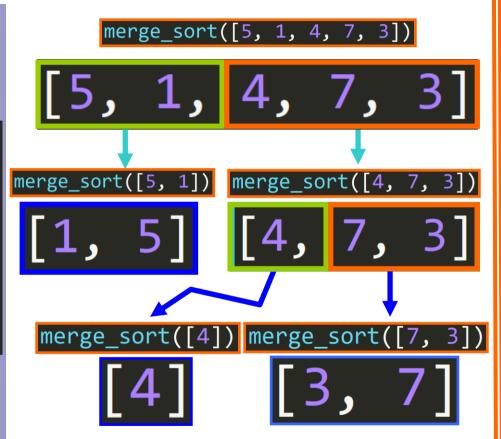


```
merge_sort([5, 1, 4, 7, 3])
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
                                                                  merge\_sort([4, 7,
                                             merge_sort([5, 1])
       return 1st
    else:
       middle_index = len(lst)//2
        left = merge sort(lst[:middle index])
       right = merge_sort(lst[middle_index:])
                                               merge_sort([4]) merge_sort([7, 3])
       return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
                                                                  merge\_sort([4, 7,
                                             merge_sort([5, 1])
        return 1st
    else:
       middle_index = len(lst)//2
        left = merge sort(lst[:middle index])
        right = merge_sort(lst[middle_index:])
                                               merge_sort([4]) merge_sort([7,
       return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
                                                                  merge\_sort([4, 7,
                                             merge_sort([5, 1])
       return 1st
    else:
       middle_index = len(lst)//2
        left = merge sort(lst[:middle index])
       right = merge_sort(lst[middle_index:])
                                               merge_sort([4]) merge_sort([7,
       return merge(left, right)
                                                               Merge
```

```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
merge_sort([5, 1, 4, 7, 3])
def merge sort(Lst):
    if len(lst) == 0 or len(lst) == 1:
                                                                         merge\_sort([4, 7,
                                                 merge_sort([5,
        return 1st
    else:
        middle_index = len(lst)//2
        left = merge sort(lst[:middle index])
        right = merge_sort(lst[middle_index:])
        return merge(left, right)
                                                                    [4]) merge_sort([7, 3])
                                              Merge
                                                            Python Searching and Sorting Algorithms: A Practical Approach
```

```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
merge_sort([5, 1, 4, 7, 3])

[5, 1, 4, 7, 3]

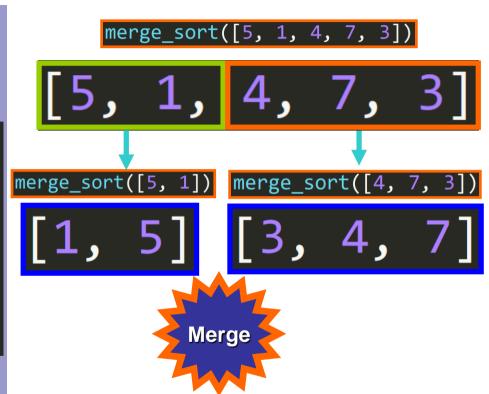
merge_sort([5, 1])

merge_sort([4, 7, 3])

[1, 5]

[3, 4, 7]
```

```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```



[1, 3, 4, 5, 7]

