Algorithm

Merge Sort Code Walkthrough Part 2





Merge Sort

```
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 left half[i] < right half[j]:</pre>
            result.append(left_half[i])
            i += 1
        else:
            result.append(right half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```



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])
```

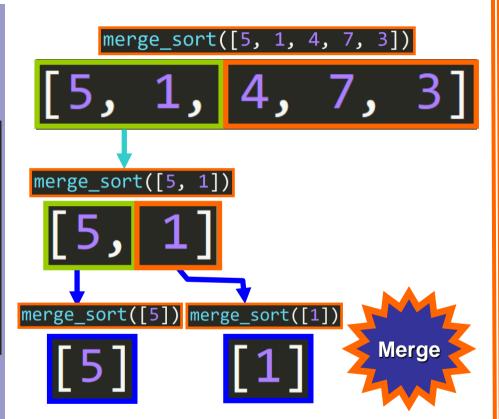


```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```

```
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(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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```





```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:]);
            break
    return result
```

$i=0 \quad j=0$

```
left_half right_half [5]
```

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

$i=0 \quad j=0$

```
left_half right_half

[5]
[1]
(i) [0]
(j) [0]
```

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

left_half

[5]

(i) [0]

right_half

(j) [0]

5 < 1?

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

result = [1, 5]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

result = [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])

[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]

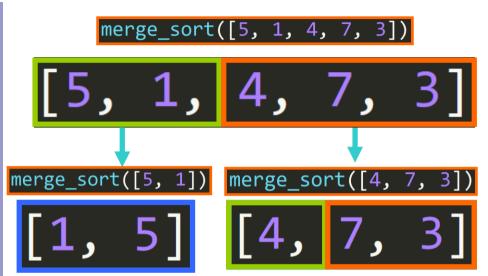
merge_sort([5, 1])

merge_sort([4, 7, 3])

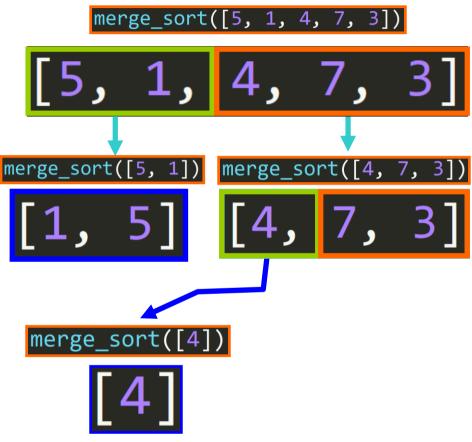
[1, 5]

[4, 7, 3]
```

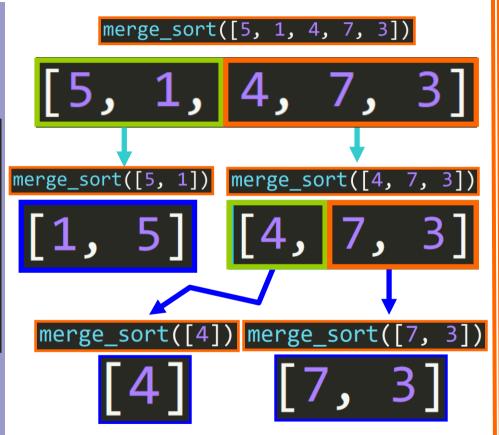
```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```



```
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)
```

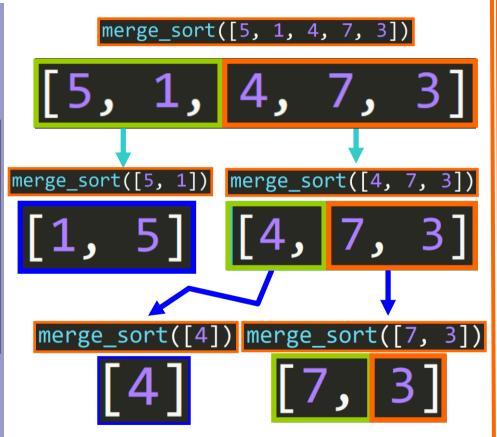


```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
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)
```

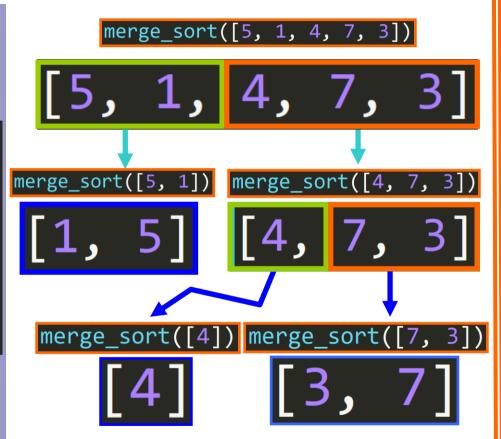


```
merge_sort([5, 1, 4, 7, 3])
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
                                                                  merge\_sort([4, 7,
                                             merge_sort([5, 1])
       return 1st
    else:
       middle_index = len(lst)//2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

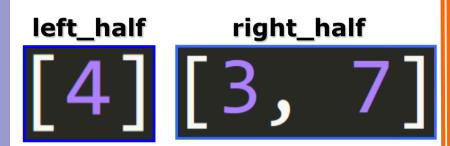
```
def merge_sort(lst):
    if len(lst) == 0 or len(lst) == 1:
        return lst
    else:
        middle_index = len(lst)//2
        left = merge_sort(lst[:middle_index])
        right = merge_sort(lst[middle_index:])
    return merge(left, right)
```





```
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(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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```



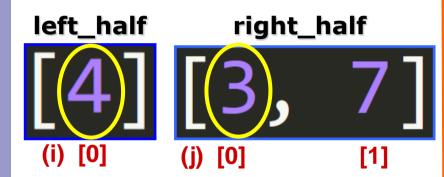
```
def merge(left_half, right_half):
    if not left half or not right half:
        return left half or right half
    result = []
    i, j = 0, 0
    while True:
        if left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

```
      left_half
      right_half

      [4]
      [3, 7]

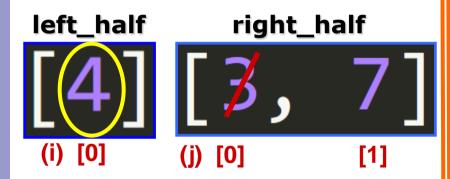
      (i) [0]
      (j) [0]
      [1]
```

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

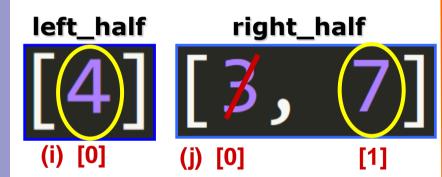


4 < 3?

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```



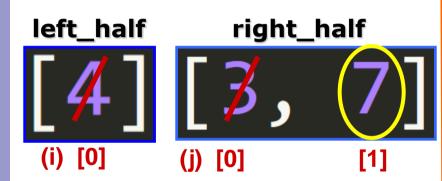
```
def merge(left_half, right_half):
    if not left half or not right half:
        return left half or right half
    result = []
    i, j = 0, 0
    while True:
        if left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```



4 < 7?

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

i=1 j=1

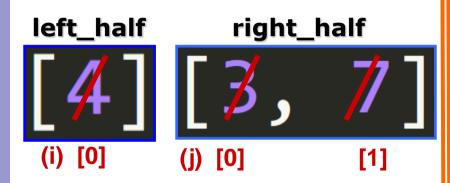


4 < 7?

result = [3, 4]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

i=1 j=1



result = [3, 4, 7]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right_half[j])
            j += 1
        if i == len(left half) or j == len(right half):
            result.extend(left_half[i:] or right_half[j:]);
            break
    return result
```

i=1 j=1

```
[3, 4, 7]
```

result = [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)
```

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

[5, 1, 4, 7, 3]

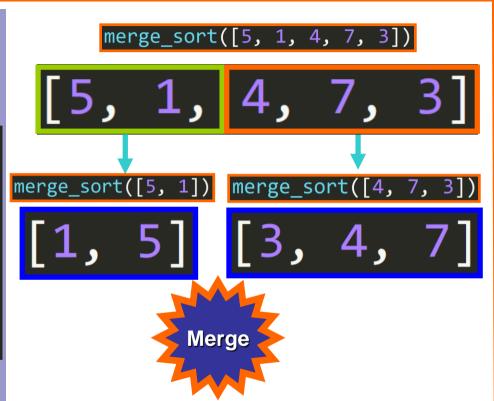
merge_sort([5, 1])

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(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 left_half[i] < right_half[j]:</pre>
             result.append(left_half[i])
            i += 1
            result.append(right_half[j])
             j += 1
        if i == len(left_half) or j == len(right_half):
             result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

```
left_half right_half
[1, 5] [3, 4, 7]
```

```
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 left half[i] < right half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left_half[i:] or right_half[j:])
            break
    return result
```

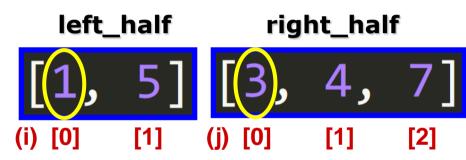
i = 0 j = 0

```
left_halfright_half\begin{bmatrix} 1, 5 \end{bmatrix}\begin{bmatrix} 3, 4, 7 \end{bmatrix}(i) [0] [1] (j) [0] [1] [2]
```

result = []

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

$i = 0 \quad j = 0$



1 < 3?

result = []

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

$i=1 \quad j=0$

```
left_half right_half

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

(i) [0] [1] (j) [0] [1] [2]
```

result = [1]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

result = [1]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left_half[i])
            i += 1
            result.append(right half[j])
            j += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

```
left_half right_half

[17, 5] [3, 4, 7]

(i) [0] [1] (j) [0] [1] [2]
```

result = [1, 3]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

5 < 4?

result = [1, 3]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

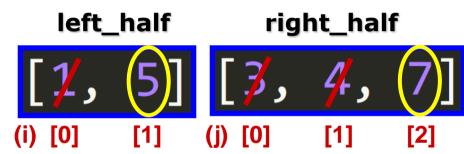
```
left_half right_half

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

(i) [0] [1] (j) [0] [1] [2]
```

result = [1, 3, 4]

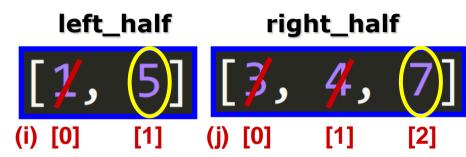
```
def merge(left half, right half):
    if not left_half or not right_half:
        return left half or right half
    result = []
    i, j = 0, 0
    while True:
        if left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left half) or j == len(right half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```



5 < 7?

result = [1, 3, 4]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left half) or j == len(right half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```



5 < 7?

result = [1, 3, 4, 5]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

$i=2 \quad j=2$

result = [1, 3, 4, 5]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left half) or j == len(right half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

i=2 j=2

```
      left_half
      right_half

      [1/3, 4/3, 7]

      (i) [0]
      [1]
      (j) [0]
      [1]
      [2]
```

result = [1, 3, 4, 5, 7]

```
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 left_half[i] < right_half[j]:</pre>
            result.append(left half[i])
            i += 1
            result.append(right half[j])
            i += 1
        if i == len(left_half) or j == len(right_half):
            result.extend(left half[i:] or right half[j:])
            break
    return result
```

$$i=2$$
 $j=2$

result = [1, 3, 4, 5, 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)
```



[1, 3, 4, 5, 7]



